

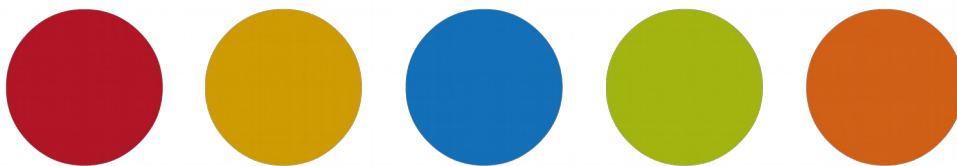
# Weather-App

## Skript 2016

Konfigurations- und Schulungsunterlagen

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

Stand: 12. Jun 2016



© Christine Janischek

## Inhaltsverzeichnis

1 Allgemeines.....	3
2 Das Projekt Weather.....	5
2.1 Überblick.....	5
2.2 Grundlagen: Projekt erzeugen.....	6
2.3 View: Layouts, Komponenten & XML für die Benutzeroberfläche.....	10
2.4 Modell: Implementierung der Fachklassen für die Datenhaltung.....	28
2.5 Controller: Daten anzeigen und aktualisieren.....	38

## 1 Allgemeines



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

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

Skript & Sources für die Projekte (für Fortgeschrittene):

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



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

Weitere Skripte und Sources online:

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

[Fortgeschrittene Apps mit Android Studio erstellen](#)

[Android Apps erstellen](#)

[Java Programmieren im Unterricht](#)

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

[Objektorientierte Sytementwicklung in Java](#)

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

[Webprogrammierung im Unterricht](#)

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

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

[Relationale Datenbanken](#)

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

[BUC01]

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

[KUE01]

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

[WAC00]

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

[FLE00]

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

[HAA00]

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

[AZF00]





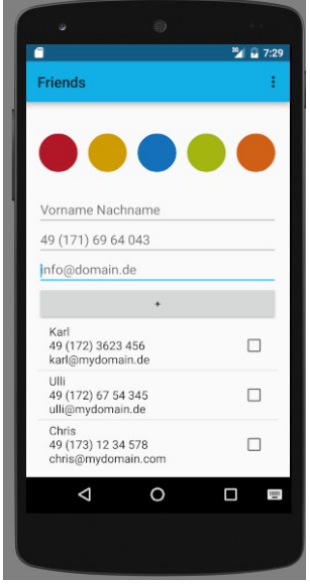
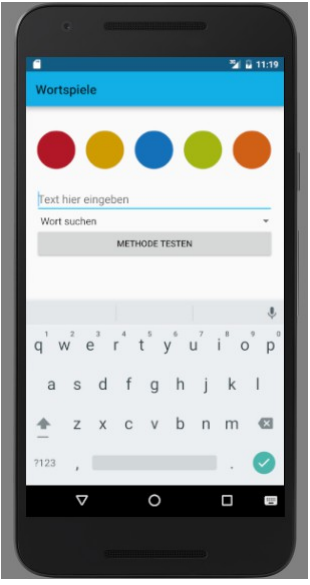
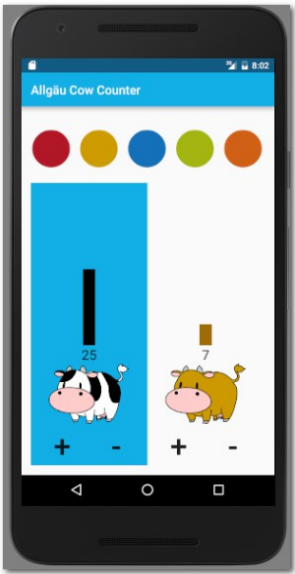
Azzola, Francesco „Android: Build real weather app: JSON, HTTP and Openweathermap“, <https://www.javacodegeeks.com/2013/06/android-build-real-weather-app-json-http-and-openweathermap.html>, 2013, zuletzt getestet am 30.04.2016

## 2 Das Projekt Weather

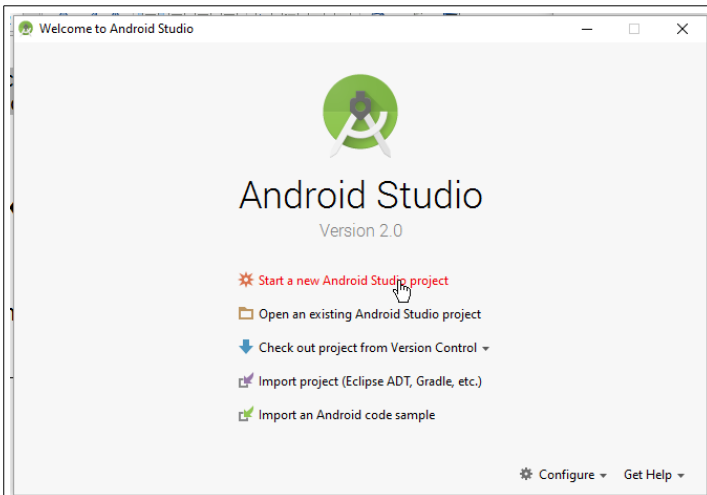
### 2.1 Überblick

Weather App:

Das Projekt soll an einer einfachen Benutzeroberfläche zeigen auf welche Weise eine entfernte Datenquelle, wie der [OpenWeatherMap API](#), für eine mobile Endapplikation genutzt werden können. Die Weather App ermöglicht die aktuelle Anzeige der Wetterdaten für eine Stadt und die Aktualisierung und Änderung der Städte-Angabe über die Benutzeroberfläche (Menü, Dialog) der Anwendung.

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

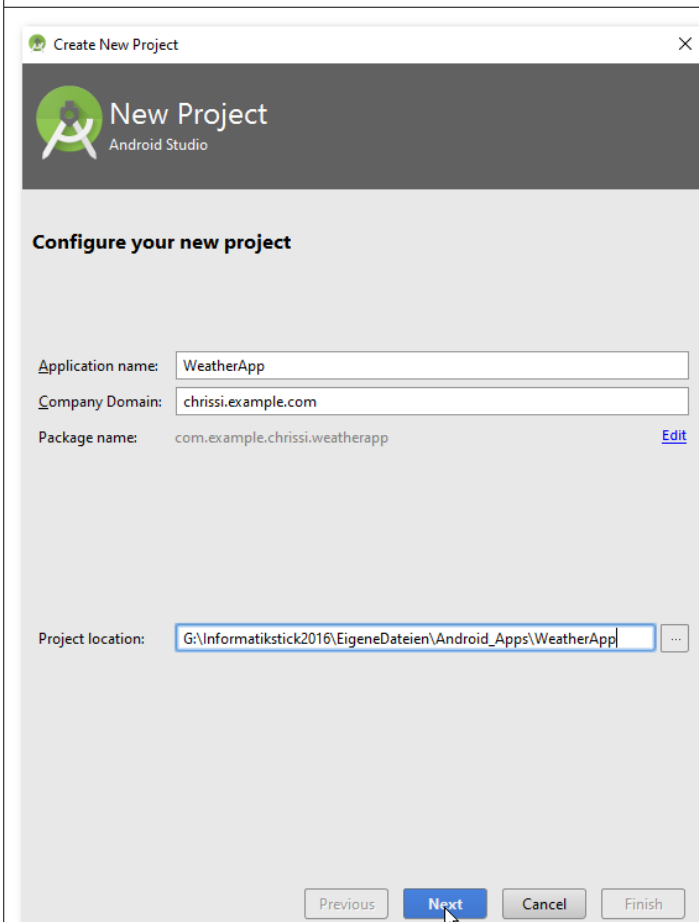
## 2.2 Grundlagen: Projekt erzeugen



### *Ein Neues Projekt erzeugen.*

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

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



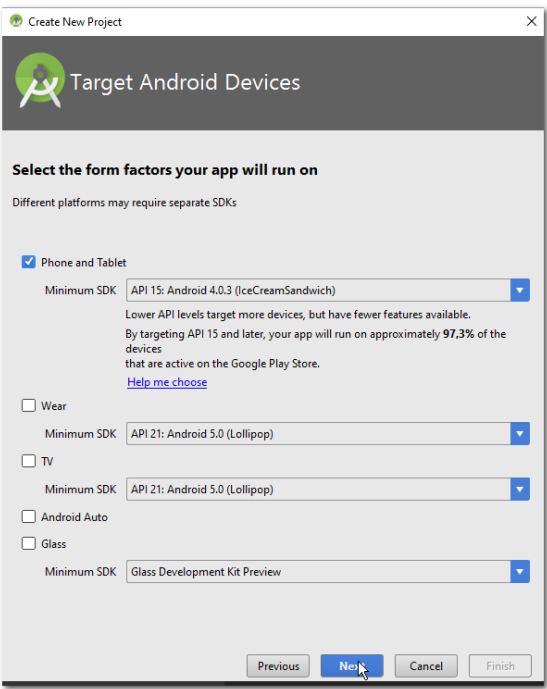
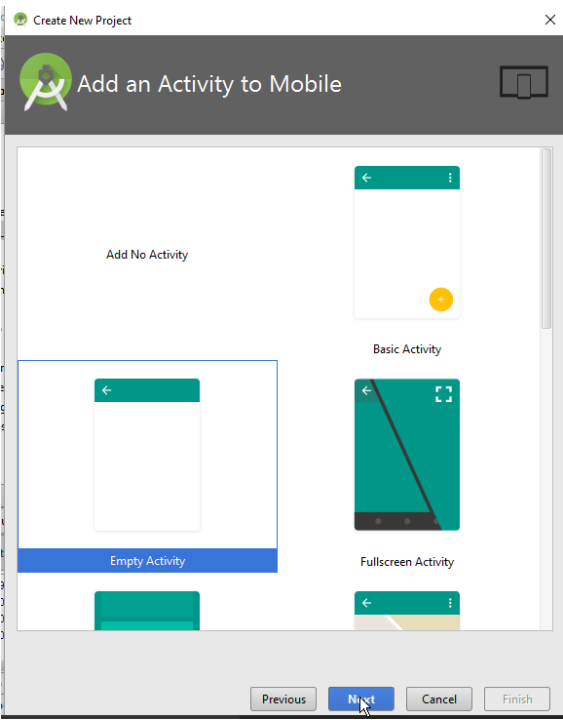
### *Legen Sie nun schrittweise die Eigenschaften für Ihr neues Android-Projekt fest.*

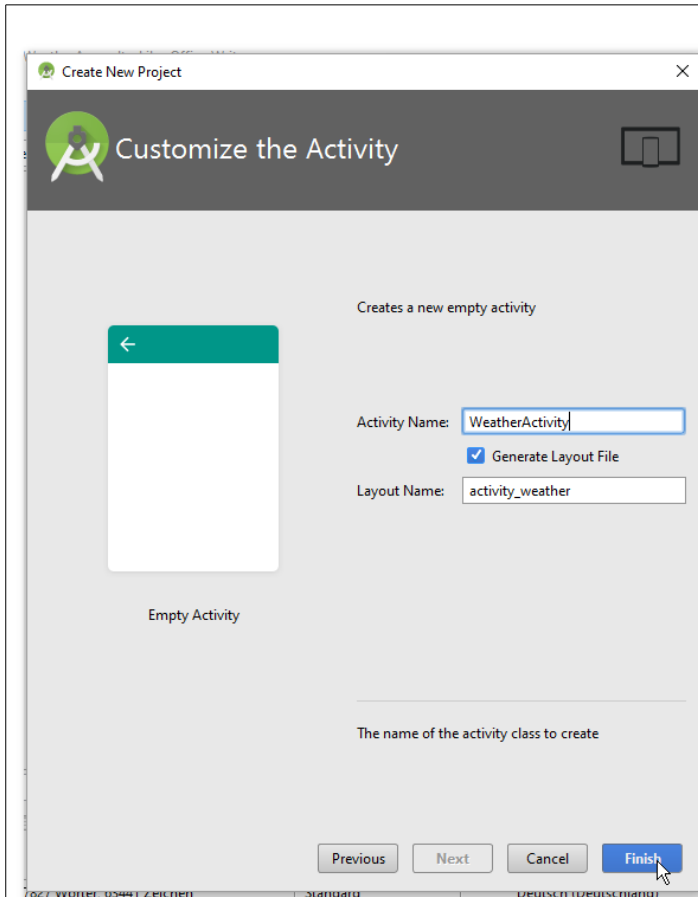
Geben Sie dazu die nebenstehend angezeigten Angaben für

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

G:\Informatikstick2016\EigeneDateien\Android\_Apps\WeatherApp

Je nach Konfiguration können diese Angaben variieren

	<p><i>Laufzeitumgebung unserer Anwendung.</i></p> <p>Wir wählen als Ziel unserer Anwendung das API Level, mit der höchsten Abdeckung für die Lauffähigkeit auf verfügbaren Android Geräten, aus.</p> <p>Der Assistent macht uns dazu einen Vorschlag für Telefone und Tablets.</p> <p>Wir nehmen den Vorschlag an und klicken auf die Schaltfläche → Next.</p>
	<p><i>Aktivität wählen.</i></p> <p>Im ersten Schritt nutzen wir die einfachste Form zur Steuerung von Ereignissen. Die → Empty Activity. Wählen wir diese Aktivität bekommen wir einige Standards mitgeliefert.</p> <p><b>Wir wählen die → Empty Activity und klicken Sie auf die Schaltfläche → Next.</b></p> <p>Hinweis: Alternativ können wir auch die Option → Add No Activity wählen und können dann nachträglich alle Maßnahmen für die Implementierung der Activity selber treffen.</p>



### 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.

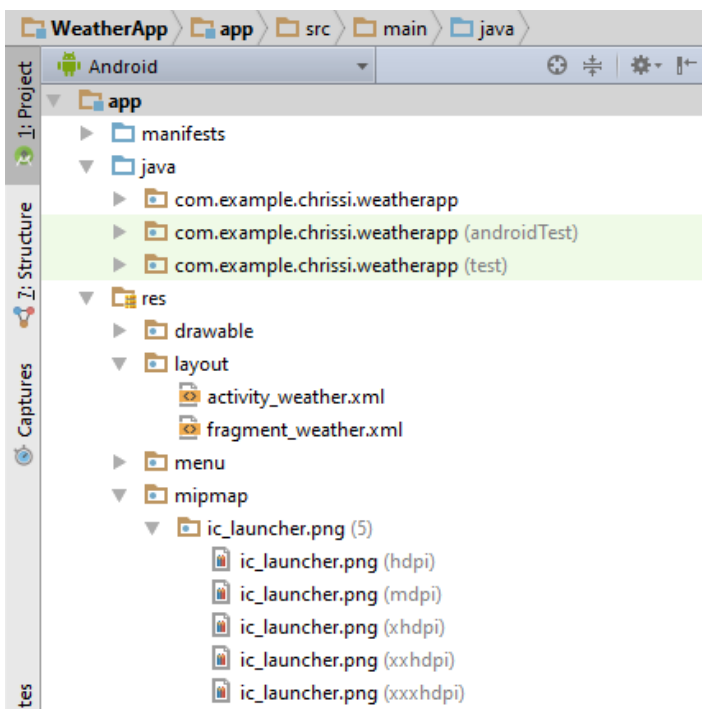
Mit dem Klick auf → Finish wird die Projektstruktur (Architektur) erzeugt.

### Hinweis:

Je nach Rechnerausstattung kann die Erzeugung einen Moment dauern.

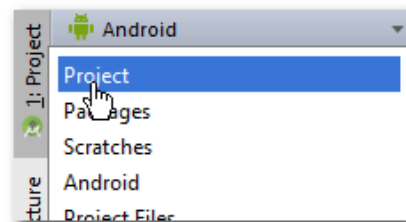
Android Studio nutzt u.a. das Gradle-PlugIn als Buildsystem. Gradle ist dabei ein Werkzeug das komplett in Android Studio integriert ist und zur Build-Automatisierung und - Management genutzt wird. Jede Anwendung muss nach jeder Änderungen im Quellcode neu erzeugt werden, dabei werden außer der Kompilierung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

### Android View



### Projektstruktur am Anfang.

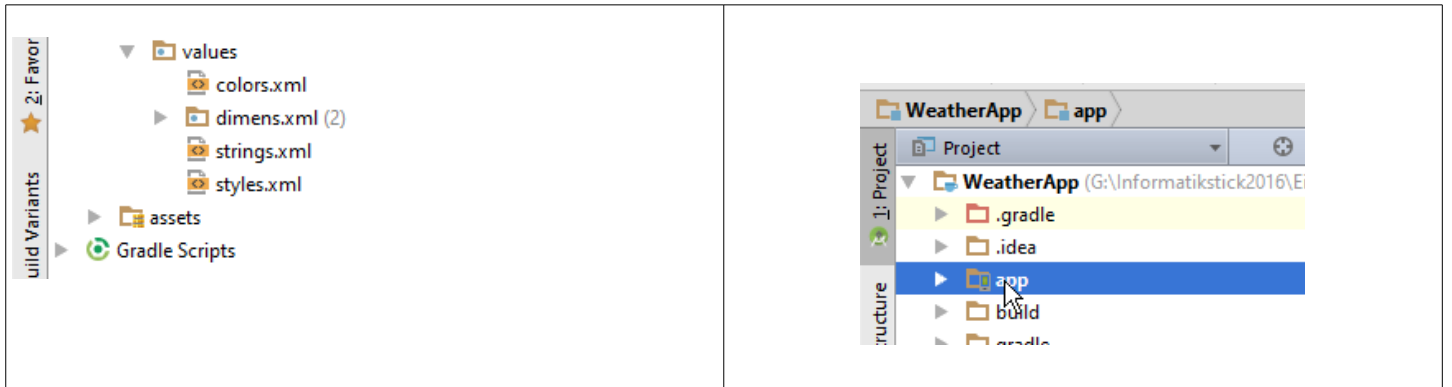
Im Anschluss an den abgeschlossenen Build-Prozess finden Sie im linken Frame die folgende Projektstruktur vor.



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

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





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



Gewünschtes Ergebnis

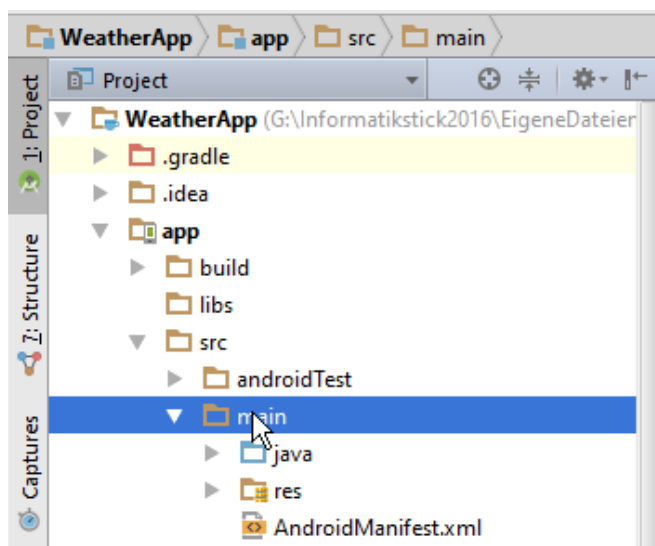
*Vorgehensweise erläutern.*

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

1. Schrift-Datei einbetten
2. Bezeichner (Strings) deklarieren und initialisieren
3. Farben deklarieren und initialisieren
4. Angaben im Manifest anpassen
5. Activity-Layout anpassen
6. Fragment-Layout erstellen
7. Menü-Eintrag hinzufügen

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

Ansicht: Projekt



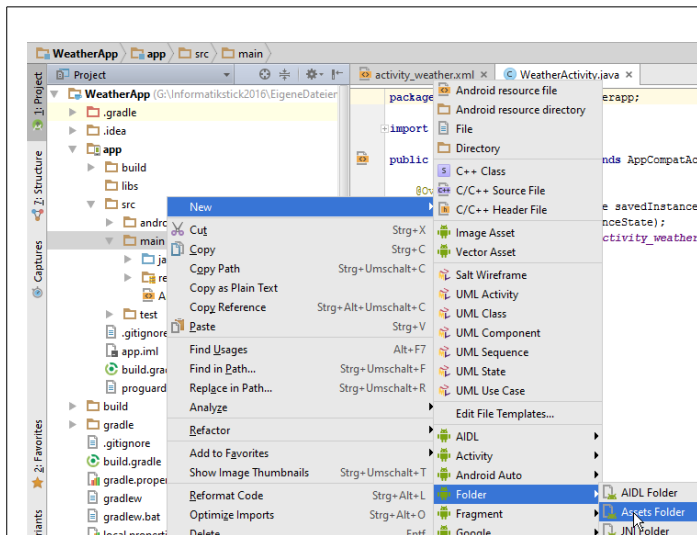
*Schriftdeklaration einbetten.*

Wir verwenden eine Schrift die zusätzlich auch eine Deklaration der Weather-Icons zur Verfügung stellt.

