

Mobile Programming

# Android 2 - Benutzerschnittstellen



[kaspar.vongunten@hslu.ch](mailto:kaspar.vongunten@hslu.ch)



# Reminder: Registrierung Übungsabgabe

Pflichtaufgaben (Android-1, Web): <https://tinyurl.com/mobpro-2019-pflicht>

Freie Wahl (Android-2 bis Android-6): <https://tinyurl.com/mobpro-2019-frei>

## Mobpro 2019 - Pflichtübungen

von Kaspar von Gunten • vor 20 Tagen • Drucken

☰ Android-1 und Webapp sind Pflichtübungen.

Präsentation in der Reihenfolge der Eintragungen.

Bitte nur Eintragungen als 2-er Team ("Teamname" oder "Nachname1/Nachname2").

	Android-1	Webapp
28 Teilnehmer	✓28	✓28

Kaspar von Gunter

Fehlen hier noch einige?  
(64 angemeldete TN)

## Mobpro 2019 - Freie Übungen

von Kaspar von Gunten • vor 20 Tagen • Drucken

☰ Es müssen pro Team 2 Übungen gewählt werden.

Maximal 15 Abgaben pro Übung. Präsentation in der Reihenfolge der Eintragungen am entsprechenden Termin.

Nur 1 Eintrag pro Team ("Teamname" oder "Nachname 1/Nachname 2").

	Android-2: UI / Layout	Android-3: Persistenz	Android-4: Backend / Nebenläufigkeit	Android-5: Services / Broadcasts	Android-6: Intent-Filter / Widgets
22 Teilnehmer	✓15/15	✓15/15	✓5/15	✓5/15	✓4/15

Kaspar von Gunter

Hier fehlen definitiv noch viele Einträge!

**Bitte tragen Sie sich bis Ende Woche noch ein, falls noch nicht geschehen!**

# Inhalt

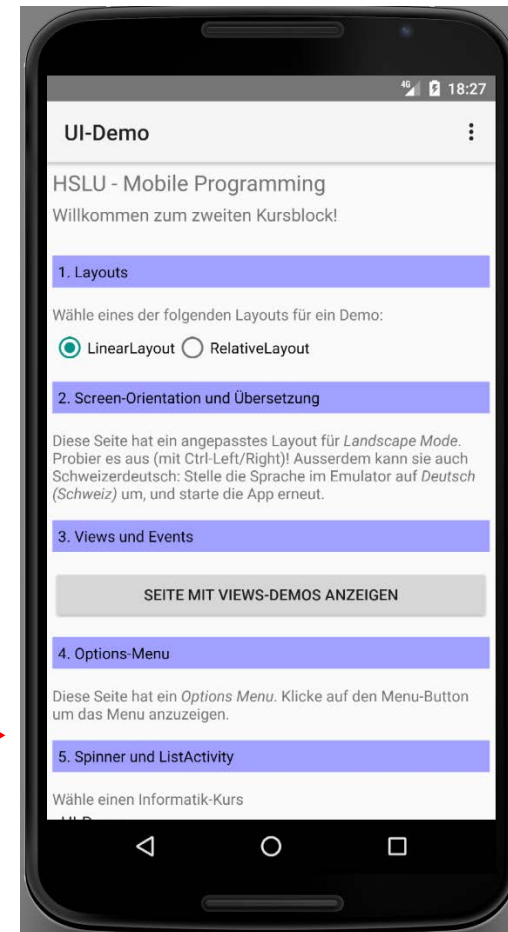
- Komponenten und Layouting
  - Grundkonzepte: Views, ViewGroups & Layouts
  - Linear Layout, Constraint Layout & ScrollView
- Ressourcen, Konfigurationen & Internationalisierung
- UI Event Handling: Registering Listeners
- Options-Menu
- Adapter-View
  - ArrayAdapter
  - Spinner, ListView & ListActivity
- Konfig.-Wechsel & temporäre Datenspeicherung
- Rückmeldung an den Benutzer
  - Toasts, Dialoge & Notifications

**Warnung:  
Viel Stoff heute!**

# Wie kommt eine Activity zu ihrem GUI?

- GUI wird i.d.R. als XML spezifiziert
- Name der XML-Datei resultiert in einer «R.layout.xxx» Konstante
- Diese wird im onCreate(...) der Activity mit setContentView(...) angegeben, vom System compiliert und instanziiert