**Klappen Sie dazu das → app Verzeichnis auf und wählen Sie das Verzeichnis → main aus.**

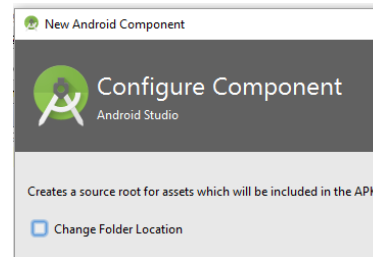
Quelle: Eric Flowers

<https://github.com/erikflowers/weather-icons/tree/master/font>

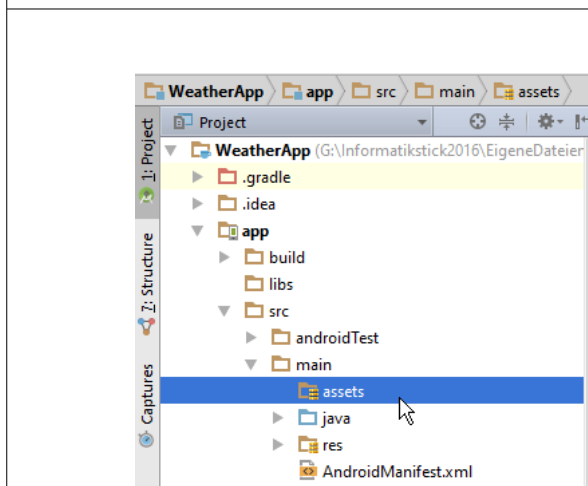


Assets-Verzeichnis erstellen.

Wählen Sie dann im Kontext-Menü (rechte Maustaste) die Optionen → New → Folder → Assets Folder.



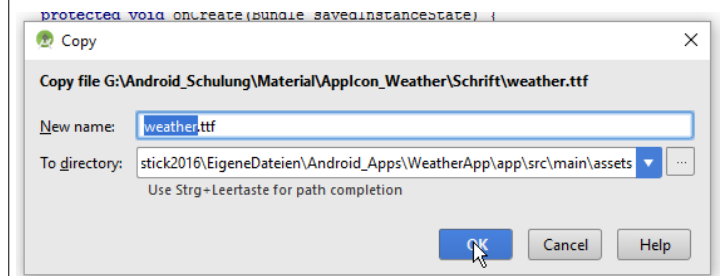
Belassen Sie Einstellung und klicken Sie auf die Schaltfläche → Finish.



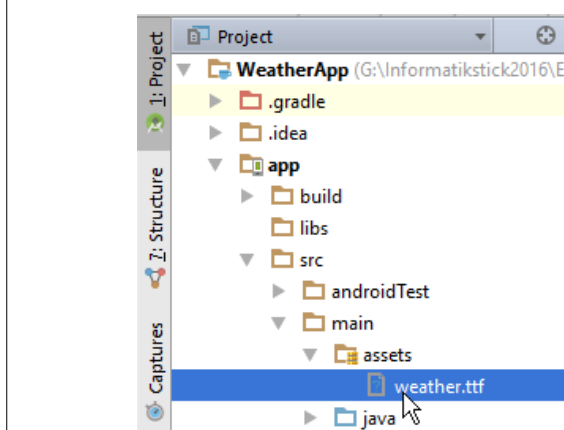
Schrift-Datei einbetten.

Kopieren Sie die Schrift-Datei (weather.ttf) mit der Tastenkombination STRG + C.

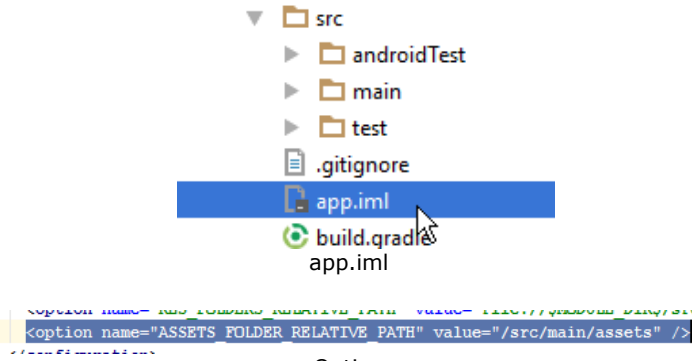
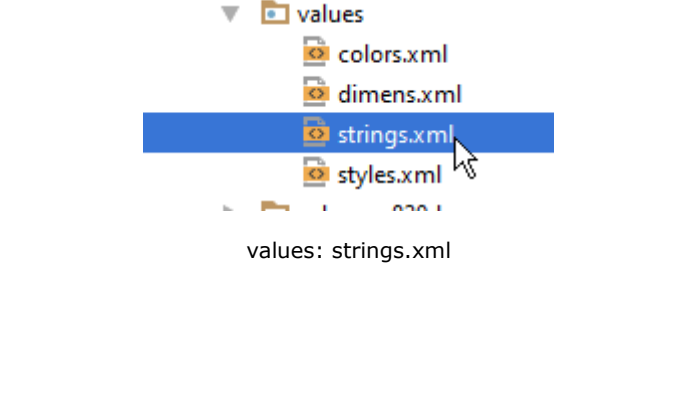
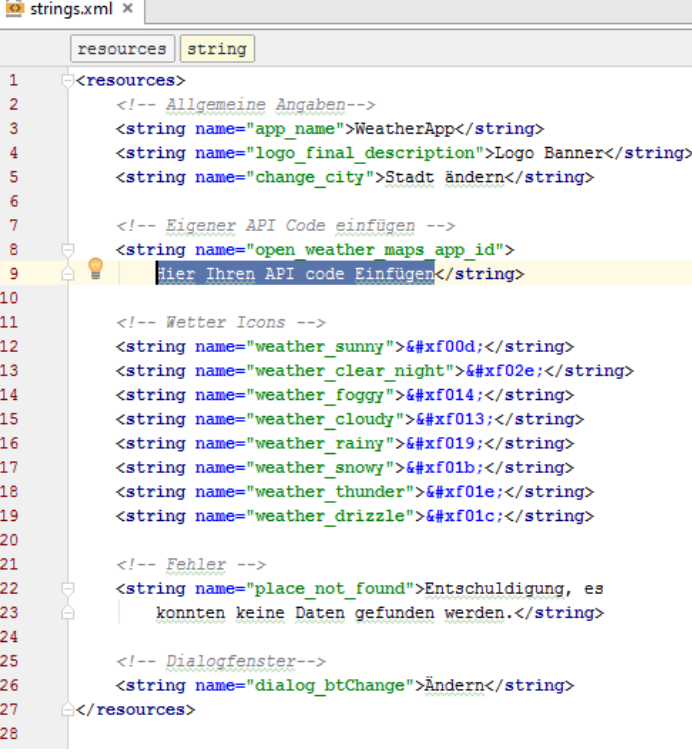
Fügen Sie danach die Datei mit der Tastenkombination STRG + V in das gerade erstellte Unterverzeichnis → assets ein.

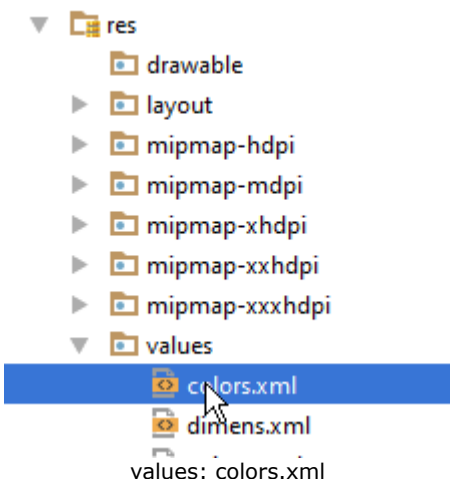
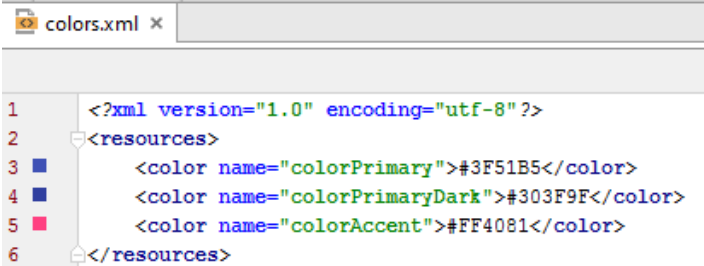


Klicken Sie dann abschließend auf die Schaltfläche → OK



Der Pfad für die neue Schrift sollte mit der Einbettung der Datei in das Verzeichnis übernommen werden.

 <p>Option</p>	<p><i>Pfad einer Option prüfen.</i></p> <p>Prüfen Sie ob der Pfad für das assets-Verzeichnis in der Datei → app.iml ergänzt wurde. Öffnen Sie die Datei und prüfen Sie die Optionen:</p> <p>Diese Pfadangabe sollte vorhanden sein:</p> <pre>&lt;option name="ASSETS_FOLDER_RELATIVE_PATH" value="/src/main/assets" /&gt;</pre>
 <p>values: strings.xml</p>	<p><i>Bezeichner (Strings) deklarieren und initialisieren.</i></p> <p>Öffnen Sie dazu im Verzeichnis → app → res → values die Datei strings.xml mit einem Doppelklick auf den Dateinamen.</p> <p>Ergänzen Sie den fehlenden Quellcode.</p>
 <p>API Code Example: 98149f523e1f1b926ca7cd6b9ce77cff</p>	<p><i>Bezeichner definieren.</i></p> <p>Für die Allgemeinen Angaben definieren wir dazu, wie folgt:</p> <pre>&lt;string name="app_name"&gt;WeatherApp&lt;/string&gt; &lt;string name="logo_final_description"&gt;   Logo Banner&lt;/string&gt; &lt;string name="change_city"&gt;   Stadt ändern&lt;/string&gt;</pre> <p>Ohne eine Autorisierung erhalten wir keinen Zugriff auf die Wetterdaten. (Error: 401) Den API Code können wir auf der Seite OpenWeatherMap kostenlos erzeugen (→ signIn notwendig, → <a href="https://home.openweathermap.org/users/sign_up">https://home.openweathermap.org/users/sign_up</a>). Übernehmen Sie den Quellcode und fügen Sie <b>ihren eigenen API Code</b> ein.</p> <p>Ergänzen Sie den Quellcode und setzen Sie Ihren API ein:</p> <pre>&lt;!-- Eigener API Code einfügen --&gt; &lt;string name="open_weather_maps_app_id"&gt;   Hier Ihren API code einfügen&lt;/string&gt;</pre> <p><b>Fügen Sie einen gültigen API Code ein!</b></p> <p>WetterIcons setzen:</p>

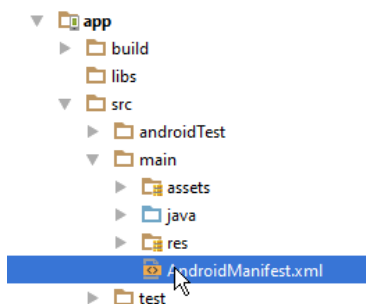
	<p>Enthalten die Unicodes, um die WetterIcon erzeugen (rendern) zu können.</p> <pre>&lt;string name="weather_sunny"&gt;&amp;#xf00d;&lt;/string&gt; &lt;string name="weather_clear_night"&gt;&amp;#xf02e;&lt;/string&gt; &lt;string name="weather_foggy"&gt;&amp;#xf014;&lt;/string&gt; &lt;string name="weather_cloudy"&gt;&amp;#xf013;&lt;/string&gt; &lt;string name="weather_rainy"&gt;&amp;#xf019;&lt;/string&gt; &lt;string name="weather_snowy"&gt;&amp;#xf01b;&lt;/string&gt; &lt;string name="weather_thunder"&gt;&amp;#xf01e;&lt;/string&gt; &lt;string name="weather_drizzle"&gt;&amp;#xf01c;&lt;/string&gt;</pre> <p>Fehlermeldungen setzen:</p> <pre>&lt;string name="place_not_found"&gt;Entschuldigung, es konnten keine Daten gefunden werden.&lt;/string&gt;</pre> <p>Schaltflächenbezeichnung im Dialogfenster setzen:</p> <pre>&lt;string name="dialog_btChange"&gt;Ändern&lt;/string&gt;</pre> <p>Ergänzen Sie den Quellcode, wie nebenstehend angezeigt.</p>
 <p>values: colors.xml</p>	<p><i>Farben deklarieren und initialisieren.</i></p> <p>Öffnen Sie dazu im Verzeichnis → app → res → values die Datei colors.xml mit einem Doppelklick auf den Dateinamen.</p>
 <p>Vorher</p>	<p><i>Änderung der Farben durchführen</i></p> <p>Hexadezimalcodes für die verwendeten Farben:</p> <pre>colorPrimary: #ff11afe5 colorPrimaryDark: #ff125c84 colorAccent: #ff11c4ff</pre> <p>Für die Hintergrundfarbe:</p> <pre>&lt;color name="background"&gt;#ff11c4ff&lt;/color&gt;</pre> <p>Ergänzen Sie den Quellcode, wie nebenstehend angezeigt.</p>

```

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

```

Nachher



Android Manifest

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3  package="com.example.chrissi.weatherapp">
4
5  <!-- Rechte setzen -->
6  <uses-permission android:name="android.permission.INTERNET"/>
7  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
8  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
9
10 <!-- Eigenschaften festlegen -->
11 <application
12     android:allowBackup="true"
13     android:icon="@mipmap/ic_launcher"
14     android:label="@string/app_name"
15     android:supportsRtl="true"
16     android:theme="@style/AppTheme">
17
18     <!-- Activity Ref, Name, Orientierung festlegen -->
19     <activity android:name="com.example.chrissi.weatherapp.WeatherActivity"
20         android:label="@string/app_name"
21         android:screenOrientation="portrait">
22
23         <intent-filter>
24             <action android:name="android.intent.action.MAIN" />
25             <category android:name="android.intent.category.LAUNCHER" />
26         </intent-filter>
27     </activity>
28 </application>
29
30 </manifest>

```

Inhalt des Manifests

Angaben im Manifest anpassen.

Öffnen Sie dazu die Datei app → src → main → AndroidManifest.xml mit einem Doppelklick auf den Dateinamen.

Fügen Sie die fehlenden Eigenschaften ein.

Rechte setzen:

```

<uses-permission
    android:name="android.permission.INTERNET"/>

<uses-permission
    android:name
    ="android.permission.WRITE_EXTERNAL_STORAGE"/>

<uses-permission
    android:name
    ="android.permission.ACCESS_NETWORK_STATE"/>

```

Eigenschaften für die Anwendung setzen:

```

<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">

```

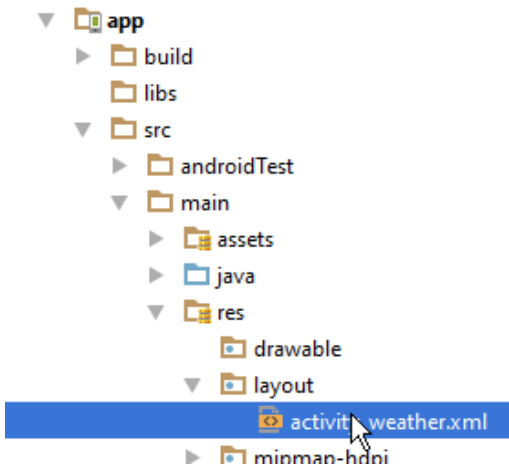
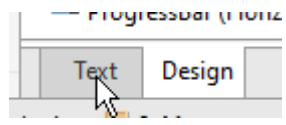
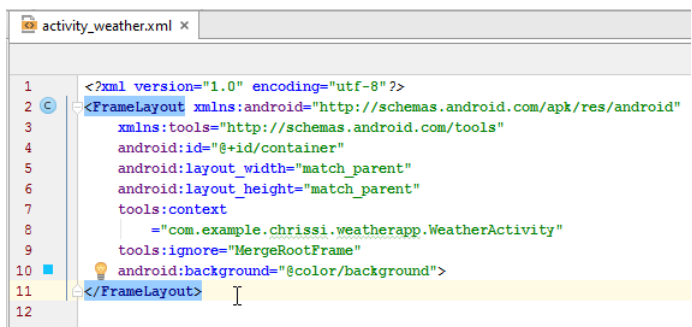
Activity, Name und Orientierung setzen:

```

<activity android:name
    ="com.example.chrissi
    .weatherapp.WeatherActivity"
    android:label="@string/app_name"
    android:screenOrientation="portrait">

    <intent-filter>
        <action android:name
            ="android.intent.action.MAIN" />
        <category android:name
            ="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

```

	<code>&lt;/application&gt;</code>
 <p>Activity Layout</p> <p>Das Relative Layout: Die in einem relativen Layout enthaltenen Komponenten werden immer in Abhängigkeit seiner direkt benachbarten Komponenten betrachtet. Deshalb erfolgt die Beschreibung der Platzierung auch in Abhängigkeit der direkt benachbarten Komponenten.</p> <p>Das Lineare Layout (vertikal): Die in einem vertikalen Linearen Layout platzierten Komponenten werden untereinander angeordnet.</p> <p>Das Lineare Layout (horizontal): Die in einem horizontalen Linearen Layout platzierten Komponenten werden nebeneinander angeordnet.</p> <p>Das Frame Layout: Die platzierten Komponenten können ausgehend vom linken oberen Rand ausgerichtet werden.</p>	<p><i>Layout der Activity anpassen.</i></p> <p>Für die erste Benutzeroberfläche:</p> <p>Öffnen Sie dazu die Datei <code>app</code> → <code>src</code> → <code>main</code> → <code>res</code> → <code>layout</code> → <code>activity_weather.xml</code> mit einem Doppelklick auf den Dateinamen.</p> <p>Wechseln Sie in den XML-Editor:</p>  <p>Klicken Sie dazu unterhalb des Designers auf den Reiter → Text.</p> <p>Das Layout für die Activity beschränkt sich auf wenige Angaben. Im folgenden wird beschrieben welche Änderungen erfolgen sollten.</p>
 <p>Activity Layout</p>	<p><i>FrameLayout verwenden.</i></p> <p>Das Activity Layout bildet den Rahmen der Anwendung. Wir verwenden dazu ein <code>FrameLayout</code> und setzen fehlende Eigenschaftswerte.</p> <p>Entfernen Sie dazu die <code>TextView</code> für das „Hallo Welt“ entfernen und ändern Sie die Angaben ab, wie nebenstehend angezeigt.</p>

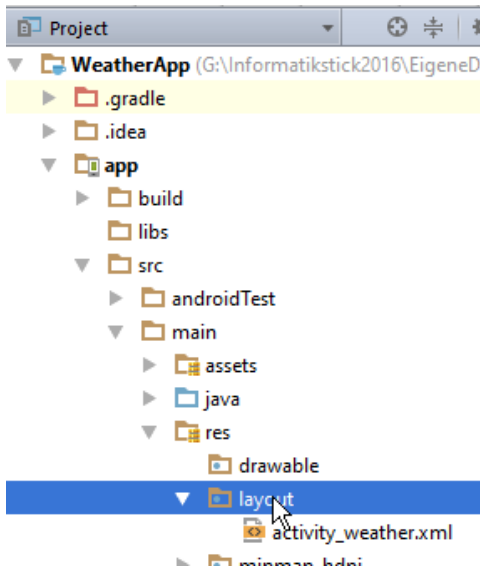


Eingabehilfe:

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android
    ="http://schemas.android.com/apk/res/android"

    xmlns:tools
    ="http://schemas.android.com/tools"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context
    ="com.example.chrissi.weatherapp.WeatherActi-
    vity"

    tools:ignore="MergeRootFrame"
    android:background="@color/background">
</FrameLayout>
```



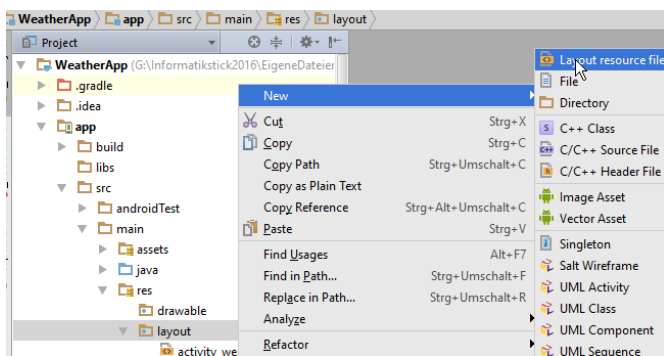
Verzeichnis Layout

*Layout des Fragments erstellen.*

Für die Anzeige der Wetterdaten werden wir eine weitere xml-Datei erzeugen:

Wählen Sie dazu das Verzeichnis in app → src → main → res → layout.

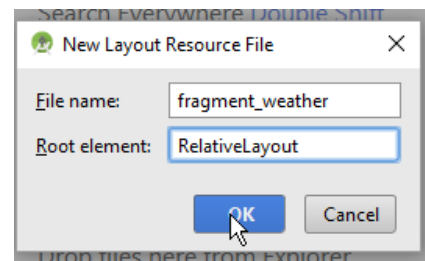
Klicken Sie dann mit der rechten Maustaste im Linken Frame auf das Unterverzeichnis → layout.



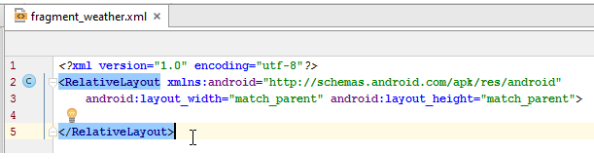

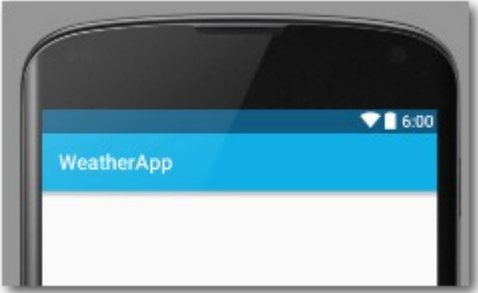
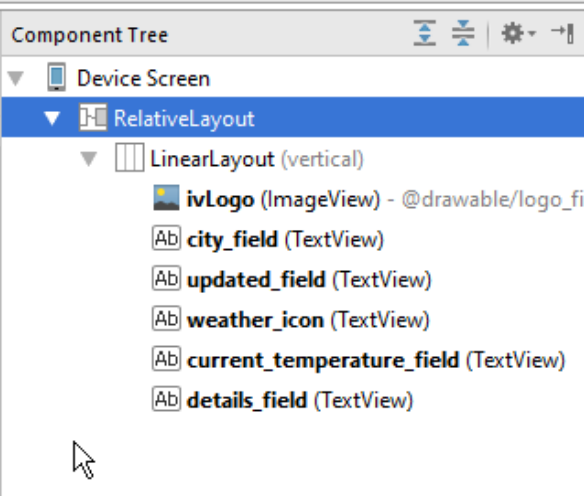
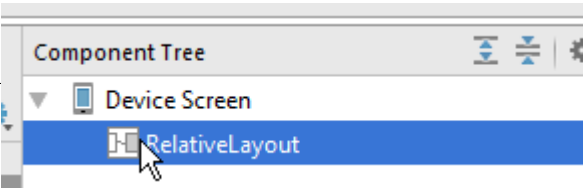
Kontext-Menü

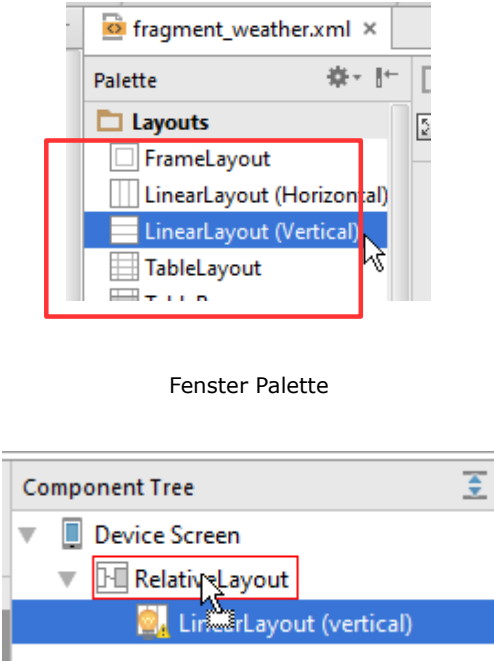
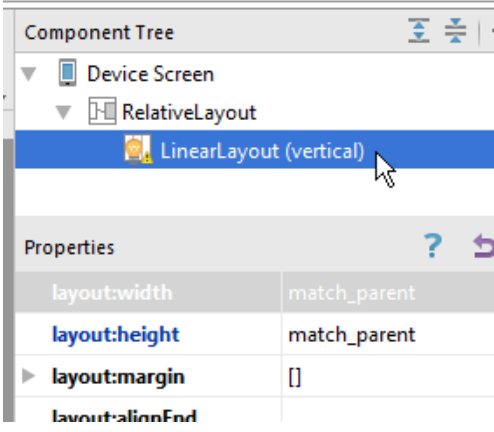
*Neue Layout-Resource erzeugen.*

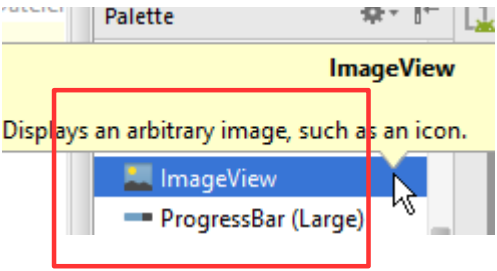
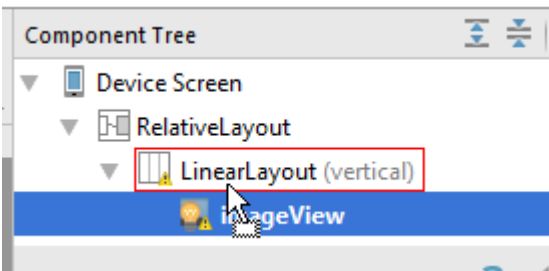
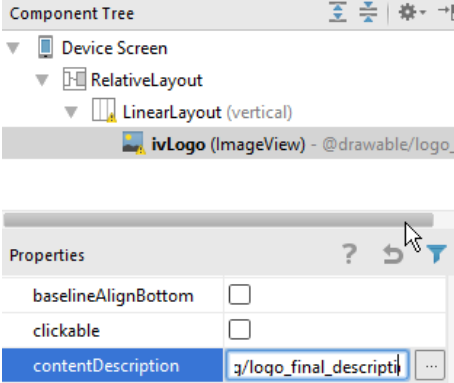
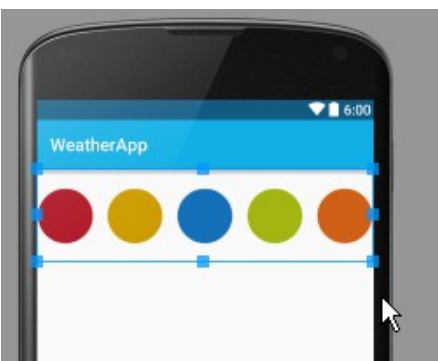
Wählen Sie dann im Kontext-Menü (rechte Maustaste) die Optionen → New → Layout resource file

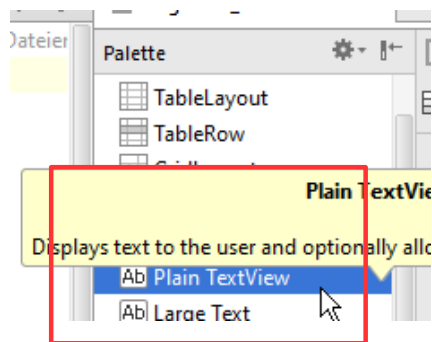




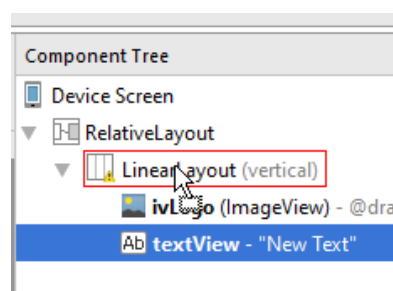
 <p>fragment_weather.xml</p>	<p>Übernehmen Sie die Angaben, angezeigt und klicken Sie auf die Schaltfläche → OK.</p>
 <p>Designer</p> <p>Hinweis: Die Anwendung besitzt ähnlich, wie in Eclipse der Swing-Designer einen Quellcode-Generator. Im Gegensatz zu Eclipse erzeugt der Quellcode-Generator in Android Studio XML-Quellcode. Wir können jederzeit zwischen den Ansichten → Text und → Design wechseln.</p>	<p><i>In den Design-Modus wechseln.</i></p> <p>Um das Design zu erstellen nutzen wir den Oberflächendesigner.</p> <p>Klicken Sie dazu auf den Reiter → Design unterhalb des angezeigten XML-Quellcodes.</p>  <p>View im Designer</p>
 <p>Gewünschtes Ergebnis</p>	<p><i>Vorgehensweise: Component Tree.</i></p> <ol style="list-style-type: none"> <li>1. Layout (falls nötig) schachteln</li> <li>2. Komponenten im Layout platzieren</li> <li>3. Komponenteneigenschaften definieren</li> </ol> <p>Nun folgen die Änderungen im aktuellen Komponenten-Baum, um das nebenstehende gewünschte Ergebnis zu erzeugen.</p>
	<p><i>Der Komponenten-Baum.</i></p> <p>Im oberen, rechten Frame-Fenster wird der</p>

<p>Aktueller Komponenten-Baum</p>	<p>Komponenten-Baum (Component Tree) angezeigt.</p> <p>Als Komponenten werden alle Elemente einer Benutzeroberfläche bezeichnet.</p> <p>Die Grundlage jeder Benutzeroberfläche sind die Layouts.</p> <p>Das Standard-Layout ist das → Relative Layout.</p>						
 <p>Fenster Palette</p> <p>Fenster Component Tree</p>	<p><i>LinearesLayout (Vertical) verwenden.</i></p> <p>Wählen Sie dazu im linken Frame-Fenster → Palette neben der Design-Bühne auf die Option → "LinearLayout (Vertical)".</p> <p>Ziehen Sie dann diese Komponente mit gedrückter linker Maustaste in das rechte, obere Frame-Fenster → Component Tree, wie nebenstehend angezeigt. Lassen Sie dann die Maustaste los.</p>						
 <p>Fenster Component Tree und Properties</p>	<p><i>Eigenschaften des Layouts ändern.</i></p> <p>Klicken Sie dazu im Fenster → Component Tree auf das → "LinearLayout (vertical)".</p> <p>Prüfen Sie dann die nebenstehend angezeigten Eigenschaften der Komponente im darunterliegenden Fenster → Properties ab.</p> <p>Eigenschaften:</p> <table border="1" data-bbox="801 1697 1524 1803"> <tr> <td>layout:width:</td> <td>match_parent</td> </tr> <tr> <td>layout:height:</td> <td>match_parent</td> </tr> <tr> <td>orientation:</td> <td>vertical</td> </tr> </table>	layout:width:	match_parent	layout:height:	match_parent	orientation:	vertical
layout:width:	match_parent						
layout:height:	match_parent						
orientation:	vertical						

 <p>Fenster Palette</p>  <p>Fenster Component Tree</p>	<p><i>ImageView-Komponente verwenden.</i></p> <p>Wählen Sie dazu im linken Frame-Fenster → Palette neben der Design-Bühne auf die Option → ImageView".</p> <p>Ziehen Sie dann diese Komponente mit gedrückter linker Maustaste in das rechte, obere Frame-Fenster → Component Tree, wie nebenstehend angezeigt. Lassen Sie dann die Maustaste los.</p>														
 <p>Fenster Component Tree und Properties</p>  <p>View im Designer</p>	<p><i>Eigenschaften der ImageView ändern.</i></p> <p>Klicken Sie dazu im Fenster → Component Tree auf die → ImageView.</p> <p>Prüfen Sie dann die nebenstehend angezeigten Eigenschaften der Komponente im darunterliegenden Fenster → Properties ab.</p> <p>Eigenschaften:</p> <table border="1"> <tr> <td>layout:width:</td> <td>wrap_content</td> </tr> <tr> <td>layout:height:</td> <td>wrap_content</td> </tr> <tr> <td>layout:margin:</td> <td>all, 15dp</td> </tr> <tr> <td>layout:gravity:</td> <td>[center], both</td> </tr> <tr> <td>contentDescription:</td> <td>@string/logo_final_description</td> </tr> <tr> <td>id:</td> <td>ivLogo</td> </tr> <tr> <td>src:</td> <td>@drawable/logo_final</td> </tr> </table>	layout:width:	wrap_content	layout:height:	wrap_content	layout:margin:	all, 15dp	layout:gravity:	[center], both	contentDescription:	@string/logo_final_description	id:	ivLogo	src:	@drawable/logo_final
layout:width:	wrap_content														
layout:height:	wrap_content														
layout:margin:	all, 15dp														
layout:gravity:	[center], both														
contentDescription:	@string/logo_final_description														
id:	ivLogo														
src:	@drawable/logo_final														



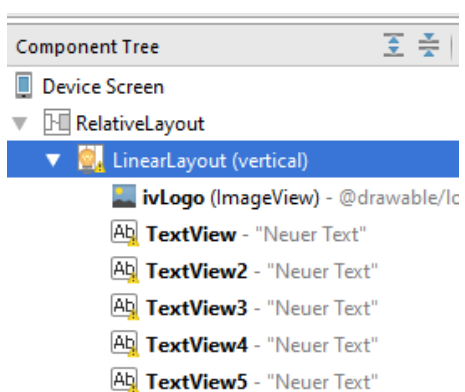
Fenster Palette: Plain TextView auswählen



Fenster Component Tree: Plain TextView einfügen



Gewünschtes Ergebnis

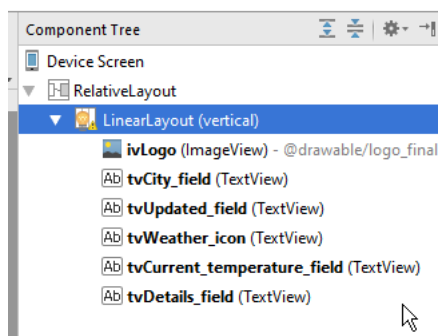


Fenster Component Tree

### Komponenten platzieren.

Alle Komponenten werden wir untereinander in das LinearLayout 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, den → Component Tree.



Fenster Component Tree

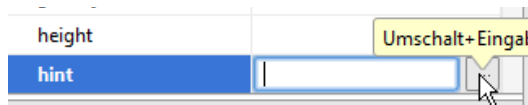
### Eigenschaften für die Text-Komponenten festlegen.

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, → layout-gravity und → id, wie nebenstehend angezeigt, ändern zu können.

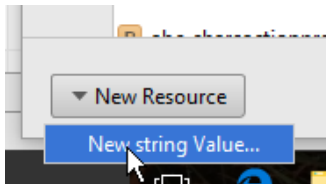
Entfernen Sie den Wert der Eigenschaft (Pro-

Hinweis: Um zusätzlich einen String zu definieren:

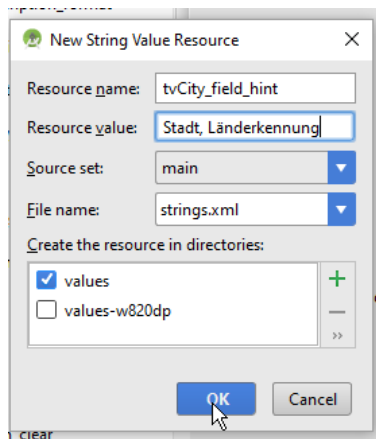
→ Schaltfläche ... anklicken



→ Option „New string Value“ wählen



→ Eigenschaften name und value festlegen



**Density-independent pixel (dp):**

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

**Density-independent Pixels:**

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

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

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

party) → text für alle TextViews.

**TextView: zum Anzeigen des Stadt**

```
layout:width:      fill_parent
layout:height:     wrap_content
layout:marginLeft: 25dp
hint:              @string/tvCity_field_hint
id:               tvCity_field
textAppearance:    ?android:attr/
                  textAppearanceLarge
text-size:        15sp
```

**TextView: zum Anzeigen letzten Aktualisierung**

```
layout:width:      fill_parent
layout:height:     wrap_content
layout:marginLeft: 20dp
hint:              @string/tvUpdated_field_hint
id:               tvUpdated_field
textAppearance:    ?android:attr/
                  textAppearanceMedium
textSize:         15sp
```

**TextView: zum Anzeigen Weather-Icon**

```
layout:width:      fill_parent
layout:height:     wrap_content
layout:marginLeft: 20dp
hint:              @string/tvWeather_icon_hint
id:               tvWeather_icon
textAppearance:    ?android:attr/
                  textAppearanceLarge
textSize:         80sp
```

**TextView: zum Anzeigen der aktuellen Temperatur**

```
layout:width:      fill_parent
layout:height:     wrap_content
layout:marginLeft: 20dp
hint:              @string/tvCurrent_field_hint
id:               tvCurrent_temperature_field
textAppearance:    ?android:attr/
                  textAppearanceLarge
textSize:         45sp
```

**TextView: zum Anzeigen der Details**

```
layout:width:      fill_parent
layout:height:     wrap_content
layout:marginLeft: 20dp
hint:              @string/tvDetails_field_hint
id:               tvDetails_field
textAppearance:    ?android:attr/
                  textAppearanceMedium
textSize:         18sp
```

Ansicht für die neuen Hinweise siehe auch → hints

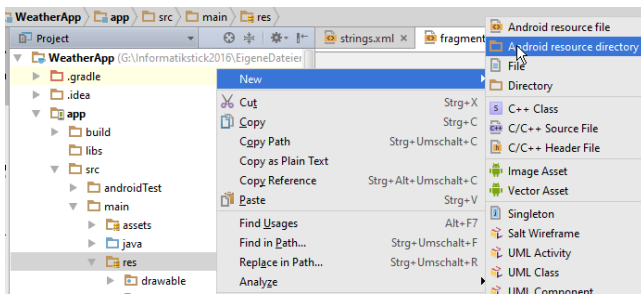


View im Designer

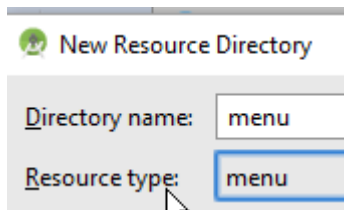
in der strings.xml.

Hinzugefügte Inhalte in der Datei: strings.xml

```
<string name="tvDetails_field_hint">
    Wetter-Details</string>
<string name="tvCity_field_hint">
    Stadt, Länderkennung</string>
<string name="tvUpdated_field_hint">
    Letzte Aktualisierung am ...</string>
<string name="tvWeather_icon_hint">Grafik</string>
<string name="tvCurrent_field_hint">
    Temperatur in °C</string>
```



Kontext-Menü



menu

*Menü-Ressourcen-Verzeichnis erstellen.*

Wir werden zu einem zu einem späteren Zeitpunkt ein Menü für die Lösch- und Änderungsoperationen erweitern. Deshalb erzeugen wir nun im Vorfeld ein Menü-Verzeichnis und integrieren eine erste Menü-Datei.

Klicken Sie dazu mit der rechten Maustaste im Linken Frame auf das Verzeichnis

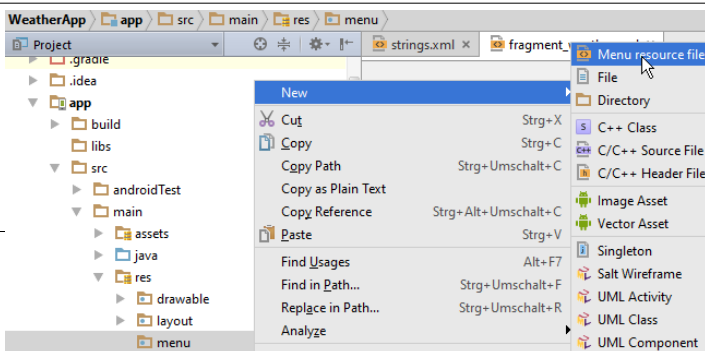
→ res.

Wählen Sie dazu im Kontext-Menü (rechte Maustaste) die Optionen

→ New → Android resource directory

Übernehmen Sie die Angaben, wie nebenstehend angezeigt und klicken Sie auf die Schaltfläche

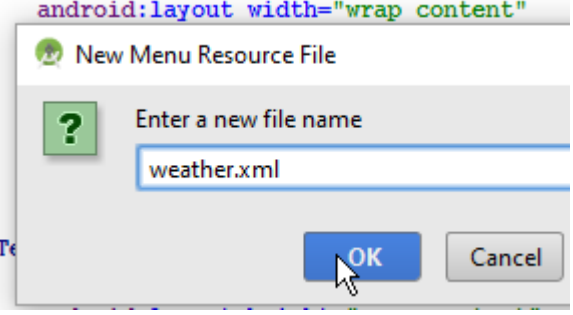
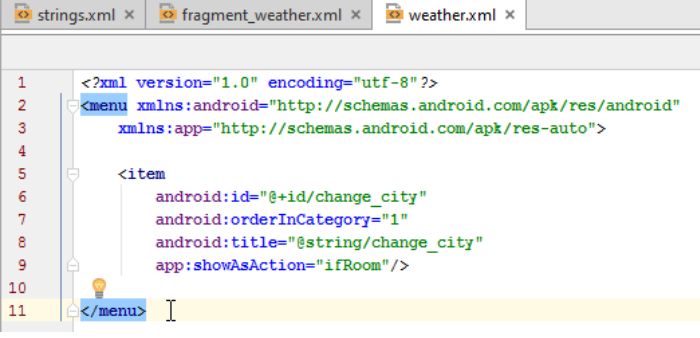
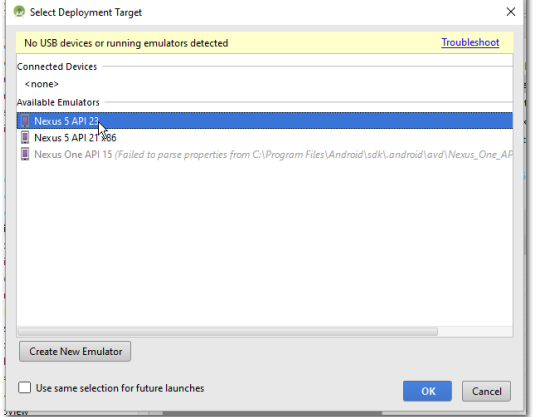
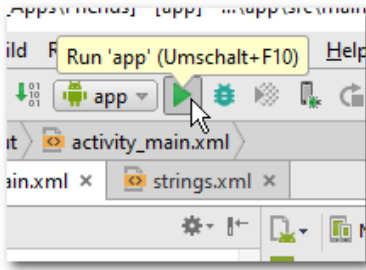
→ OK.