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    initializeSpinner();
    counterLabel = (TextView) findViewById(R.id.main_label_counter);
}
```



# Beispiel-Layout (XML)

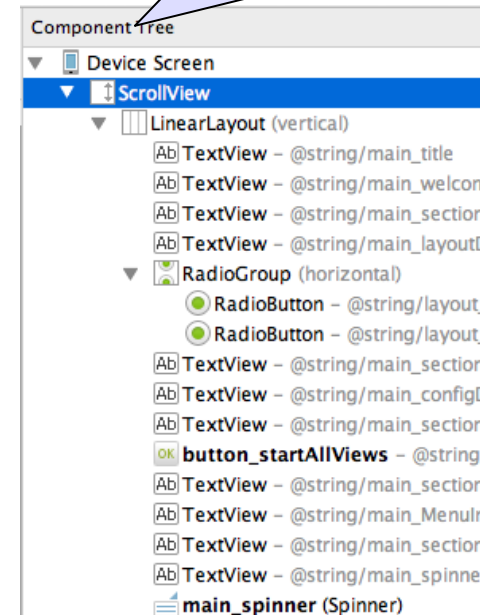
```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:padding="@dimen/padding">

        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:paddingBottom="@dimen/padding"
            android:text="@string/main_title"
            android:textSize="@dimen/textSizeTitle" />