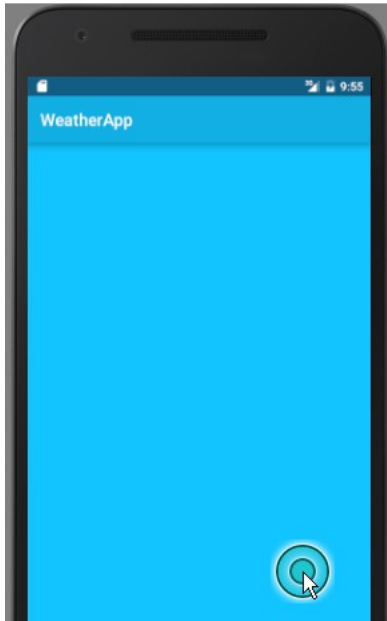
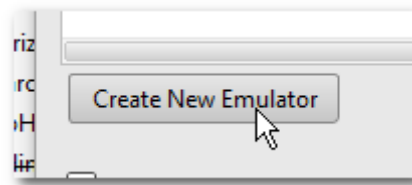
*Menü-Ressourcen-Datei erstellen.*

Klicken Sie dazu mit der rechten Maustaste im Linken Frame auf das Verzeichnis

→ res → menu

<p>Kontext-Menü</p>  <p>weather.xml</p>	<p>Wählen Sie dazu im Kontext-Menü (rechte Maustaste) die Optionen → New → Android resource file</p> <p>Übernehmen Sie die Angaben, wie nebenstehend angezeigt und klicken Sie auf die Schaltfläche → OK.</p>
 <p>Übernehmen Sie dazu die Angaben, wie angezeigt.</p>	<p><i>Inhalt der Menu-Datei ändern.</i></p> <p>Wir definieren in der Menü-Datei das Element → item für die Änderung der Stadt.</p> <p>Eingabehilfe:</p> <pre>&lt;?xml version="1.0" encoding="utf-8"?&gt; &lt;menu xmlns:andro- id="http://schemas.android.com/apk/res/android" xmlns:app="http://schemas.android.com/apk/res-auto"&gt;   &lt;item     android:id="@+id/btChange_city"     android:orderInCategory="1"     android:title="@string/change_city"     app:showAsAction="ifRoom"/&gt; &lt;/menu&gt;</pre>
 <p>Alternativ → Create New Emulator: Für wenig leistungsfähige Rechner empfiehlt sich ein neues Gerät → Nexus One Device mit API 15 (SanwichIceCream) zu erzeugen:</p>	<p><i>Testen der View.</i></p> <p>Wir starten nun den Emulator.</p>  <p>Emulator: Der Emulator simuliert im vorliegenden Fall ein virtuelles Mobiltelefon vom Typ → Nexus 5 API 23.</p>





*Der Emulator öffnet sich.*

Beim ersten öffnen kann das einen Moment dauern.

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

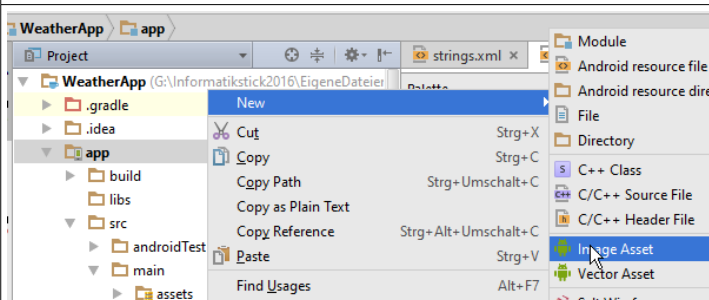
Wenn Sie nicht ungeduldig werden, startet der Emulator die App nach Abschluss des Built-Prozesses von selbst.

Im Ergebnis sollte die Benutzeroberfläche erscheinen.

Die Listenansicht ist nicht zu sehen, da nicht keine Anzeigedaten vorhanden sind.

Hinweis:

Noch ist das gerade noch erzeugte FrameLayout nicht sichtbar. Wir werden es aber zu einem späteren Zeitpunkt indirekt über die Controllerobjekte (WeatherActivity, WeatherFragment) mit dem Datenmodell in Beziehung setzen und steuern.

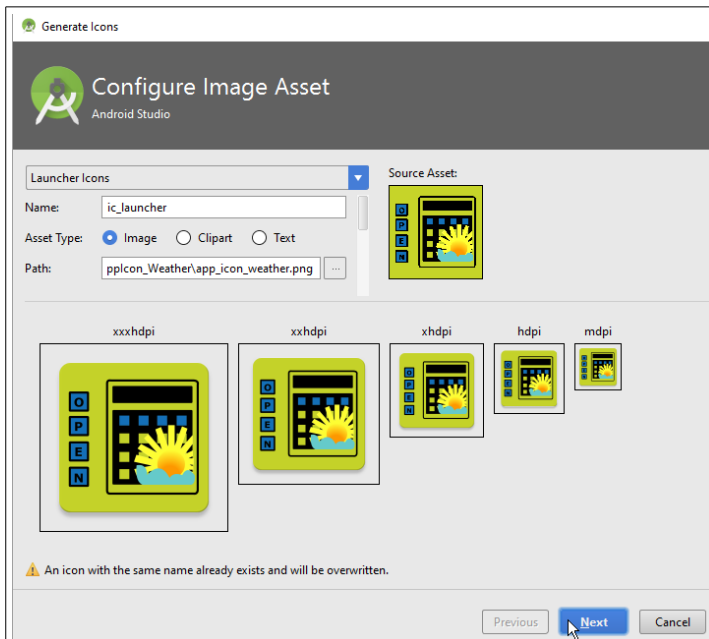


Kontext-Menü

*App Icon ändern.*

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



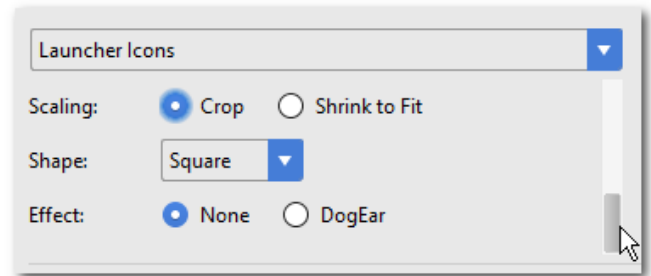


### Image Icon definieren.

Bildquelle:

**Material\AppIcon\_Weather\app\_icon\_weather.png**

Aktivieren Sie für die Eigenschaft → Scaling die Option → Crop und für die Eigenschaft → Shape die Option → Square aus:



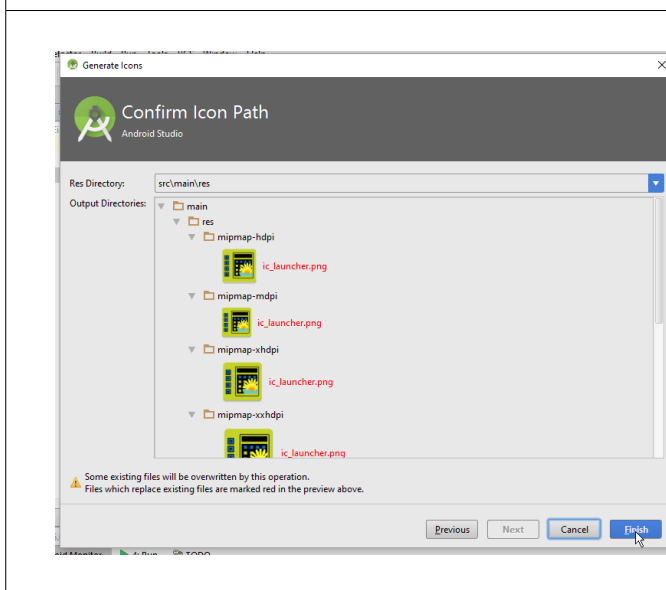
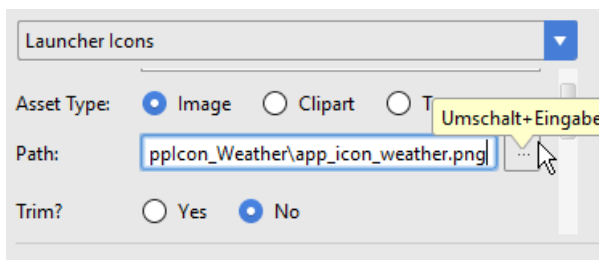
Klicken Sie auf die Schaltfläche → Next.



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

G:\Android\_Schulung\Material\Applcon\_Weather

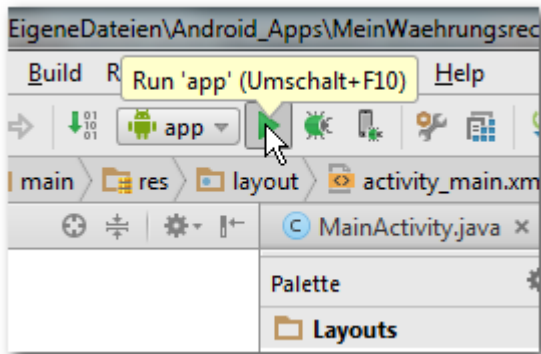
Bildquelle



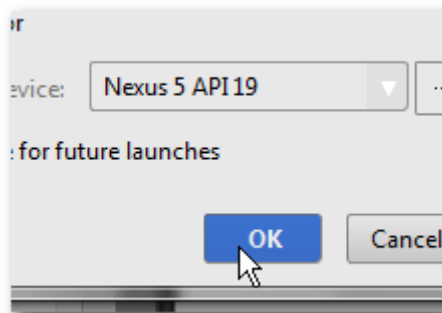
### Icon Konfiguration abschließen.

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





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“.

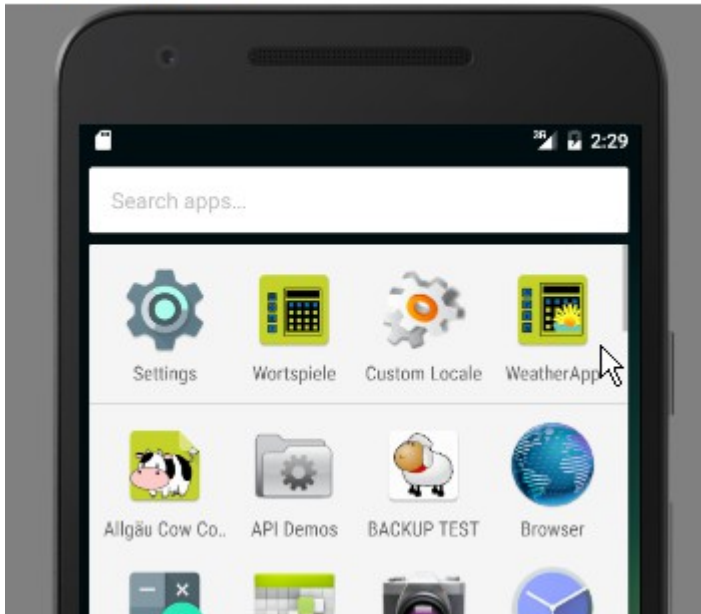
*Bild (Logo) anzeigen.*

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

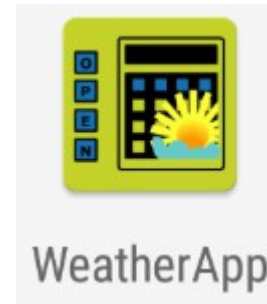
Um das App Icon zu sehen wechseln Sie in das App-Menü. Klicken Sie dazu diese Schaltfläche auf dem Display:



5554:Nexus\_5X\_API\_23


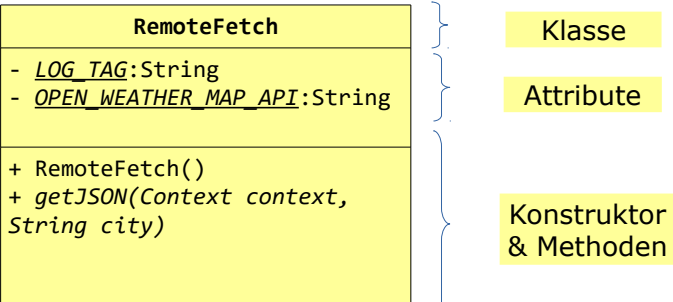
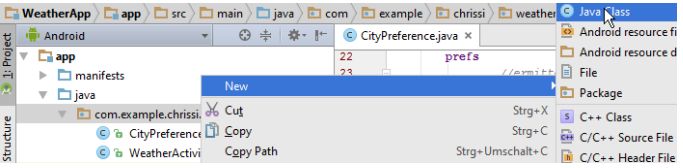
*Icon anzeigen.*

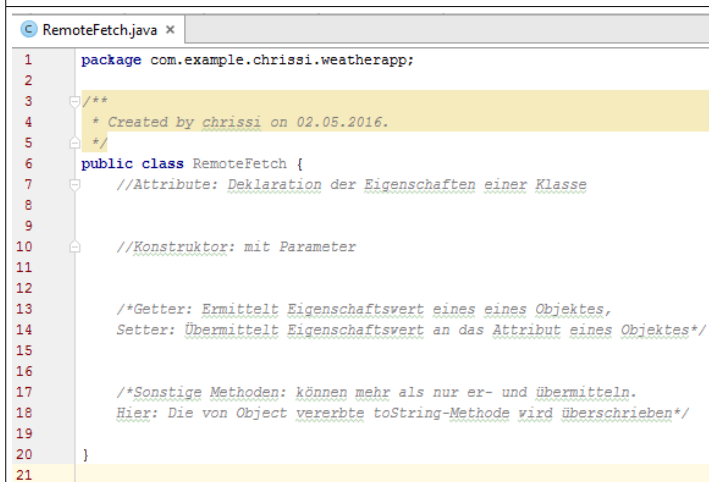
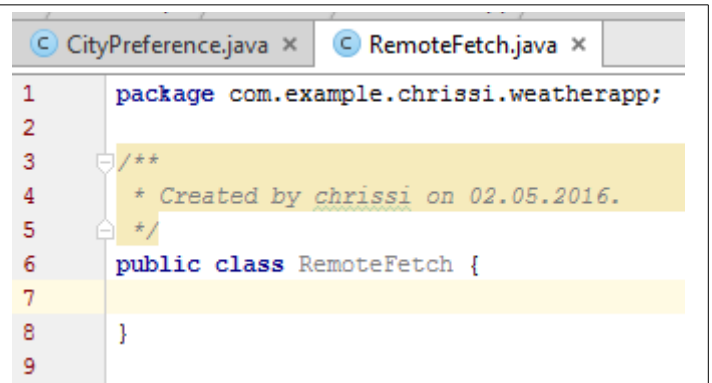
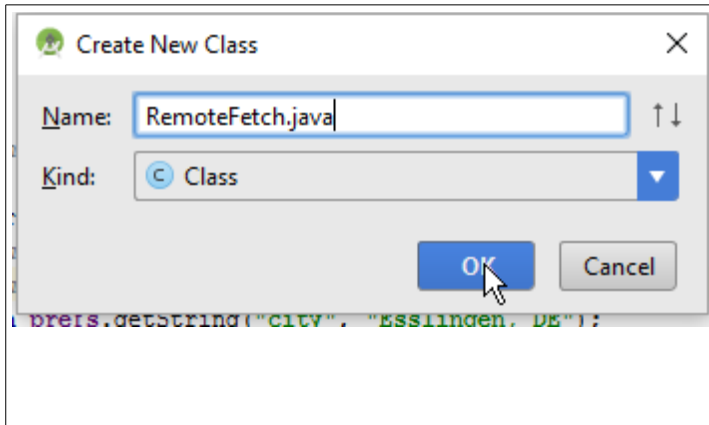
Auf dem Display ist das App Icon nun aufgeführt.



**Gratulation die Benutzeroberfläche ist erstellt!**

## 2.4 Modell: Implementierung der Fachklassen für die Datenhaltung

	<p><i>Vorgehensweise erläutern.</i></p> <p>Es folgen nun die Erläuterungen zur Erstellung unseres Modells. Das Modell enthält die Deklaration und Implementierung aller systemrelevanten Objekteigenschaften und -verhaltensweisen die der zeitweisen Datenaquirierung und -haltung dienen.</p> <p>Im Falle der WeatherApp benötigen wir folgende Schritte:</p> <ol style="list-style-type: none"> <li>1. Implementierung der Schnittstelle für die Nutzung der OpenWeatherMap API. Objekte dieser Klasse sind verantwortlich für die Ermittlung von Wetterdaten aus der OpenWeatherMap API.</li> <li>2. Implementierung einer Klasse die in der Lage ist das bezogene Wetterdatenobjekt (CityPreference-Objekt) temporär zu handeln und zu speichern.</li> </ol>
 <p>UML-Klasse: RemoteFetch</p>	<p><i>Neue Modellklasse → RemoteFetch erstellen.</i></p> <p>Die Objekte dieser Klasse bilden die Schnittstelle und sind verantwortlich für die Ermittlung von Wetterdaten aus der OpenWeatherMap API .</p> <p>Entsprechend den Vorgaben (Anforderungen) der nebenstehend angezeigten UML-Klasse, werden wir in den kommenden Schritten diese Fachklasse implementieren.</p>
	<p><i>Klassenname festlegen.</i></p> <p>Klicken Sie im → app-Verzeichnis mit der rechten Maustaste auf das Package und wählen Sie die Option New → Java Class.</p> <p>Geben Sie als Klassennamen → RemoteFetch ein und klicken Sie auf die Schaltfläche → OK.</p>



Grundgerüst einer Klasse festlegen.

Übernehmen Sie die nebenstehend angezeigten Kommentare.

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

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

Was ist → deklarieren?

In der objektorientierten Programmierung ist mit der Deklaration die

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

gemeint.

Was ist → implementieren?

In der objektorientierten Programmierung ist mit der Implementation die Einbettung bzw. Umsetzung konkreter Programmstrukturen gemeint. Die sogenannte Umsetzung vom „Business Logic“ (automatisierte Prozesse) in Programmcode (Quellcode) 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.

Eingabehilfe:

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

//Konstruktor: ohne Parameter, Leer

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

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

```

1 package com.example.chrissi.weatherapp;
2
3 /**
4  * Created by chrissi on 02.05.2016.
5  */
6 public class RemoteFetch {
7     //Attribute: Deklaration der Eigenschaften einer Klasse
8
9
10    //LOG
11    private static final String LOG_TAG = RemoteFetch.class.getSimpleName();
12
13    //Attribute: Angabe der Datenquelle
14    private static final String OPEN_WEATHER_MAP_API =
15        "http://api.openweathermap.org/data/2.5/weather?q=8&units=metric";

```

**Deklaration und Initialisierung der Attribute.**

Alle Eigenschaften dieser Klasse sind *statisch* und *final*. Jede Eigenschaft erhält einen fixen Werte von uns.

**Modifikatoren:**

**static:**

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

**final:**

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

**Zugriffsmodifikatoren:**

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

**Hinweis:**

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

**Erläuterung Zugriffsmodifikatoren:**

→ **private (-)**  
stellt sicher, dass nur die Objekte der Klasse selbst auf die Eigenschaftswerte direkt zugreifen können.

→ **package (~)**  
stellt sicher, dass die Objekte des Pakets auf die Eigenschaftswerte direkt zugreifen können.

→ **public (+)**  
stellt sicher, dass alle Objekte auf die Eigenschaftswerte direkt zugreifen können.

→ **protected, kein Modifikator (#)**  
stellt sicher, dass nur die Objekte der Klasse und Objekte erbender Klassen auf die Eigenschaftswerte direkt zugreifen können.