        <TextView
            android:layout_width="match_parent"
```

Android Studio: Ansicht „Component Tree“ vom Standard-XML-Editor gibt schnelle Überblick über GUI-Struktur (Layout-Hierarchie)



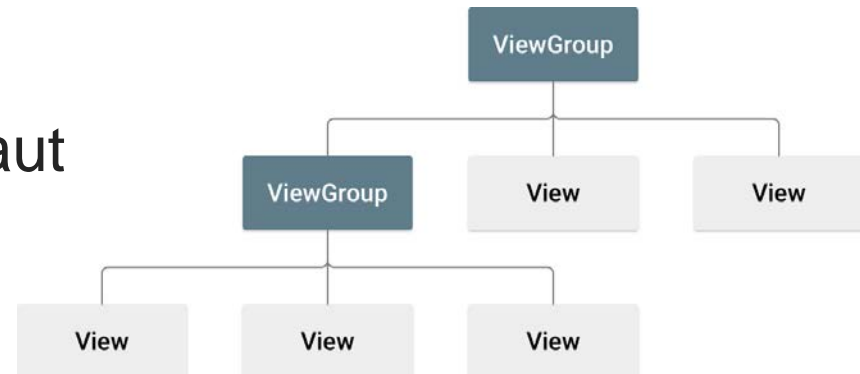
## ■ Layout-Parameter?

- Je nach Layout müssen die enthaltenen Elemente unterschiedlich konfiguriert werden (unterschiedliche Layouts unterstützen unterschiedliche Parameter)
- <http://developer.android.com/reference/android/view/ViewGroup.LayoutParams.html>

Bei Arbeit mit Layout-Editor nicht offensichtlich, trotzdem gut zu wissen!

# Grundkonzepte Android UI

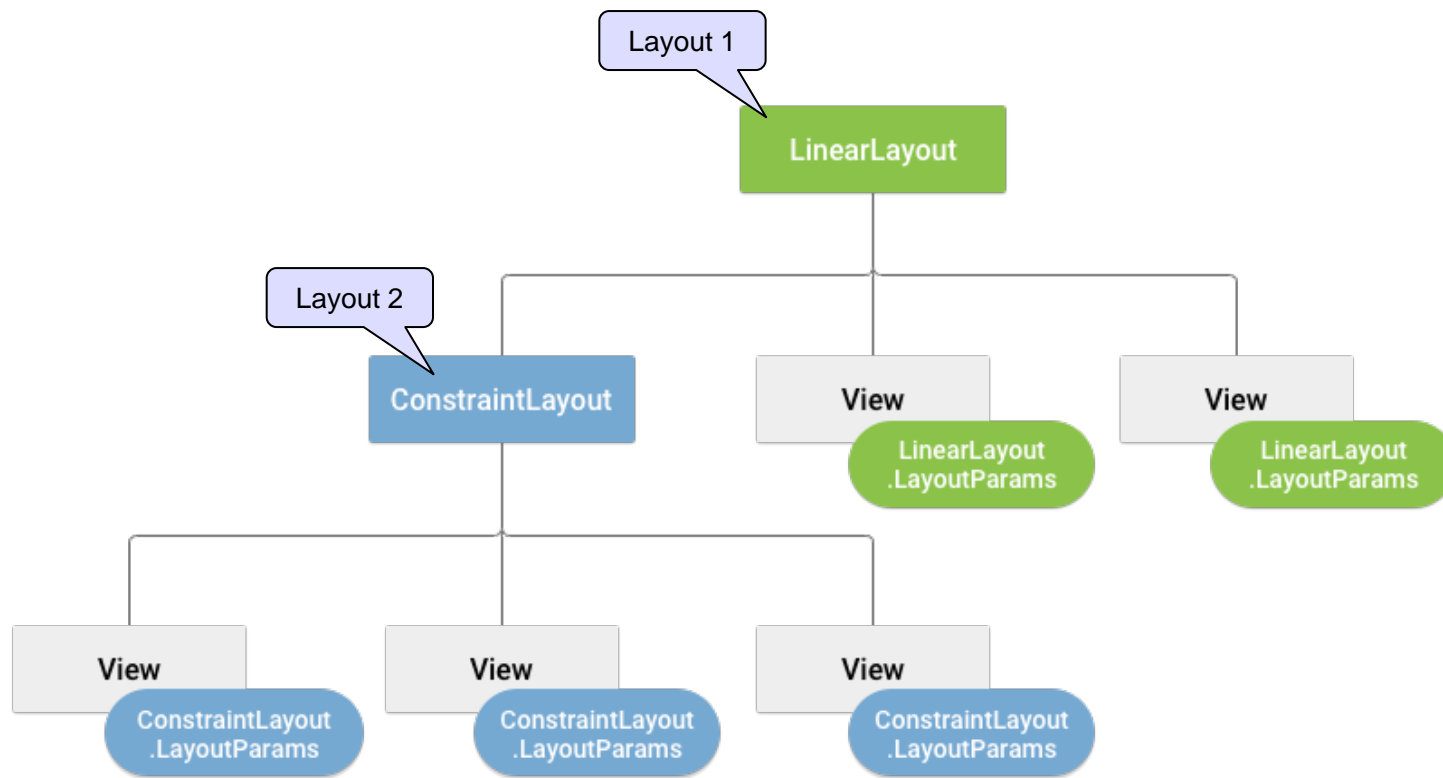
- Ein Android UI...
  - ist **hierarchisch** aufgebaut
  - besteht aus
    - ViewGroups
      - Behälter für Views und andere ViewGroups
      - Anordnung durch ein Layout
    - Views (Widgets)
  - sollte auf unterschiedlichen Bildschirmgrößen gleich aussehen!



Sehr wichtige Bedingung! D.h. Elemente werden eigentlich *nie absolut*, sondern *relativ* positioniert!

<https://developer.android.com/training/multiscreen/screensizes.html>

# Beispiel Hierarchie von ViewGroups & Views



- Schachtelung führt zu Ziel-Layout, aber oft nicht effizient
- Wenn möglich *Constraint-Layout* verwenden

Es gibt noch mehr, die behandeln wir aber nicht:  
Constraint Layout = one for all

# Android-Studio: Übersicht Views & ViewGroups

The image displays a collage of screenshots from the Android Studio interface, specifically focusing on the 'Palette' window which lists various Android Views and ViewGroups. The screenshots show different categories being selected, such as 'Common', 'Text', 'Buttons', 'Widgets', 'Layouts', 'Containers', 'Google', and 'Legacy'. Specific views highlighted include:

- Common:** TextView, Button, ImageView, RecyclerView, ScrollView, Switch, Spinner, RecyclerView, ScrollView, HorizontalScrollView, NestedScrollView, ViewPager, CardView, AppBarLayout, NavigationView, BottomNavigationView, Toolbar, TabLayout, TabItem, ViewStub, <include>, <fragment>, NavHostFragment, <view>, GridLayout, ListView, TabHost, RelativeLayout, and GridView.
- Text:** Plain Text, Password, Password (Numeric), E-mail, Phone, Postal Address, Multiline Text, Time, Date, Number, Number (Signed), Number (Decimal), AutoCompleteTextView, MultiAutoCompleteTextView, CheckedTextView, and TextInputLayout.
- Buttons:** ImageButton, CheckBox, RadioGroup, RadioButton, ToggleButton, Switch, and FloatingActionButton.
- Widgets:** Image, Web, Video, Calendar, Progress, Progress (Horizontal), Seek, Seek (Discrete), Rating, Search, Texture, Surface, Horizontal Divider, and Vertical Divider.
- Layouts:** Guideline (horizontal), Guideline (vertical), LinearLayout (horizontal), LinearLayout (vertical), FrameLayout, TableLayout, TableRow, and Space.
- Containers:** RecyclerView, ScrollView, HorizontalScrollView, NestedScrollView, ViewPager, CardView, AppBarLayout, NavigationView, BottomNavigationView, Toolbar, TabLayout, TabItem, ViewStub, <include>, <fragment>, NavHostFragment, <view>, GridLayout, ListView, TabHost, RelativeLayout, and GridView.
- Google:** MapView.
- Legacy:** None.



# Layout-Spezifikation: 2 Optionen

Verwenden wir in diesem Modul grundsätzlich

## ■ **Statisch / Deklarativ (XML)**

Viele Vorteile: Deklarativ, weniger umständlich als Code, Struktur eminent, Umformungen ohne Rekompilierung möglich, ...

- Deklarative Beschreibung des GUI als Komponentenbaum
- XML-Datei im Verzeichnis `res/layout`
- Referenzen auf Bilder, Texte, usw.
- Typischerweise ein XML pro Activity (~Screen)

XML -> Java = Inflating


## ■ **Dynamisch (in Java)**

Jedes XML-Element hat eine korrespondierende Java-Klasse!

- Aufbau und Definition des GUI im Java-Code
- I.d.R. nicht nötig: die meisten GUIs haben fixe Struktur
- Aber die Änderung von Eigenschaften (z.B. Enablement, Visibility) zur Laufzeit ist normal

Häufigste Layout-relevante Änderung zur Laufzeit:  
Ausblenden einer View, wenn nicht benötigt

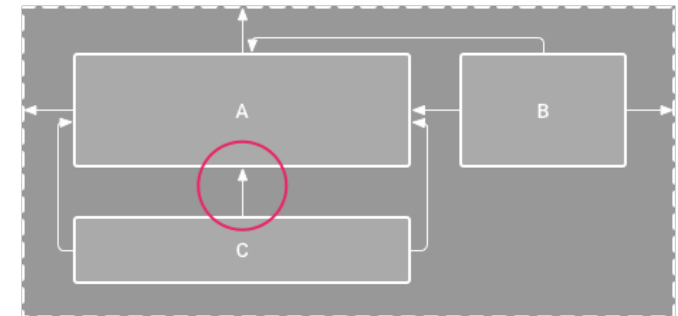
# XML Layout

- Jedes Layout ist ein eigenes File
  - Root-Element = View oder ViewGroup
  - Kann Standard- wie eigene View-Klassen enthalten