**Übersicht Zugriffsmodifikatoren:**

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

j: erreichbar/zugreifbar  
n: nicht erreichbar/zugreifbar

```

8 //LOG
9
10 private static final String LOG_TAG
11     = RemoteFetch.class.getSimpleName();

```

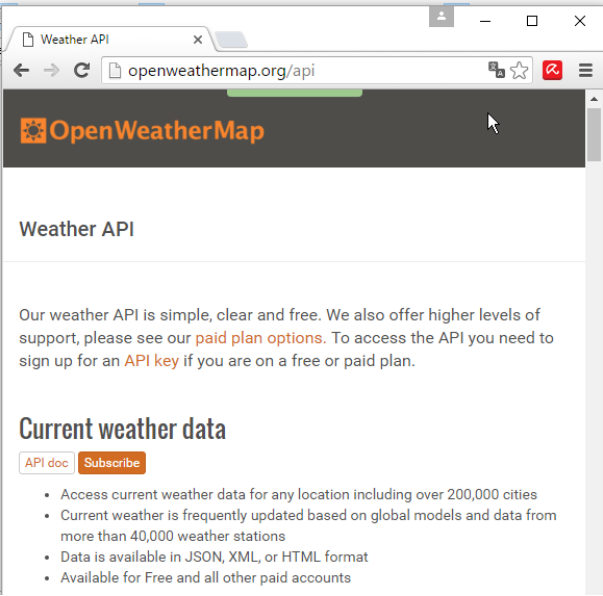
**Die Eigenschaft LOG\_TAG.**

Im Ersten Schritt deklarieren wir ein String-Objekt → LOG-TAG. Das Objekt dient rein dazu, die Abarbeitung der Programmlogik protokollieren zu können. Wir lassen uns später die Meldung im Logcat-Fenster ausgeben. Das hilft

**Eingabehilfe:**

```
private static final String LOG_TAG
```



<pre>= RemoteFetch.class.getSimpleName();</pre>	<p>während der Entwicklung, vor allem dann, wenn wir Fehlersuchen.</p> <p><b>Ergänzen Sie den Quellcode und die Kommentare, wie nebenstehend angezeigt.</b></p>
<pre>12 13 //Angabe der Datenquelle 14 private static final String OPEN_WEATHER_MAP_API = 15     "http://api.openweathermap.org/data/2.5/weather?q=%s&amp;units=metric";</pre> <p><a href="http://openweathermap.org/api">http://openweathermap.org/api</a></p> 	<p><i>Entfernte Datenquelle definieren.</i></p> <p>Übernehmen Sie die Deklaration und Initialisierung der Datenquelle, wie nebenstehend angezeigt.</p> <p>Eingabehilfe:  <pre>private static final String OPEN_WEATHER_MAP_API ="http://api.openweathermap.org/data/2.5/weather?q= %s&amp;units=metric";</pre></p> <p>Hinweise:</p> <ul style="list-style-type: none"> <li>• Der Zusatz: %s ist der Reguläre Ausdruck für den Suchbegriff.</li> <li>• Der Zusatz: &amp;units=metric liefert die Temperaturangabe in Celsius.</li> <li>• Der Zusatz: &amp;lang=de liefert die Angaben in Deutscher Sprache</li> </ul> <p>Erklärung: &amp;lang=de  Die Angabe ermöglicht beispielsweise in Verbindung mit dem Zusatz → Locale.Germany in der Methode → renderWeather (siehe WeatherFragment.java) die Ausgabe der Beschreibung → description in Deutscher Sprache.</p>
<pre>10 17 //Default Konstruktor 18 public RemoteFetch(){ 19 }</pre> <p>Eingabehilfe:  <pre>public RemoteFetch(){ }</pre></p>	<p><i>Deklaration des Standard-Konstruktors.</i></p> <p>Entsprechend dem Grundgerüst einer Klasse implementieren wir den Standardkonstruktor für diese Klasse.</p>
<pre>27 /*Getter: Ermittelt Eigenschaftswert eines eines Objektes, 28 Setter: Übermittelt Eigenschaftswert an das Attribut eines Objektes*/ 29 30 // Getter:Ermittelt den Datensatz mit Wetterdaten anhand der 31 // Angabe fuer die Datenquelle und dem Staedtenamen 32 public static JSONObject getJSON(Context context, String city){ 33 34 35 }</pre> <p>vorher</p>	<p><i>Ermitteln des Wetterdatensatzes.</i></p> <p>Wir deklarieren eine statische Get-Methode, die uns im Ergebnis den Wetterdatensatz aus der entfernten Datenquelle ermitteln soll. Die Methode nutzt dazu die gerade definierte Datenquelle, verkettet die Quelle mit dem Städtenamen (city), bittet um Zugriffsberechtigungen.</p>



Importanweisungen einfügen:

```
org.json.JSONObject? Alt+Eingabe
// Angabe fuer die Datenquelle
public static JSONObject getJSON
```

JSONObject

```
android.content.Context? Alt+Eingabe
// Angabe fuer die Datenquelle und dem Staedtenamen
JSONObject getJSON(Context context, St
```

Context

```
34 // Getter:Ermittelt den Datensatz mit Wetterdaten anhand der
35 // Angabe fuer die Datenquelle und dem Staedtenamen
36 public static JSONObject getJSON(Context context, String city){
37
38
39
```

nachher

```
3 import android.content.Context;
4 import org.json.JSONObject;
Kontrolle der import-Anweisungen
```

gung auf die entfernte Datenquelle und versucht die Anfrage zu stellen. Ist die Anfrage erfolgreich erhalten wir den Datensatz in Form eines JSONObject zurück, anderenfalls erhalten wir die Fehlermeldung im Logcat-Fenster angezeigt.

Eingabehilfe:

```
public static JSONObject getJSON(Context context,
String city){
//Hier fehlt Quellcode
}
```

Wir müssen fehlende Importanweisungen einfügen.

Klicken Sie dazu auf die rot angezeigten Klassennamen und wählen Sie die Tastenkombination ALT + ENTER, um die fehlende Importanweisung einzufügen.

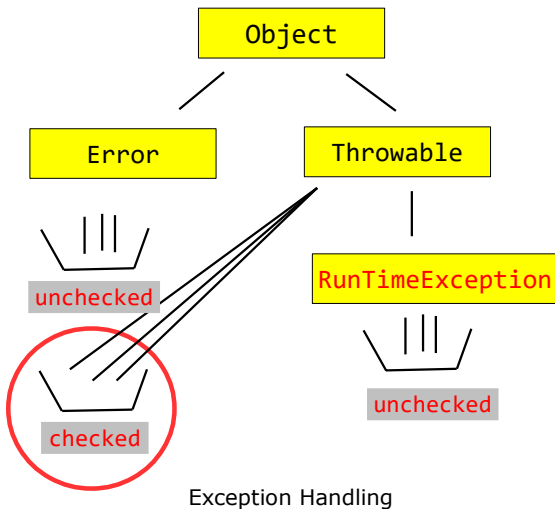
```
33 // Getter:Ermittelt den Datensatz mit Wetterdaten anhand der
34 // Angabe fuer die Datenquelle und dem Staedtenamen
35 public static JSONObject getJSON(Context context, String city){
36 //Versuche...
37 try {
38
39
40 }catch(Exception e){
41 //Meldung auf der Konsole
42 LOG_TAG.concat(" Fehler beim Zugriff auf die Datenquelle!");
43 return null;
44 }
45
46
```

Anfrage-Versuch implementieren.

Wir nutzen innerhalb der gerade erzeugten Methode die Kontrollstruktur → TRY CATCH.

```
try{
Anweisung;
}catch(Exception e){
Ausnahmenbehandlung;
}finally{
Aufräumarbeiten;
}
```

Hierarchie der Ausnahmebehandlung in Java:



Implementieren Sie dazu den folgenden Quellcode für die Kontrollstruktur → TRY CATCH.

Eingabehilfe:

```
try {
//hier fehlt Quellcode
}catch(Exception e){
//Meldung auf der Konsole
LOG_TAG.concat(
" Fehler beim Zugriff auf die Datenquelle!");
return null;
}
```



## Vom Speziellen zum Allgemeinen:

- java.lang.RuntimeException
- java.lang.ArithmeticException
- java.lang.ArrayStoreException
- java.lang.ClassCastException
- java.lang.IllegalArgumentException
- java.lang.IllegalThreadStateException
- java.lang.NumberFormatException
- java.lang.IllegalMonitorException
- java.lang.IllegalStateException
- java.lang.IndexOutOfBoundsException
- java.lang.ArrayOutOfBoundsException
- java.lang.StringOutOfBoundsException
- java.lang.NegativeArraySizeException
- java.lang.NullPointerException
- java.lang.SecurityException
- java.lang.UnsupportedOperationException

## Hinweis: Ausnahmebehandlung

Wenn eine Behandlung erfolgen soll muss bei mehreren sequentiellen **Catch-Blöcken von der speziellen Ausnahme zur allgemeinen Ausnahme** abgefangen werden. Im anderen Falle würde unerreichbarer Code entstehen. Das gilt für alle Exceptions! Um über die Reihenfolge entscheiden zu können, müsste man die Vererbungshierarchie der Exceptions kennen. Nach einer try-catch-Anweisung kann optional eine finally-Anweisung stehen. Bei Try ohne Catch-Anweisung ist finally allerdings vorgeschrieben.

Die Anweisungen im Finally-Block werden grundsätzlich ausgeführt, unabhängig ob im try-Block eine Exception aufgetreten ist oder nicht. In dem finally-Block können demnach „Aufräumarbeiten“ programmiert werden, die in jedem Fall ausgeführt werden müssen.

In unserem Fall wählen wir die einfache Variante und fangen ganz allgemein alle Laufzeitfehler → **Exceptions** ab.

```

36 //Versuche...
37 try {
38     //Ein url-Objekt: verkettet URL der
39     //Datenquelle mit dem Städtenamen
40     URL url = new URL(
41         String.format(
42             OPEN_WEATHER_MAP_API, city));
43
44     //Initialisierung eines Datenverbindungsobjekts
45     HttpURLConnection connection =
46         (HttpURLConnection)url.openConnection();
47
48     //Übermittelt für die Anfrage den API Code
49     connection.setRequestProperty("x-api-key",
50         "98149f523e1fb926ca7cd6b9ce77cf6");
51
52
53     //Erzeugt und initialisiert ein lesbares
54     // Objekt (Empfangsobjekt)
55     BufferedReader reader = new BufferedReader(
56         //Nutzt die Datenverbindung
57         new InputStreamReader(
58             connection.getInputStream()));
59
60     //Erzeugt ein String Buffer Objekt einer bestimmten
61     //Kapazität an Zeichen
62     StringBuffer json = new StringBuffer(1024);

```

## Try-Block implementieren.

Ergänzen Sie den angezeigten Quellcode schrittweise, lesen und fügen Sie auch die Kommentare ein.

Erzeugt ein URL-Objekt. Dazu wird die URL (Unified Resource Locator, Internetadresse) der Datenquelle mit dem Städtenamen verkettet.

```
URL url
    = new URL(String.format(
        OPEN_WEATHER_MAP_API, city));
```

Initialisierung eines Datenverbindungsobjekts und öffnen der Verbindung.

```
HttpURLConnection connection =
    (HttpURLConnection)url.openConnection();
```

Übermittelt für die Anfrage den API Code um die Berechtigung für den Zugriff auf die Daten zu erhalten.

**Bitte prüfen Sie bei dieser Gelegenheit ob in der strings.xml ein gültiger API Code eingefügt wurde!**

```
connection.setRequestProperty("x-api-key",
    context.getString(
        R.string.open_weather_maps_app_id));
```

```

64 //Initialisierung eines String-Objekts
65 // für die temporäre Datenhaltung
66 String tmp="";
67 while((tmp=reader.readLine())!=null)
68     json.append(tmp).append("\n");
69 reader.close();
70
71 //JSONObject (Datenobjekt) wird erzeugt und
72 // mit dem Datensatz (Zeichenkette) initialisiert
73 JSONObject data = new JSONObject(json.toString());
74 System.out.println(json.toString());
75
76 // Wert 404 wenn die Anfrage misslingt
77 if(data.getInt("cod") != 200){
78     return null;
79 }
80
81 //gibt das JSONObject (Datenobjekt) zurueck
82 return data;
83
84 }catch(Exception e){
85     //Meldung auf der Konsole
86     LOG_TAG
87     .concat(
88         " Fehler beim Zugriff auf die Datenquelle!");
89     return null;
90 }
91
92 }
    
```

Erzeugt ein lesbares Objekt von Typ → BufferedReader. Dessen Inhalt wird mit dem empfangenen Datensatz bestückt.

```

BufferedReader reader
    = new BufferedReader(
        new InputStreamReader(
            connection.getInputStream()));
    
```

Erzeugt ein StringBuffer-Objekt mit einer bestimmten Kapazität an Zeichen

```

StringBuffer json = new StringBuffer(1024);
    
```

Initialisierung eines String-Objekts für die temporäre Datenhaltung. Solange weitere Datenzeilen vorhanden sind werden die Datenzeilenweise gelesen und an das StringBuffer-Objekt angehängt. Abschließend wird das reader-Objekt geschlossen.

```

String tmp="";
while((tmp=reader.readLine())!=null)
    json.append(tmp).append("\n");
reader.close();
    
```

JSONObject (Datenobjekt) wird erzeugt und mit dem Datensatz (StringBuffer → String) initialisiert

```

JSONObject data
    = new JSONObject(json.toString());
    
```

Der letzte Eigenschaftswert im Datensatz ist → cod:200. Im folgenden wird geprüft ob dieser Wert nicht vorhanden ist. Wenn das der Fall ist, meldet das System den Fehler 404 da die Anfrage misslungen oder unvollständig ist.

```

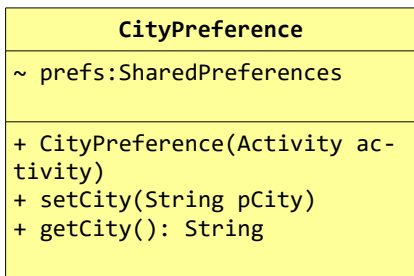
if(data.getInt("cod") != 200){
    return null;
}
    
```

Gibt dann das JSONObject (Datenobjekt/Datensatz) abschließend zurueck.

```

return data;
    
```

Es folgt nach der Anweisung → return data nur noch der bereits implementierte catch-Block.



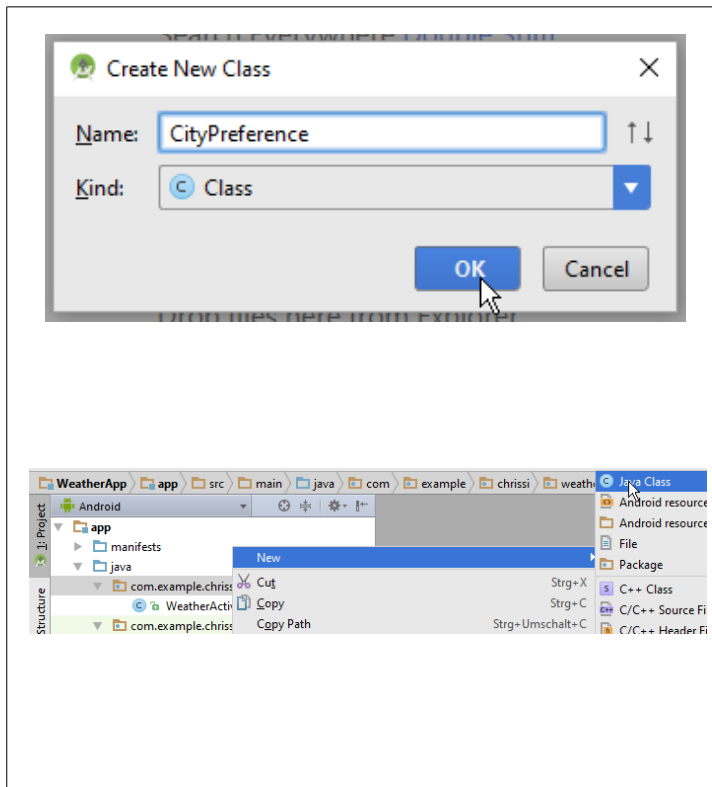
UML-Klasse: CityPreference

- Klasse
- Attribute
- Konstruktor & Methoden

*Neue Modellklasse erstellen.*

Diese Klasse ist in der Lage das bezogene Wetterdatenobjekt (CityPreference-Objekt) temporär zu händeln und zu speichern.

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



### Klassenname festlegen.

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

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

```

CityPreference.java x
1 package com.example.chrissi.weatherapp;
2
3 /**
4  * Created by chrissi on 02.05.2016.
5  */
6 public class CityPreference {
7
8
9

```

```

CityPreference.java x
1 package com.example.chrissi.weatherapp;
2
3 /**
4  * Created by chrissi on 02.05.2016.
5  */
6 public class CityPreference {
7     //Attribute: Deklaration der Eigenschaften einer Klasse
8
9
10    //Konstruktor: mit Parameter
11
12
13    /*Getter: Ermittelt Eigenschaftswert eines eines Objektes Setter:
14     Übermittelt Eigenschaftswert an das Attribut eines Objektes*/
15
16
17    /*Sonstige Methoden: können mehr als nur er- und übermitteln.
18     Hier: Die von Object vererbte toString-Methode wird überschrieben*/
19
20
21

```

### Grundgerüst einer Klasse festlegen.

Übernehmen Sie die nebenstehend angezeigten Kommentare.

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

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

### 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 toString-Methode wird überschrieben*/

```

```

CityPreference.java x
5
6 /**
7  * Created by chrissi on 02.05.2016.
8  */
9 public class CityPreference {
10     //Attribute: Deklaration der
11     //Eigenschaften einer Klasse
12     SharedPreferences prefs;
    
```

**Die Klasse SharedPreferences:**

Das Objekt → prefs der Klasse → SharedPreferences dient der Datenhaltung. Objekte dieser Klasse gewährleisten den Zugriff auf die Eigenschaftswerte genau einer Instanz (z.B. aktuellen Wetterdaten einer Stadt).

**Zugriffsmodifikatoren:**

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

j: erreichbar/zugreifbar  
 n: nicht erreichbar/zugreifbar

public interface **SharedPreferences** Added in API level 1

android.content.SharedPreferences

^ Class Overview

Interface for accessing and modifying preference data returned by `getSharedPreferences(String, int)`. For any particular set of preferences, there is a single instance of this class that all clients share. Modifications to the preferences must go through an `SharedPreferences.Editor` object to ensure the preference values remain in a consistent state and control when they are committed to storage. Objects that are returned from the various `get` methods must be treated as immutable by the application.

*Note: currently this class does not support use across multiple processes. This will be added later.*

**Deklaration der Attribute.**

Wir erzeugen ein Objekt der Klasse SharedPreferences. Dieses Interface erlaubt den Zugriff und die Änderung der Eigenschaftswerte eines Datensatzes. In unserem speziellen Fall nutzen wir das Interface, um die aktuellen Wetterdaten (Datensatz) für genau eine Stadt zu ermitteln und zu übermitteln, für die Änderung besitzen wir in unserem speziellen Fall keine Berechtigung. Das Attribut vom Typ SharedPreferences stellt u.a. die Zugriffsmethoden bereit. Alle systemrelevanten Objekte greifen auf die selbe Instanz der Klasse zurück.

**Eingabehilfe:**

```
SharedPreferences prefs;
```

→ prefs ist der Attributname. Attribute werden in Java kleingeschrieben und enthalten keine Umlaute und/oder Sonderzeichen.

Was ist die Bedeutung der Implementierung ohne Zugriffsmodifikator?

**Zugriffsmodifikatoren:**

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

→ protected, kein Modifikator (#) stellt sicher, dass nur die Objekte der Klasse und Objekte erbender Klassen auf die Eigenschaftswerte direkt zugreifen können.



```

19 //Konstruktor: mit Parameter
20 public CityPreference(Activity activity){
21     //Objekt wir initialisiert
22     prefs
23     //ermittelt die Eigenschaftswerte
24     //mittels des Activity-Objektes
25     = activity.getPreferences(
26         //Activity erbt indirekt von Context
27         //regelt die Zugriffsrechte auf die Datei
28         //Aufruf über die Activity
29         //Mode: statischer Wert (int)
30         Activity.MODE_PRIVATE);
31 }

```

Eingabehilfe:

```

public CityPreference(Activity activity){
    prefs = activity.getPreferences(
        Activity.MODE_PRIVATE);
}

```

*Deklaration eines Konstruktors mit Parameter.*

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

Erklärung zum Inhalt des Konstruktors:  
Objekt wir initialisiert

**prefs**

Ermittelt die Eigenschaftswerte mittels des Activity-Objektes

**= activity.getPreferences (**

Activity erbt indirekt von Context. Die Zugriffsrechte auf die Datei wird dazu über den Eigenschaftswert → MODE\_PRIVATE, ein statischer Wert (int) gesetzt.

**Activity.MODE\_PRIVATE);**

```

33 // GETTER und SETTER
34 // Für den Fall, dass der Benutzer noch keine Stadt gewählt hat,
35 // werden die Wetterdaten für diese Stadt ermittelt
36 public String getCity(){
37     //return prefs.getString("city", "Muenchen, DE");
38     //return prefs.getString("city", "Wangen, DE");
39     return prefs.getString("city", "Esslingen, DE");
40 }
41
42 //Übermittelt den Wert für city an das aktuelle prefs Objekt
43 public void setCity(String pCity){
44     prefs.edit().putString("city", pCity).commit();
45 }
46

```

Getter und Setter:

```

public String getCity(){
    return prefs
        .getString("city", "Esslingen, DE");
}

public void setCity(String pCity){
    prefs.edit()
        .putString("city", pCity).commit();
}

```

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

Berücksichtigen Sie, dass wir auf die Eigenschaftswerte der CityPreference-Objekte von außerhalb der Klasse (z.B. von der Benutzeroberfläche aus) zugreifen müssen. Das Attribut benötigt deshalb eine Get- und Set-Methode.

**Implementieren Sie diese Methoden, wie nebenstehend angezeigt.**

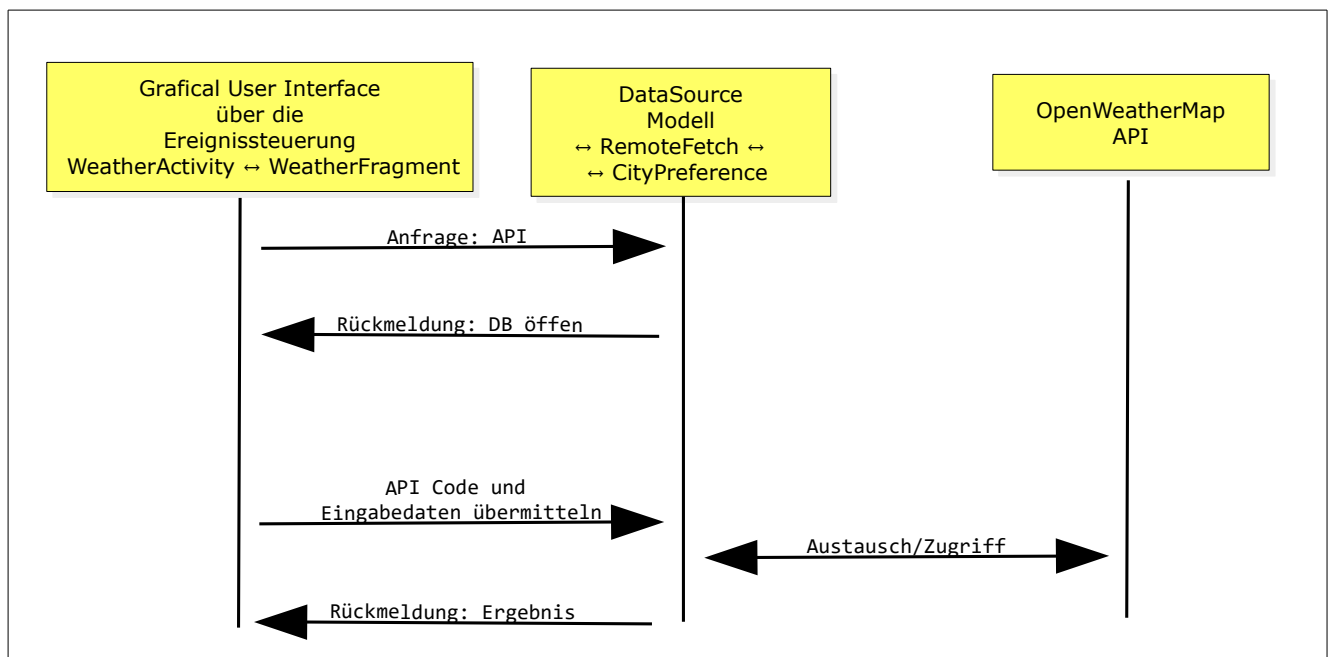
Damit sind die Modellklassen fertig erstellt. Es ist nun sinnvoll Modell und View indirekt über die Controller-Klassen in Beziehung zu setzen. Damit wir die Ereignisse unserer Anwendung über die Benutzeroberfläche steuern und testen können.

Hinweis:

Wir haben es uns in unserem Beispiel nur vorgenommen genau einen aktuellen Wetterdatensatz je Stadt zuzuordnen (Multiplizität → 1:1), dementsprechen einfach gestalten wir diese Klasse.

## 2.5 Controller: Daten anzeigen und aktualisieren

### Zugriff auf OpenWeatherMap API



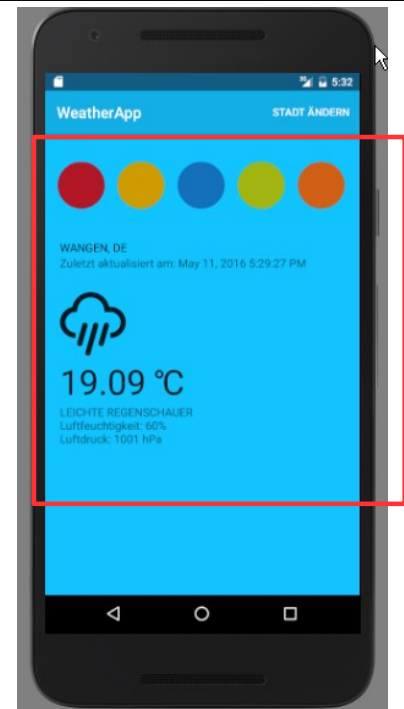
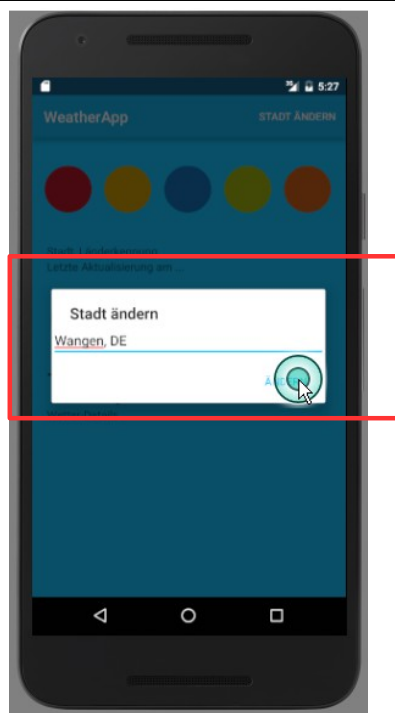
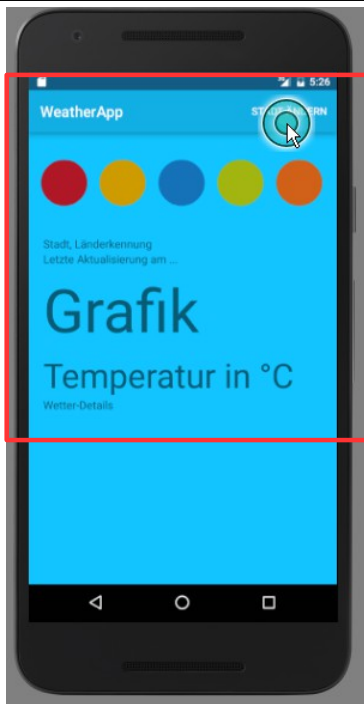
### Vorgehensweise erläutern.

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

1. Steuerung (Fragment) implementieren
2. Steuerung (Activity) implementieren



## Anzeigen und Aktualisieren



## WeatherFragment

```

~ weatherfont: Typeface
~ cityField:TextView
~ updateField:TextView
~ detailsField:TextView
~ currentTemperatureField:TextView
~ weatherIcon:TextView
~ handler: Handler

+ WeatherFragment()
+ onCreateView(LayoutInflater inflater,
    ViewGroup container,
    Bundle savedInstanceState):View
+ onCreate(Bundle savedInstanceState)
+ updateWeatherData(final String city)
+ renderWeather(JSONObject json)
+ setWeatherIcon(int actualId,
    long sunrise, long sunset)
+ changeCity(String city)

```

UML-Klasse: WeatherFragment.java

Controller-Klasse → *WeatherFragment* erstellen.

Fragmente repräsentieren einen Teil einer Aktivität. Wir könnten mehrere Fragmente für eine Aktivität verwenden. Dies wäre beispielsweise dann der Fall, wenn wir eine MultiPane oder TabLayout verwenden würden. Im folgenden Beispiel verwenden wir das Fragment um die Aktualisierung und Darstellung der Wetterdaten zu kapseln.

Die Klasse *WeatherFragment* stellt mit der Methode:

```

→ onCreateView(
    LayoutInflater inflater,
    ViewGroup container,
    Bundle savedInstanceState)

```

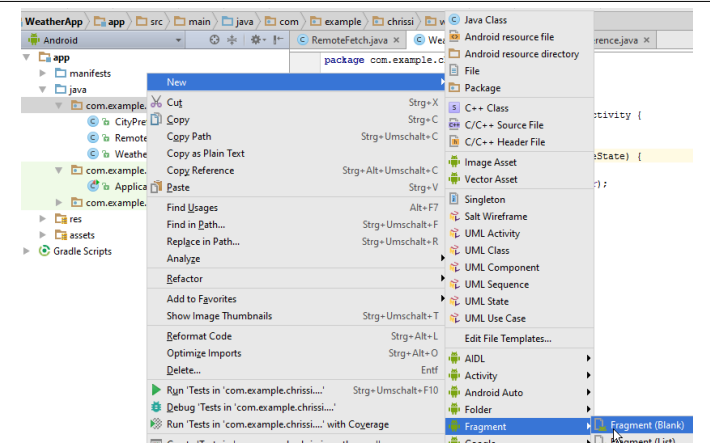
beim Starten der Anwendung sicher, dass die Benutzeroberfläche initialisiert und angezeigt wird.

Im folgenden werden wir zudem alle Voraus-



setzungen für die Anzeige und Aktualisierung der Daten schaffen.

Entsprechend den Vorgaben (Anforderungen) der nebenstehend angezeigten UML-Klasse, werden wir die Implementierung in den kommenden Schritten umsetzen.

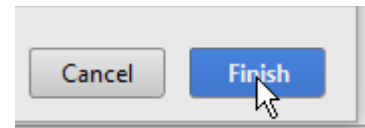


*Neue Fragment-Klasse erstellen.*

Wir implementieren die Controller-Klasse → FragmentActivity indem wir sie erst neu anlegen und dann mit dem benötigten Quellcode ausstatten.

Klicken Sie im → app-Verzeichnis mit der rechten Maustaste auf das Package und wählen Sie die Option New → Fragment → Fragment (Blank).

Geben Sie als Klassennamen → WeatherFragment ein, da wir das zugehörige Layout bereits erstellt haben entfernen Sie das Häkchen für „Create layout XML“, verzichten Sie ebenfalls auf die Includes und klicken Sie abschließend auf die Schaltfläche → Finish.



Führen Sie nun Schrittweise die Implementierung dieser Klassen durch.

```

17 public class WeatherFragment extends Fragment {
18     //Deklaration Schrift-Objekts
19     Typeface weatherFont;
20     Handler handler;
21     //Deklaration des handler-Objektes
22     Handler handler;
23
24     //Deklaration der Komponenten
25     TextView cityField;
26     TextView updatedField;
27     TextView detailsField;
28     TextView currentTemperatureField;
29     TextView weatherIcon;
30

```

Vorher

*Deklaration fehlender Komponenten.*

Wir deklarieren ein Objekt für die Schriftdeklaration, ein Handler-Objekt und fünf TextView Komponenten für die Wetterdaten die wir auf der Benutzeroberfläche darstellen möchten.

Fügen Sie die import-Anweisung für die Typeface- und Handler-Klasse ein, klicken Sie dazu jeweils auf den Klassennamen an und wählen Sie die Tastenkombination ALT + ENTER auf Ihrer Tastatur.



Eingabehilfe:

```

Typeface weatherFont;
Handler handler;
TextView cityField;
TextView updatedField;
TextView detailsField;
TextView currentTemperatureField;
TextView weatherIcon;
17 public class WeatherFragment extends Fragment {
18     //Deklaration Schrift-Objekts
19     Typeface weatherFont;
20
21     //Deklaration des Handler-Objektes
22     Handler handler;
23
24     //Deklaration der Komponenten
25     TextView cityField;
26     TextView updatedField;
27     TextView detailsField;
28     TextView currentTemperatureField;
29     TextView weatherIcon;
30
    
```

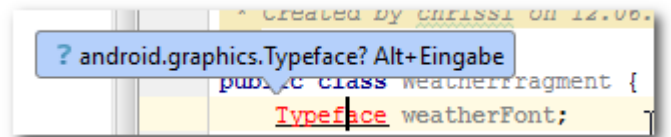
Nachher

Kontrollieren Sie anschließend die Import-Anweisungen oberhalb der Klassendeklaration:

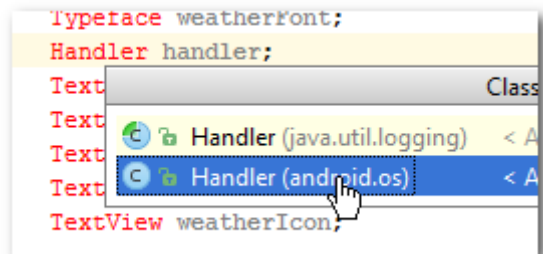
```

import android.graphics.Typeface;
import android.os.Handler;
import android.widget.TextView;
    
```

Für die Import-Anweisung der Klasse Typeface:



Für die Import-Anweisung der Klasse Handler:



## Typeface

public class Typeface  
extends Object

java.lang.Object

↳ android.graphics.Typeface

The Typeface class specifies the typeface and intrinsic style of a font. This is used in the paint, along with optionally Paint settings like textSize, textSkewX, textScaleX to specify how text appears when drawn (and measured).

API Klasse Typeface

Added in API level 1  
Summary: Constants | Fields | Methods | Protected Methods | Inherited Methods | [Expand All]

## Handler

public class Handler  
extends Object

java.lang.Object

↳ android.os.Handler

Known Direct Subclasses

AsyncQueryHandler, AsyncQueryHandler.WorkerHandler, HttpAuthHandler, SslErrorHandler

A Handler allows you to send and process Message and Runnable objects associated with a thread's MessageQueue. Each Handler instance is associated with a single thread and that thread's message queue. When you create a new Handler, it is bound to the thread / message queue of the thread that is creating it – from that point on, it will deliver messages and runnables to that message queue and execute them as they come out of the message queue.

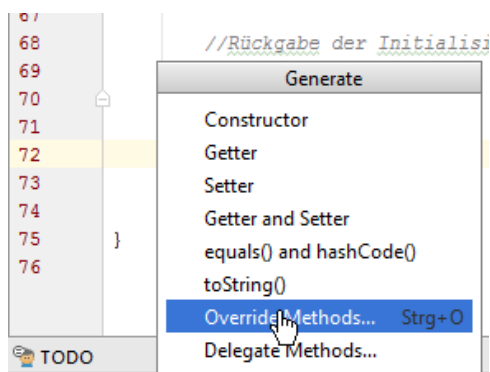
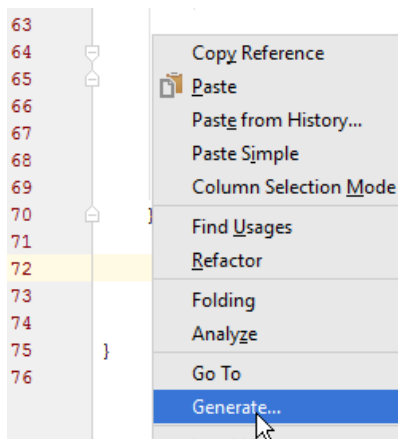
There are two main uses for a Handler: (1) to schedule messages and runnables to be executed as some point in the future; and (2) to enqueue an action to be performed on a different thread than your own.

API Klasse Handler

Added in API level 1  
Summary: Nested Classes | Ctors | Methods | Inherited Methods | [Expand All]

Fügen Sie auf die gleiche Weise die Importanweisung für die TextView-Klasse ein.

<pre> 33 public WeatherFragment() { 34     //Initialisierung des Handler-Objektes 35     handler = new Handler(); 36 } </pre>	<p><i>Erweiterung des Konstruktors.</i></p> <p>Wir erweitern den Standard-Konstruktor, um die Initialisierung des Händler-Objekts.</p> <p>Eingabehilfe:  <pre> public WeatherFragment(){     handler = new Handler(); } </pre> </p>
<p>Vorher</p> <pre> 39 @Override 40 public View onCreateView(LayoutInflater inflater, ViewGroup container, 41     Bundle savedInstanceState) { 42     TextView textView = new TextView(getActivity()); 43     textView.setText(R.string.hello_blank_fragment); 44     return textView; 45 } </pre> <p>Nachher</p> <pre> 39 @Override 40 public View onCreateView( 41     LayoutInflater inflater, 42     ViewGroup container, 43     Bundle savedInstanceState) { 44     //Initialisierung der Komponenten 45     View rootView 46         = inflater.inflate( 47             R.layout.fragment_weather, container, false); 48     cityField 49         = (TextView) rootView.findViewById( 50             R.id.tvCity_field); 51     updatedField 52         = (TextView) rootView.findViewById( 53             R.id.tvUpdated_field); 54     detailsField 55         = (TextView) rootView.findViewById( 56             R.id.tvDetails_field); 57     currentTemperatureField 58         = (TextView) rootView.findViewById( 59             R.id.tvCurrent_temperature_field); 60     weatherIcon 61         = (TextView) rootView.findViewById( 62             R.id.tvWeather_icon); 63 64     //Übermittlung der Schriftdeklaration für 65     //Ausgabe der Wetterdaten in der TextView-Komponente 66     weatherIcon.setTypeface(weatherFont); 67 68     //Rückgabe der Initialisierten Benutzeroberfläche 69     return rootView; 70 } </pre>	<p><i>FrameLayout anzeigen.</i></p> <p>Wir implementieren/überschreiben dazu die Methode → onCreateView.</p> <p>Ergänzen Sie den folgenden Quellcode und die Kommentare, wie nebenstehend angezeigt.</p> <p>Initialisierung der Komponenten mittels des Inflater-Objekts (Befüller):</p> <pre> View rootView     = inflater.inflate(         R.layout.fragment_weather, container, false); cityField     = (TextView) rootView.findViewById(         R.id.tvCity_field); updatedField     = (TextView) rootView.findViewById(         R.id.tvUpdated_field); detailsField     = (TextView) rootView.findViewById(         R.id.tvDetails_field); currentTemperatureField     = (TextView) rootView.findViewById(         R.id.tvCurrent_temperature_field); weatherIcon     = (TextView) rootView.findViewById(         R.id.tvWeather_icon); </pre> <p>Übermittlung der Schriftdeklaration für die Ausgabe der Wetterdaten in der TextView-weatherIcon-Komponente:</p> <pre> weatherIcon.setTypeface(weatherFont); </pre> <p>Rückgabe der Initialisierten Benutzeroberfläche:</p> <pre> return rootView; </pre>



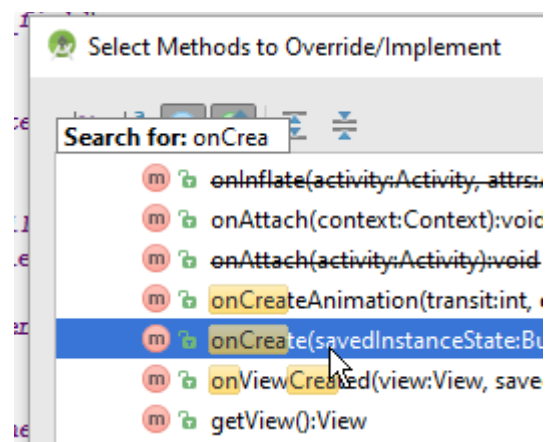
Kontext-Menü

### Erzeugen der Anwendung.

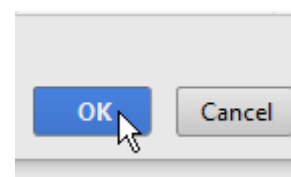
Beim Erzeugen der Anwendung sollen bereits die wichtigsten programmlogischen Schritte ausgelöst werden. Diese Schritte legen wir in der onCreate-Methode.

Klicken Sie in die Klasse unterhalb der onCreateView-Methode wählen sie dann im Kontext-Menü (rechte Maustaste) die Option → Generiere.

Wählen Sie weiter im Kontext-Menü die Option → Override Methods.



Im Fenster „Select Methods to Override/Implement“ geben Sie einfach den Methodennamen → onCreate ein, wählen Sie die Methode aus und klicken Sie auf die Schaltfläche → OK.



```

73  @Override
74  public void onCreate(
75      @Nullable Bundle savedInstanceState) {
76      super.onCreate(savedInstanceState);
77  }

```

Vorher

### Überschreiben der onCreate-Methode.

Ergänzen Sie dazu unterhalb der angezeigten Implementierung die Initialisierung des Typeface-Objekts. Dazu wird die Quellangabe für die Datei mit der Schriftdekoration → weather.ttf indirekt über das aktuelle Activity-Object aus dem Verzeichnis → assets ermittelt

```

73  @Override
74  public void onCreate(Bundle savedInstanceState) {
75      super.onCreate(savedInstanceState);
76
77      //Initialisierung
78      weatherFont = Typeface.createFromAsset(
79          getActivity().getAssets(), "weather.ttf");
80
81      //Aktualisierung der Wetterdaten anhand der
82      // aktuellen Stadt. Entweder zuletzt eingegeben
83      // Stadt oder Default-Wert (Wangen,DE)
84      updateWeatherData(new CityPreference(
85          getActivity()).getCity());
86  }

```

Nachher

```

88  //Aktualisierung der Wetterdaten anhand der Stadt
89  private void updateWeatherData(final String city) {
90
91  }

```

Deklaration: updateWeatherData

Eingabehilfe:

```

private void updateWeatherData(final String city) {
    //hier fehlt Quellcode
}

```

```

91  //Aktualisierung der Wetterdaten anhand der Stadt
92  private void updateWeatherData(final String city) {
93      //Erzeugt dazu ein neues Thread-Objekt
94      new Thread() {
95          //Startet den Thread
96          public void run() {
97              //Erzeugt ein neues JSON-Objekt und
98              //ermittelt anhand der Stadt den
99              // entfernt liegenden Wetterdatensatz
100             final JSONObject json
101                 = RemoteFetch.getJSON(getActivity(), city);
102
103             //Für den fall dass kein passender
104             // Datensatz gefunden wird
105             if (json == null) {
106                 //Übernimmt das Handler-Objekt die
107                 // Ausgabe der Meldung (Toast)
108                 handler.post(new Runnable() {
109                     public void run() {
110                         //Erzeugt und zeigt die Meldung an
111                         Toast.makeText(getActivity(),
112                             getActivity()
113                                 .getString(
114                                     R.string.place_not_found),
115                             //Meldungsfenster
116                             Toast.LENGTH_LONG).show();
117                     }
118                 });
119             }
120             //Ansonsten

```

und an das Typeface-Objekt → weatherFont übergeben:

```

weatherFont = Typeface.createFromAsset(
    getActivity().getAssets(), "weather.ttf");

```

Ergänzen Sie die die Anweisung zur Aktualisierung der Wetterdaten anhand der aktuellen Stadt:

```

updateWeatherData(new CityPreference(
    getActivity()).getCity());

```

Da diese Methode von uns bisher weder deklariert noch initialisiert wurde, wird Sie rot angezeigt. Wir widmen uns also im nächsten Schritt der Methode → updateWeatherData.

*Die Aktualisierung/Änderung der Wetterdaten.*

Wir deklarieren die parameterbehaftete Methode → updateWeatherData ohne Rückgabewert, um die Wetterdaten später aktualisieren und ändern zu können. Fügen Sie dazu die nebenstehende Deklaration ein.

Fügen Sie die Deklaration ein.

*Wetterdaten aktualisieren.*

Implementieren Sie zum besseren Verständnis die Methode, wie folgt schrittweise und fügen Sie die Kommentare ein.

Wir initialisieren und starten ein neues Thread-Objekt. Ein Thread beinhaltet eine Ausführungsreihenfolge, um ein Teilaufgabe innerhalb eines Programms zu erledigen. In unserem Fall umfasst die Teilaufgabe die Aktualisierung von Daten:

```

new Thread() {
    //Hier fehlt Quellcode
}.start();

```

Innerhalb des Threads implementieren wir die run-Methode. Die Methode führt den Quellcode für den neuen Thread aus :

```

public void run() {
    //Hier fehlt Quellcode
}

```

Innerhalb der run-Methode erzeugen wir ein neues JSON-Objekt (zur Speicherung der Wetterdaten) und führen die Initialisierung des Objektes durch.

```

119     } else {
120         //Anderenfalls
121         //Übernimmt das Handler-Objekt die Ausgabe
122         // des Wetterdaten
123         handler.post(new Runnable() {
124             public void run() {
125                 //dazu wird der ermittelte
126                 // Wetterdatensatz übersetzt
127                 renderWeather(json);
128             }
129         });
130     }
131
132     } //Startet den Thread
133
134     }.start();
135 }

```

## Thread

extends `Object`  
implements `Runnable`

`java.lang.Object`  
↳ `java.lang.Thread`

Known Direct Subclasses  
`ForkJoinWorkerThread`, `HandlerThread`

### ^ Class Overview

A `Thread` is a concurrent unit of execution. It has its own call stack for methods being invoked, their arguments and local variables. Each application has at least one thread running when it is started, the main thread, in the main `ThreadGroup`. The runtime keeps its own threads in the system thread group.

There are two ways to execute code in a new thread. You can either subclass `Thread` and overriding its `run()` method, or construct a new `Thread` and pass a `Runnable` to the constructor. In either case, the `start()` method must be called to actually execute the new `Thread`.

API Klasse: Thread

public class `JSONObject` Summary: Fields | Ctors | Methods | Inherited Methods | [Expand All]  
Added in API level 1

extends `Object`

`java.lang.Object`  
↳ `org.json.JSONObject`

### ^ Class Overview

A modifiable set of name/value mappings. Names are unique, non-null strings. Values may be any mix of `JSONObjects`, `JSONArrays`, Strings, Booleans, Integers, Longs, Doubles or `NULL`. Values may not be `null`, `NANs`, `infinities`, or of any type not listed here.

This class can coerce values to another type when requested.

- When the requested type is a boolean, strings will be coerced using a case-insensitive comparison to "true" and "false".
- When the requested type is a double, other `Number` types will be coerced using `doubleValue`. Strings that can be coerced using `valueOf(String)` will be.
- When the requested type is an int, other `Number` types will be coerced using `intValue`.

API Klasse: JSONObject

Dazu wird mittels eines statischen Objekts unserer bereits vorhandenen Klasse → `RemoteFetch` die aktuelle Aktivität → `getActivity()` (der Klasse `WeatherActivity`), der aktuellen Datensatz → `getJSON` ermittelt und die aktuelle Activity, sowie der Eigenschaftswert für die Stadt übermittelt.

JSON-Objekt: Speicherung der Wetterdaten

```
final JSONObject json
    = RemoteFetch.getJSON(getActivity(), city);
```

Wir unterscheiden dann weiter in zwei mögliche Fälle.

Fallunterscheidung: IF-ELSE

```
if (json == null) {
    JA-Fall
    //Ansonsten
} else {
    NEIN-Fall
}
```

JA-Fall: Für den Fall, dass kein passender Datensatz gefunden wurde, übernimmt das Handler-Objekt die Ausgabe der Meldung → Entschuldigung, es konnten keine Daten gefunden werden.

Das Handler-Objekt benötigt dazu ein Objekt der Klasse `Runnable`. Dieses Objekt erzeugt und übermittelt die Meldung in einem einfachen Fenster (Toast). Die Klasse repräsentiert nur ein Kommando, die `run`-Methode.

Ergänzen Sie erst den **JA-Fall** im Quellcode:

```
handler.post(new Runnable() {
    public void run() {
        Toast.makeText(getActivity(),
            getActivity()
                .getString(
                    R.string.place_not_found),
            Toast.LENGTH_LONG).show();
    }
});
```

NEIN-Fall: Anderenfalls konnte ein passender Datensatz ermittelt werden. Wenn das der Fall ist, müssen dann die benötigten Eigenschaftswerte aus dem JSON-Objekt ermittelt werden. Dazu nutzen wir eine neue Methode → `renderWeather`.

Ergänzen Sie den **NEIN-Fall** im Quellcode:

```
handler.post(new Runnable() {
    public void run() {
```

public interface

# Runnable

Summary: Methods | [Expand All]  
Added in API level 1

java.lang.Runnable

Known Indirect Subclasses

AnimationDrawable, CookieSyncManager, ForkJoinWorkerThread, FutureTask<V>, HandlerThread, RenderScript.RSErrorHandler, RenderScript.RSMessageHandler, RunnableFuture<V>, RunnableScheduledFuture<V>, Thread, TimerTask

## ^ Class Overview

Represents a command that can be executed. Often used to run code in a different [Thread](#).

API Klasse: Runnable

```
renderWeather(json);
    }
});
```

Die Methode → updateWeatherData ist nun vollständig implementiert.

Da die Methode → renderWeather von uns bisher weder deklariert noch implementiert wurde, wird Sie noch rot angezeigt. Wir widmen uns also im nächsten Schritt der Methode → renderWeather.

```
137 //Call-By-Name: Ermittelt die Wetterdaten aus dem
138 //Json-Objekt und schreibt die
139 //Facts in die Ausgabe-Komponenten (TextViews)
140 private void renderWeather(JSONObject json) {
141
142 }
```

Deklaration: renderWeather

Element-Tree: OpenWeatherMap

1. coord
  - lon
  - lat
2. weather
  - id
  - main
  - description
  - icon
3. main
  - temp
  - pressure
  - humidity
  - temp\_min
  - temp\_max
  - sea\_level
  - grnd\_level
4. wind
  - speed
  - deg
5. clouds
  - all
6. rain
  - 3h
7. snow
  - 3h
8. dt
9. sys
  - type
  - id
  - message
  - country
  - sunrise
  - sunset
10. id
11. name
12. cod

```
1  {
2  "coord":{
3    "lon":9.61,
4    "lat":48.73
5  },
6  "weather":[{"id":803,
7    "main":"Clouds",
8    "description":"Überwiegend bewölkt",
9    "icon":"04d"}],
10 "base":"cmc stations",
11 "main":{"temp":20.83,
12   "pressure":962.67,
13   "humidity":59,
14   "temp_min":20.83,
15   "temp_max":20.83,
16   "sea_level":1025.34,
17   "grnd_level":962.67
18 },
19 "wind":{"speed":3.26,
20   "deg":100.501
21 },
22 "clouds":{"all":80
23 },
24 "dt":1462810014,
25 "sys":{"message":0.01,
26   "country":"DE",
27   "sunrise":1462765639,
28   "sunset":1462819770},
29 "id":2814279,
30 "name":"Wangen",
31 "cod":200
32 }
33 }
```

Datensatz für Wangen,DE am 09.05.2016

Die Wetterdaten darstellen.

Die Methode → renderWeather ist eine Methode ohne Rückgabewert, aber mit Parameter. Als Parameter wird der ermittelte Wetterdatensatz (JSON-Objekt) übergeben.

Aus dem Element-Tree des JSON-Objekt werden nun die erforderlichen Eigenschaftswerte anhand des Namens (Call-By-Name) ermittelt und auf der Benutzeroberfläche in die dafür vorgesehenen TextView-Komponenten geschrieben/übermittelt.

Eingabehilfe:

```
private void renderWeather(JSONObject json) {
    //hier fehlt Quellcode
}
```

Wir werden vorerst nur sieben ausgewählte Eigenschaftswerte auf der Benutzeroberfläche anzeigen.

**Fügen Sie die Deklaration der Methode ein.**



```

141
142 //Call-By-Name: Ermittelt die Wetterdaten aus dem
143 // Json-Objekt und schreibt die
144 //Facts in die Ausgabe-Komponenten (TextViews)
145 private void renderWeather(JSONObject json) {
146 //Versuch...
147 try {
148 //Ermittelt für Stadt und Land die Eigenschaftswerte
149 // im Element-Tree
150 //und schreibt die Werte in die TextView-Komponente
151 // cityField
152 cityField.setText(json.getString("name")
153 .toUpperCase(Locale.GERMANY) +
154 ", " +
155 json.getJSONObject("sys")
156 .getString("country"));
157
158 //Ermittelt die Zweige für Wetter, main und
159 // sys im Element-Tree
160 JSONObject details = json.getJSONArray("weather")
161 .getJSONObject(0);
162 JSONObject main = json.getJSONObject("main");
163 JSONObject sys = json.getJSONObject("sys");
164
165 //Ermittelt am Wetter-Zweig für die Elemente Beschreibung,
166 // Luftfeuchtigkeit und Luftdruck die Eigenschaftswerte
167 // im Element-Tree
168 //und schreibt die Werte in die TextView-Komponente
169 // detailsField
170 detailsField.setText(
171 details.getString("description")
172 .toUpperCase(Locale.GERMANY) +
173 "\n" + "Luftfeuchtigkeit: "
174 + main.getString("humidity") + "% " +
175 "\n" + "Luftdruck: "
176 + main.getString("pressure") + " hPa");
177
178 //Ermittelt am Main-Zweig für die Elemente Temperatur
179 // im Element-Tree
180 //und schreibt die Werte in die TextView-Komponente
181 //currentTemperatureField
182 currentTemperatureField.setText(
183 String.format("%.2f", main.getDouble("temp")) + " °C");
184
185 //Ermittelt den aktuellen Zeitstempel die Eigenschaftswerte
186 // im Element-Tree
187 //und schreibt die Werte in die TextView-Komponente
188 // updatedField
189 DateFormat df = DateFormat.getDateInstance();
190 String updatedOn = df.format(new Date(json.getLong("dt") * 1000));
191 updatedField.setText("Zuletzt aktualisiert am: " + updatedOn);
192
193 //Ermittelt am Sys-Zweig für die Elemente sunrise
194 // und sunset im Element-Tree und schreibt die
195 // Werte in die TextView-Komponente updatedField
196 setWeatherIcon(details.getInt("id"),
197 sys.getLong("sunrise") * 1000,
198 sys.getLong("sunset") * 1000);
199
200 //Falls der Versuch scheitert
201 } catch (Exception e) {
202 //Fehlermeldung für die Logcat
203 Log.e("WeatherApp",
204 "Ein oder mehrere Werte konnten
205 + "nicht ermittelt werden!");
206
207 }

```

### Wetterdaten ermitteln und anzeigen.

Da die Ermittlung einzelner Eigenschaftswerte im Element-Tree des JSON-Objektes ganz oder teilweise scheitern könnte, fangen wir im Rahmen einer Ausnahmenbehandlung alle auftretenden → Exceptions mit der Kontrollstruktur TRY-CATCH ab.

Fügen Sie den folgenden Quellcode schrittweise ein und fügen Sie die Kommentare hinzu.

Implementieren Sie dazu zunächst die Kontrollstruktur wie folgt:

```

try {
//hier fehlt Quellcode
} catch (Exception e) {

    Log.e("WeatherApp",
        "Ein oder mehrere Werte konnten"
        + "nicht ermittelt werden!");
}

```

Erklärung: catch

Für den Fall, dass eine Ausnahme auftritt soll im Logcat-Fenster die Meldung → „Ein oder mehrere Werte konnten nicht ermittelt werden“, ausgegeben werden.

Implementieren Sie nun den Versuch die Eigenschaftswerte im Element-Tree, zu ermitteln und auszugeben (TRY-Zweig der Kontrollstruktur), wie folgt schrittweise erläutert.

Erklärung: try

Im Ersten Schritt ermitteln wir die Eigenschaftswerte für die Attribute Stadtname (→ name) und Länderkennung (→ country) im Element-Tree des json-Objektes und schreiben die Werte in die TextView-Komponente → cityField:

```

cityField.setText(json.getString("name")
    .toUpperCase(Locale.GERMANY) +
    ", " +
    json.getJSONObject("sys")
    .getString("country"));

```

Hinweis: Locale.Germany

Die Angabe ermöglicht beispielsweise in Verbindung mit dem Zusatz → &lang=de in der → OPEN\_WEATHER\_MAP\_API (siehe RemoteFetch.java) die Ausgabe der Beschreibung → description in Deutscher Sprache.

public final class  
**Log**  
 Summary: Constants | Methods | Inherited Methods | [Expand All]  
 Added in API level 1

extends [Object](#)

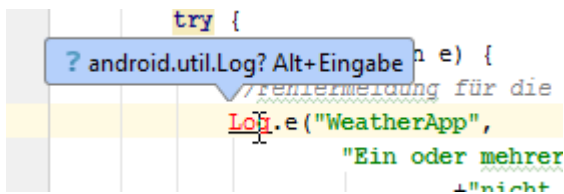
[java.lang.Object](#)  
[↳ android.util.Log](#)

## Class Overview

and use that in subsequent calls to the log methods. that when you're building the string to pass into Log.d, the compiler uses a StringBuilder and at least three allocations occur: the StringBuilder itself, the buffer, and the String object. Realistically, there is also another buffer allocation and copy, and even more pressure on the gc. That means that if your log message is filtered out, you might be doing significant work and incurring significant overhead.

API Klasse: Log

Fügen Sie außerdem die noch fehlende Import-Anweisung für die Klasse → Log ein. Klicken Sie dazu den Klassennamen an und wählen Sie dann die Tastenkombination ALT+ENTER.



[java.lang.Object](#)  
[↳ java.util.Locale](#)

## Class Overview

[Locale](#) represents a language/country/variant combination. Locales are used to alter the presentation of information such as numbers or dates to suit the conventions in the region they describe.

The language codes are two-letter lowercase ISO language codes (such as "en") as defined by [ISO 639-1](#). The country codes are two-letter uppercase ISO country codes (such as "US") as defined by [ISO 3166-1](#). The variant codes are unspecified.

Note that Java uses several deprecated two-letter codes. The Hebrew ("he") language code is rewritten as "iw", Indonesian ("id") as "in", and Yiddish ("yi") as "ji". This rewriting happens even if you construct your own [Locale](#) object, not just for instances returned by the various lookup methods.

### Available locales

This class' constructors do no error checking. You can create a [Locale](#) for languages and countries that don't exist, and you can create instances for combinations that don't exist (such as "de\_US" for "German as spoken in the US").

API Klasse: Locale

Fügen Sie die Import-Anweisung für die Klasse → Locale ein. Klicken Sie den Klassennamen an und wählen Sie dann die Tastenkombination ALT+ENTER.

Danach ermitteln wir die Element-Knoten für die Wetterdetails (→ weather), Main (→ main) und Sys (→ sys) und übermitteln die Daten an drei neue Json-Objekte:

```
JSONObject details
= json.getJSONArray("weather")
    .getJSONObject(0);
JSONObject main
= json.getJSONObject("main");
JSONObject sys
= json.getJSONObject("sys");
```

Dann ermitteln wir die Elemente Beschreibung (→ description), Luftfeuchtigkeit (→ humidity) und Luftdruck (→ pressure) die Eigenschaftswerte im Element-Tree und schreiben die Werte in die TextView-Komponente detailsField:

```
detailsField.setText(
    details.getString("description")
        .toUpperCase(Locale.GERMANY)
    + "\n" + "Luftfeuchtigkeit: "
    + main.getString("humidity") + "%"
    + "\n" + "Luftdruck: "
    + main.getString("pressure") + " hPa");
```

Außerdem ermitteln wir den Eigenschaftswert für das Element Temperatur (→ temp) im Element-Tree und schreiben den Wert formatiert in die TextView-Komponente currentTemperatureField:

```
currentTemperatureField.setText(
    String.format("%.2f",
        main.getDouble("temp")) + " °C");
```

Weiter ermitteln wir den aktuellen Zeitstempel (→ dt) den Eigenschaftswert im Element-Tree und schreiben Wert formatiert in die TextView-Komponente updatedField:

```
DateFormat df
= DateFormat.getDateTimeInstance();
String updatedOn
= df.format(new Date(
    json.getLong("dt") * 1000));
updatedField.setText(
    "Zuletzt aktualisiert am: " + updatedOn);
```

Abschließend ermitteln wir für die Elemente id, sunrise und sunset die Eigenschaftswerte im Element-Tree und schreiben die Werte in die TextView-Komponente und übermitteln die Werte als Parameter an die Hilfsmethode → setWeatherIcon:

```
setWeatherIcon(details.getInt("id"),
    sys.getLong("sunrise") * 1000,
    sys.getLong("sunset") * 1000);
```

Da die Methode → setWeatherIcon von uns bisher weder deklariert noch implementiert



```
//und schreibt die Werte in die T
// cityF ? java.util.Locale? Alt+Eingabe
cityField.setText(jsn.getString(
    .toUpperCase(Locale.GERMA
    " " +
```

wurde, wird Sie noch rot angezeigt. Wir widmen uns also im nächsten Schritt der Methode → setWeatherIcon.

Ergänzen Sie zuvor abschließend noch die fehlende import-Anweisungen für die Klasse → DateFormat und → Date, auf gleiche Weise, wie zuvor für die Klasse → Log und → Locale.

```
209 //Call-By-Name: Ermittelt u.a. das
210 // WetterIcon aus dem Json-Objekt
211 private void setWeatherIcon(
212     int actualId,
213     long sunrise,
214     long sunset) {
215
216 }
```

*Das Wetterbild.*

Wir deklarieren dafür die parameterbehaftete Methode → setWeatherIcon ohne Rückgabewert. Fügen Sie dazu die nebenstehende Deklaration ein.

Die aus dem Element-Tree des JSON-Objekts übermittelten Eigenschaftswerte für die → weather.id, → sys.sunrise und sys.sunset werden nun dazu genutzt das geeignete → icon auszuwählen, um dieses anschließend in der dafür vorgesehenen TextView-Komponente → weatherIcon auszugeben.

Eingabehilfe:

```
private void setWeatherIcon(int actualId, long sunrise, long sunset) {
    //hier fehlt Quellcode
}
```

**Fügen Sie die Deklaration ein.**

```
209 //Call-By-Name: Ermittelt u.a. das
210 // WetterIcon aus dem Json-Objekt
211 private void setWeatherIcon(
212     int actualId,
213     long sunrise,
214     long sunset) {
215
216     //Berechnung der id für
217     //besondere Wetterverhältnisse
218     int id = actualId / 100;
219
220     //Platthalter für das icon
221     String icon = "";
222
223     //Für den Fall, dass keine besonderen
224     // Wetterverhältnisse vorliegen
225     // ist die actualId = 800
226     // prüft dazu erst die actualId
227     //IF-Case
228     if (actualId == 800) {
229
230         //Ermittelt die aktuelle Tageszeit
231         long currentTime = new Date().getTime();
```

*Wetterbild ermitteln und anzeigen.*

**Fügen Sie den folgenden Quellcode schrittweise in die Methode ein und ergänzen Sie die Kommentare zum besseren Verständnis.**

Wir berechnen im Ersten Schritt den Wert für das lokale Attribut → id. Denn im Falle besonderer Wetterverhältnisse (z.B. Unwetter, Regen, Nebel,...) soll der Wert → id dazu dienen ein passendes → icon auszuwählen. Außerdem deklarieren und initialisieren wir ein lokales Attribut → icon als Platzhalter für die Bildreferenz (Hinweis: Unicodes definiert in res → strings.xml):

```
int id = actualId / 100;
String icon = "";
```

Wir unterscheiden dann zwei Fälle mit Hilfe der Kontrollstruktur IF-ELSE und prüfen ob besondere Wetterverhältnisse vorliegen. Für den Fall, dass keine besonderen Wetterverhältnisse vorliegen ist der Wert für die → actualId gleich 800 (JA-Fall).

Implementieren Sie dazu zunächst die Kontrollstruktur wie folgt:

```

233 //prüft dann weiter ob die aktuelle Zeit zwischen
234 // Sonnenaufgang und
235 // Sonnenuntergang befindet (ob Tag oder Nacht)
236 //JA-Fall: tagsüber
237 if (currentTime >= sunrise
238     && currentTime < sunset) {
239     //für Tag
240     icon = getActivity()
241         .getString(
242             R.string.weather_sunny);
243 } else {
244     //NEIN-Fall: Nachts
245     //für Nachts
246     icon = getActivity()
247         .getString(
248             R.string.weather_clear_night);
249 }

```

```

250 //Else-Case
251 //Besondere Wetterverhältnisse prüfen
252 } else {
253 //Ansonsten prüfe die id
254 switch (id) {
255 //Für Unwetter
256 case 2:
257     icon = getActivity()
258         .getString(
259             R.string.weather_thunder);
260     break;
261 //Für Regnerisches
262 case 3:
263     icon = getActivity()
264         .getString(
265             R.string.weather_drizzle);
266     break;
267 //Für Regen
268 case 5:
269     icon = getActivity()
270         .getString(
271             R.string.weather_rainy);
272     break;
273 //Für Schnee
274 case 6:
275     icon = getActivity()
276         .getString(
277             R.string.weather_snowy);
278     break;

```

```

279 //Für Nebel
280 case 7:
281     icon = getActivity()
282         .getString(
283             R.string.weather_foggy);
284     break;
285 //Für Wolkelig
286 case 8:
287     icon = getActivity()
288         .getString(
289             R.string.weather_cloudy);
290     break;
291 }
292 }
293 //schreibt den Unicode für die Grafik in die
294 weatherIcon.setText(icon);
295 }
296 }

```

```

if (actualId == 800) {
    JA-Fall
} else {
    SONST-FALL
}

```

Für den **JA-Fall**:

Im JA-Fall liegen keine besonderen Wetterverhältnisse vor.

Wir ermitteln innerhalb des JA-Falls die Aktuelle Tageszeit:

```
long currentTime = new Date().getTime();
```

Dann unterscheiden wir zwei weitere Fälle. Wir prüfen darin weiter ob die aktuelle Tageszeit zwischen Sonnenaufgang (→ sunrise) und Sonnenuntergang (→ sunset) liegt, also ob Tag oder Nacht ist. Je nach dem, initialisieren wir das → icon mit dem bereits definierten Unicode (→ strings.xml) für tagsüber mit dem → weather\_sunny icon bzw. für Nachts mit dem → weather\_clear\_night icon.

Implementieren Sie dazu im Anschluss an das Attribut → currentTime die gerade erläuterte Kontrollstruktur IF-ELSE wie folgt:

```

if (currentTime >= sunrise && currentTime < sunset) {
    //Tag: sonnig
    icon = getActivity()
        .getString(
            R.string.weather_sunny);
} else {
    //Nachts: klar
    icon = getActivity()
        .getString(
            R.string.weather_clear_night);
}

```

Für den **SONST-Fall**:

Für den Fall, dass besondere Wetterverhältnisse vorliegen differenzieren wir sechs weitere Fälle und nutzen dazu den Wert der eingangs der Methode berechneten → id.

Hinweis: weather conditions

Die Berechnung der → id für die Identifizierung besonderer Wetterverhältnisse lässt sich aus der OpenWeatherMap API ableiten. Siehe dazu:

<http://openweathermap.org/weather-conditions>

actualId	id	Bezeichnung
200 - 232	2	→ Unwetter

Eingabehilfe:

```
//Ansonsten prüfe die id
switch (id) {
    //Für Unwetter
    case 2:
        icon = getActivity()
            .getString(
                R.string.weather_thunder);
        break;
    //Für Regnerisches
    case 3:
        icon = getActivity()
            .getString(
                R.string.weather_drizzle);
        break;
    //Für Regen
    case 5:
        icon = getActivity()
            .getString(
                R.string.weather_rainy);
        break;
    //Für Schnee
    case 6:
        icon = getActivity()
            .getString(
                R.string.weather_snowy);
        break;
    //Für Nebel
    case 7:
        icon = getActivity()
            .getString(
                R.string.weather_foggy);
        break;
    //Für Wolkig
    case 8:
        icon = getActivity()
            .getString(
                R.string.weather_cloudy);
        break;
}
```

300 - 321	3	→ Regnerisch
500 - 531	5	→ Regen
600 - 622	6	→ Schnee
700 - 781	7	→ Nebel
801 - 804	8	→ Bewölkt

Implementieren Sie dazu im ELSE-Fall der ersten (äußeren) Kontrollstruktur IF-ELSE die Kontrollstruktur SWITCH-CASE wie nebenstehend angezeigt.

Abschließend müssen wir nun das ermittelte icon nun nur noch an die weatherIcon-TextView-Komponente übermitteln. Bevor Sie die Methode schließen fügen Sie dazu die folgende Anweisung ein:

```
weatherIcon.setText(icon);
```

```
298 | public void changeCity(String city) {
299 |     //Aktualisiert den Datensatz
300 |     updateWeatherData(city);
301 | }
```

Stadt ändern.

Dazu wird die changeCity-Methode implementiert.

Eingabehilfe:

```
public void changeCity(String city) {
    updateWeatherData(city);
}
```

Für die Änderung von Daten ermöglichen wir einen indirekten, öffentlichen Zugriff auf die bereits implementierte aber private Hilfsmethode → updateWeatherData.

Fügen Sie den Quellcode für die Methode, wie nebenstehend angezeigt ein.

Bravo! Wir haben nun nahezu alle programmlo-

	<p>gischen Voraussetzungen geschaffen, um die Anzeige der Wetterdaten in unserer WeatherApp zu ermöglichen.</p> <p>Nur eine Sache fehlt dafür noch!</p> <p>Wir müssen in der Activity-Klasse die Voraussetzungen für die Anzeige und Aktualisierung der Daten schaffen.</p>
<div data-bbox="102 577 766 851" style="border: 1px solid black; padding: 5px;"> <p style="text-align: center;"><b>WeatherActivity</b></p> <hr/> <pre> + onCreate(Bundle savedInstanceState) + onCreateOptionsMenu(Menu menu): boolean + onOptionsItemSelected(MenuItem item) - showInputDialog() + changeCity(String city) </pre> </div> <p style="text-align: center;">UML-Klasse: WeatherActivity.java</p>	<p><i>Controller-Klasse: WeatherActivity anpassen.</i></p> <p><b>Öffnen Sie dazu die Datei: → WeatherActivity.java.</b></p> <p>Ist eine Klasse die u.a. der Startpunkt der Anwendung ist. Die onCreate-Methode agiert hier in ähnlicher Weise, wie die Main-Methode in einer konventionellen Java-Anwendung.</p> <p>Für die Anzeige und Aktualisierung der Wetterdaten werden wir diese Methode erweitern.</p>
<div data-bbox="86 1030 788 1429" style="border: 1px solid gray; padding: 5px;"> <p>WeatherActivity.java ×</p> <pre> 1 package com.example.chrissi.weatherapp; 2 3 import android.support.v7.app.AppCompatActivity; 4 import android.os.Bundle; 5 6 public class WeatherActivity extends AppCompatActivity { 7 8     @Override 9     protected void onCreate(Bundle savedInstanceState) { 10         super.onCreate(savedInstanceState); 11         setContentView(R.layout.activity_weather); 12     } 13 } 14 </pre> </div> <p style="text-align: center;">Vorher</p>	<p><i>Erweitern der onCreate-Methode.</i></p> <p>Erweitern Sie den automatisch erzeugten Quellcode, um die folgenden Anweisungen. Für den Fall, dass die Anwendung erstmalig gestartet wurde, wird das FrameLayout eingebettet und ein neues Objekt der Klasse → WeatherFragment erzeugt. Der FragmentManager übernimmt die Verwaltung dieser Angelegenheit.</p> <p>Eingabehilfe:</p> <pre> if (savedInstanceState == null) {     getSupportFragmentManager()         .beginTransaction()         .add(R.id.container,             new WeatherFragment())         .commit(); } </pre> <p>Hinweis: Achten Sie beim Einfügen darauf, dass Sie die richtigen Importanweisungen einfügen.</p> <p>Für die WeatherActivity.java</p>

```

4  import android.os.Bundle;
5
6  public class WeatherActivity extends AppCompatActivity {
7
8      @Override
9  protected void onCreate(Bundle savedInstanceState) {
10     super.onCreate(savedInstanceState);
11     setContentView(R.layout.activity_weather);
12
13     //Für den Fall dass die Anwendung erstmalig verwendet wird
14     if (savedInstanceState == null) {
15         //FragmentManager fügt das FragmentLayout
16         // in den Layout-Container der ActivityLayouts ein.
17         getSupportFragmentManager().beginTransaction()
18             .add(R.id.container, new WeatherFragment())
19             .commit();
20     }
21 }
22 }
    
```

Nachher

```

4  import android.app.AlertDialog;
5  import android.os.Bundle;
6  import android.support.v7.app.AppCompatActivity;
7  import android.view.Menu;
8  import android.view.MenuItem;
9
10 public class WeatherActivity extends AppCompatActivity {
    
```

Für die WeatherFragment.java

```

3
4  import android.graphics.Typeface;
5  import android.os.Bundle;
6  import android.os.Handler;
7  import android.support.v4.app.Fragment;
8  import android.util.Log;
9  import android.view.LayoutInflater;
10 import android.view.View;
11 import android.view.ViewGroup;
12 import android.widget.TextView;
13 import android.widget.Toast;
14
15 import org.json.JSONObject;
16
17 import java.text.DateFormat;
18 import java.util.Date;
19 import java.util.Locale;
20
21 /**
22  * Created by chrissi on 12.06.2016.
23  */
24 public class WeatherFragment extends Fragment {
25     Typeface weatherFont;
    
```

```

31 @Override
32 public boolean onCreateOptionsMenu(Menu menu) {
33     // Das Inflater-Objekt sorgt dafür dass Menüeinträge
34     // die ermittelt wurden im Menü integriert/eingebettet werden
35     getMenuInflater().inflate(R.menu.weather, menu);
36     return true;
37 }
    
```

Eingabehilfe:

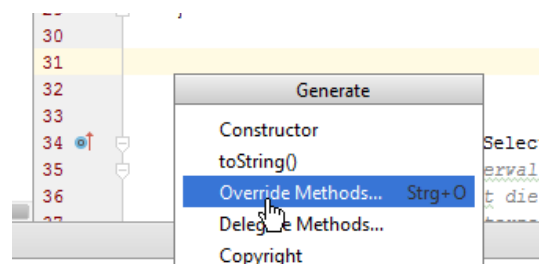
```

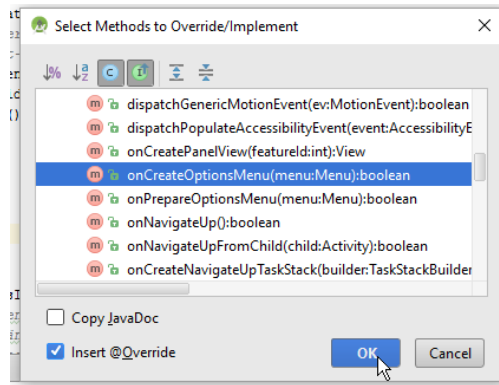
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.weather, menu);
    return true;
}
    
```

Das Menü.

Mit dieser Methode → onCreateOptionsMenu werden die Menüeinträge ermittelt und im Menü integriert/eingebettet (→ rendering). Das Inflater-Objekt (Befüller) kümmert sich genau darum.

Klicken Sie unterhalb der onCreate-Methode in den Quellcode und wählen Sie auf der Tastatur die Tastenkombination ALT + Einfg.





Wählen Sie die Option → Override Methods geben Sie dann im Fenster → Select Methods to Override/Implement den Namen der Methode ein. Damit springt die Markierung auf die gesuchte Methode und Sie können diese mit einem Klick auf die Schaltfläche → OK einfügen.

Ersetzen Sie den Quellcode wie nebenstehend angezeigt.

```

39  @Override
40  public boolean onOptionsItemSelected(MenuItem item) {
41      // Das Action-Bar-Menü verwaltet an dieser Stelle die
42      // Klicks auf Menüeinträge
43      // Die Action-Bar handelt die Klicks auf den Home
44      // und Up Button, so lange in der
45      // Manifest-Datei das Elternelement (Activity)
46      // spezifiziert ist.
47      //Wir nutzen das Menü anderweitig und bestücken das Menü
48      // mit der Schaltfläche --> Stadt ändern
49
50      //Für den Fall dass die Schaltfläche gewählt wurde
51      if(item.getItemId() == R.id.btChange_city){
52          //Zeige den Inputdialog für die Eingabe der Stadt
53          showInputDialog();
54      }
55      return false;
56  }

```

Auswahl eines Menüeintrags.

Mit der Methode → onOptionsItemSelected wird das Action-Bar-Menü verwaltet. Dabei wird i.d.R. geprüft ob ein Eintrag ausgewählt wurde.

Für unseren speziellen Fall möchten wir für das Menü genau einen Eintrag vornehmen.

Wir nutzen das Menü und bestücken das Menü mit der Schaltfläche → Stadt ändern.

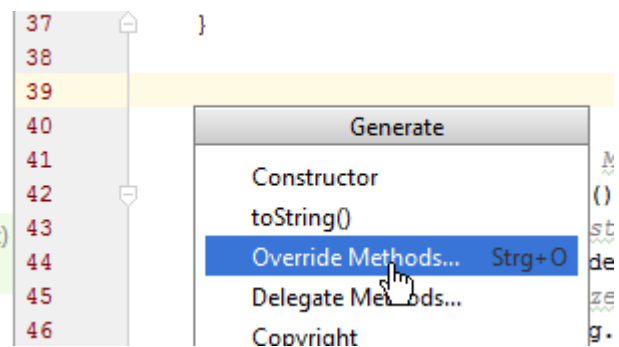
Eingabehilfe:

```

public boolean onOptionsItemSelected(MenuItem item) {
    if(item.getItemId() == R.id.btChange_city){
        showInputDialog();
    }
    return false;
}

```

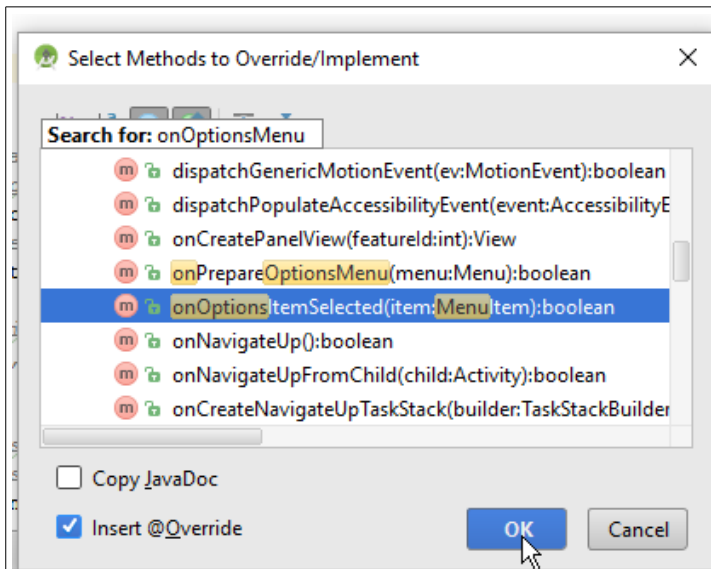
Klicken Sie unterhalb der onCreateOptionsMenu-Methode in den Quellcode und wählen Sie auf der Tastatur die Tastenkombination ALT + Einfg.



Kontext-Menü

Wählen Sie die Option → Override Methods geben Sie dann im Fenster → Select Methods to Override/Implement den Namen der Methode ein. Damit springt die Markierung auf die gesuchte Methode und Sie können diese mit einem Klick auf die Schaltfläche → OK einfügen.





gen.

Ersetzen Sie den Quellcode wie nebenstehend angezeigt.

Hinweis:

Da die Methode → showInputDialog von uns bisher weder deklariert noch implementiert wurde, wird Sie noch rot angezeigt. Wir widmen uns also im nächsten Schritt der Methode → showInputDialog.

```

58 //Ein Inputdialog bietet die Möglichkeit
59 //eine Aktualisierung/Änderung durchzuführen
60 private void showInputDialog(){
61     //Erzeugt eine Dialogfenster-Objekt
62     // und übermittelt das Activity-Objekt
63     AlertDialog.Builder builder
64         = new AlertDialog.Builder(this);
65     //Setzt als Titel die Bezeichnung für die
66     // Schaltfläche --> Stadt ändern
67     builder.setTitle(R.string.btChange_city);
68
69     //Deklaration und Initialisierung einer
70     // Texteingabe-Komponente für
71     //die Eingabe: Stadt, Länderkennung
72     final EditText input = new EditText(this);
73
74     //Setzt den finale, statische Eigenschaftswert
75     //TYPE_CLASS_TEXT als Datentyp
76     input.setInputType(InputType.TYPE_CLASS_TEXT);
77

```

Änderung/Aktualisierung der Wetterdaten.

Die Methode → showInputDialog ermöglicht die Eingabe der Daten und die Durchführung der Änderung/Aktualisierung.

Deklaration der Methode:

```
private void showInputDialog(){
    //hier fehlt Quellcode
}
```

Implementierung der Methode. Im Ersten Schritt wird dazu ein Dialogfenster-Objekt erzeugt und das aktuelle Activity-Objekt übermittelt:

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

Das Dialogfenster erhält als Titel die Bezeichnung die wir auch bereits für die Schaltfläche verwendet haben:

```
builder.setTitle(R.string.btChange_city);
```

Deklaration und Initialisierung einer Texteingabe-Komponente für die Eingabe der Stadt mit Länderkennung:

```
final EditText input = new EditText(this);
```

```

78 //Platziert die Eingabe-Komponente im Dialogfenster
79 builder.setView(input);
80
81 //Ereignissteuerung: Ein Listener-Objekt reagiert
82 // auf die Interaktionen des Benutzers
83 //und löst die Änderung aus wenn die Schaltfläche
84 // --> OK im Dialogfenster angeklickt wird
85 builder.setPositiveButton(R.string.dialog_btChange,
86     new DialogInterface.OnClickListener() {
87         @Override
88         public void onClick(DialogInterface dialog, int which) {
89             changeCity(input.getText().toString());
90         }
91     });
92 //Anzeige des Dialogfensters
93 builder.show();
94 }

```

Eingabehilfe:

```

builder.setPositiveButton(
    R.string.dialog_btChange,
    new DialogInterface.OnClickListener() {

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

            changeCity(
                input.getText().toString());
        }
    });

```

Legt den finalen, statische Eigenschaftswert `TYPE_CLASS_TEXT` als Datentyp fest:

```
input.setInputType(InputType.TYPE_CLASS_TEXT);
```

Zeigt das nun das initialisierte Texteingabefeld an:

```
builder.setView(input);
```

Ereignissteuerung: Ein Listener-Objekt reagiert auf die Interaktionen des Benutzers und löst die Änderung aus, wenn die Schaltfläche → OK im Dialogfenster anklickt.

Zeigt das nun initialisierte Dialogfenster an:

```
builder.show();
```

Hinweis:

Da die Methode → `changeCity` von uns bisher weder deklariert noch implementiert wurde, wird Sie noch rot angezeigt. Wir widmen uns also im nächsten Schritt der Methode → `changeCity`.

```

84 //Löst die Änderungen aus.
85 public void changeCity(String city){
86     //Ermittelt das Wetter-Fragment
87     WeatherFragment wf = (WeatherFragment)getSupportFragmentManager()
88         .findFragmentById(R.id.container);
89
90     //Übermittelt die Eingabe an das Fragment
91     //und löst damit die Aktualisierung der Daten aus
92     wf.changeCity(city);
93
94     //Erzeugt ein neues CityPreference-Objekt und setzt den
95     //Attributwert für --> city
96     new CityPreference(this).setCity(city);
97 }

```

**Im Ersten** deklarieren wir die Methode

```
public void changeCity(String city){
    //hier fehlt Quellcode
}
```

Innerhalb der Methode wird dann das Objekt der Klasse `WeatherFragment` ermittelt:

```
WeatherFragment wf
=
(WeatherFragment)getSupportFragmentManager()
    .findFragmentById(R.id.container);
```

*Stadt ändern.*

Löst die Änderungen aus. Sie nutzt die bereits implementierte `changeCity`-Methode der Klasse → `WeatherFragment`, um die Aktualisierung und Änderung der Wetterdaten durchzuführen.

**Im zweiten Schritt** wird mittels des Fragments dann die Eingabe übermittelt und löst damit die Aktualisierung der Daten aus:

```
wf.changeCity(city);
```

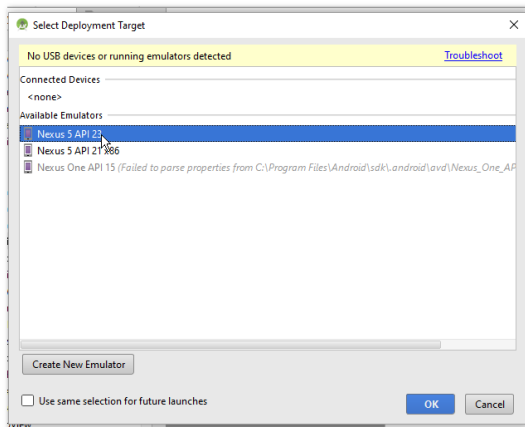
Außerdem werden die Eingabedaten an ein neues `CityPreference`-Objekt übermittelt:

```
new CityPreference(this).setCity(city);
```

Gratulation! Die Anwendung ist nun einsatzbereit. Danke für Ihr Durchhaltevermögen :)

Testen Sie nun die Anwendung.

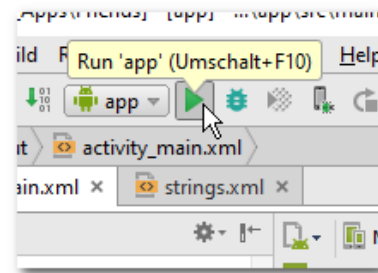




Alternativ → Create New Emulator:  
Für wenig leistungsfähige Rechner empfiehlt sich ein neues Gerät → Nexus One Device mit API 15 (SanwichIceCream) zu erzeugen:

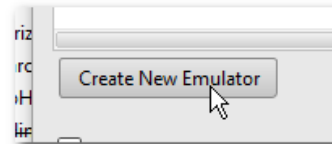
### Testen der Anwendung.

Wir starten nun den Emulator.



### Emulator:

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



### 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 Schösschen mit gedrückter linken Maustaste senkrecht nach oben.

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

Im Ergebnis sollte die Benutzeroberfläche erscheinen.

Testen Sie alle Funktionen der App!

**Gratulation!**