- XML können mit Inflater «aufgeblasen» werden, d.h. instanziiert werden
  - Auf diese Weise können eigene, wiederverwendbare Komponenten, Templates, Prototypen erzeugt werden
- Innere Elemente können unterhalb von einem Parent via View-ID referenziert werden  findViewById()-Funktion
- Debugging mit Layout-Inspector  
<https://developer.android.com/studio/debug/layout-inspector.html>

# Beispiel 1: Constraint Layout

Eingeführt mit Android 6,  
Rückwärtskompatibel bis 2.3

- «One for all» Lösung
  - Erstellung von komplexen Layouts, ohne zu schachteln
  - Elemente werden relativ mit «Bedingungen» platziert
    - zu anderen Elementen
    - zum Parent-Container
    - Element-Chains (spread/pack)
- Layout-Hilfen
  - Hilfslinien
  - Barriers



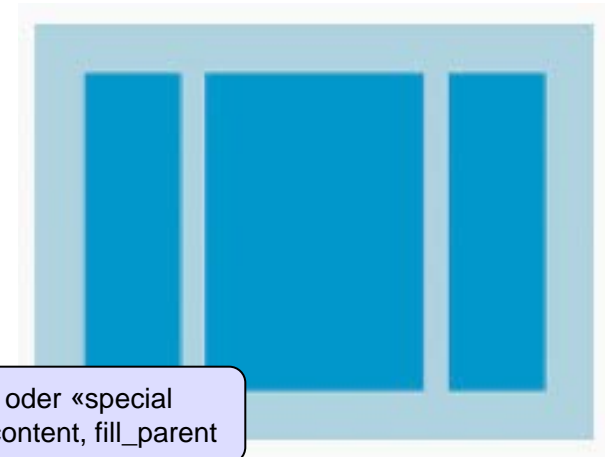
Video und Erklärung  
im Detail

Siehe <https://developer.android.com/training/constraint-layout/index.html>  
und <https://medium.com/exploring-android/exploring-the-new-android-constraintlayout-eed37fe8d8f1>

## Beispiel 2: LinearLayout

- Reiht Elemente neben-/untereinander auf
  - Kann geschachtelt werden, um Zeilen/Spalten zu formen
- Eigenschaften
  - orientation, gravity, weightSum
- Layout-Parameter für Children
  - layout\_width, layout\_height
  - layout\_margin, ...
  - layout\_weight, layout\_gravity

Aber nicht zu tief, sonst schlechte Performanz!



Grösse (12dp) oder «special constant»: wrap\_content, fill\_parent

Platzierung der View in der Grid-Cell, Aufteilung von zuviel Platz an Kinder

Siehe <http://developer.android.com/reference/android/widget/LinearLayout.LayoutParams.html>

# Demo LinearLayout (Siehe Übung 2)

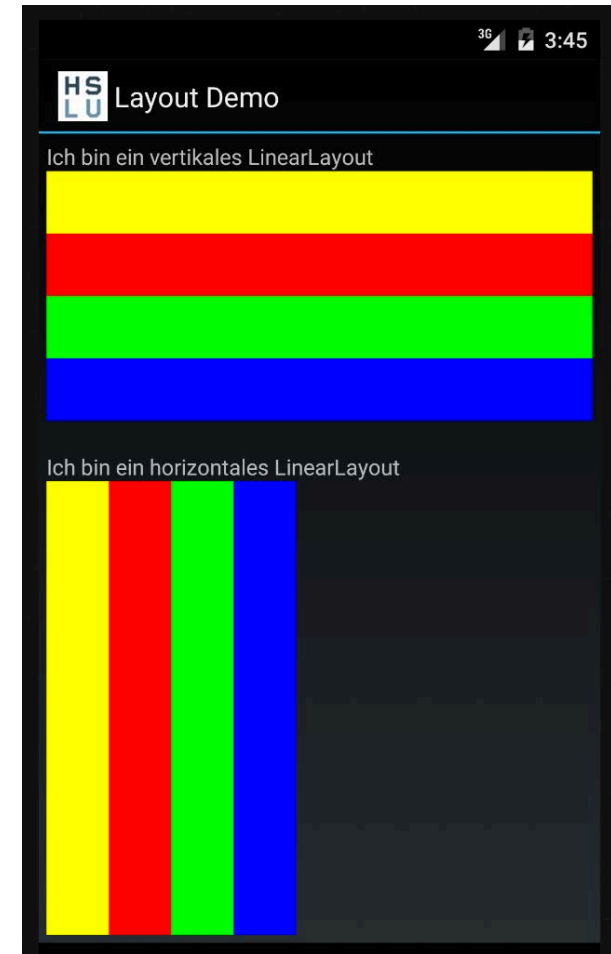
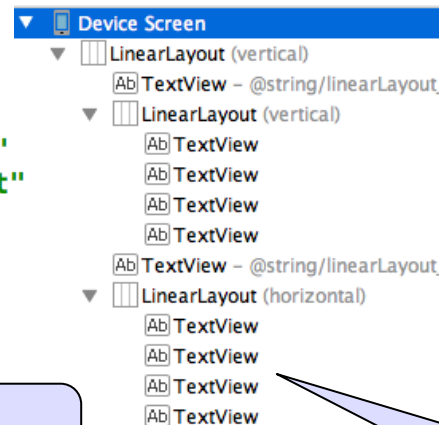
- Geschachtelte lineare Layouts
  - Vertikal und horizontal
- Inhalt: TextViews
  - Inkl. Hintergrundfarbe

```
<TextView  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:background="@color/yellow"  
    android:height="40dp" />
```

Fixe Höhe/Breite! «dp»?

Farbe selber festgelegt in colors.xml:

```
<resources>  
    <color name="green">#00ff00</color>
```



„4 TextViews = 4 Spalten“

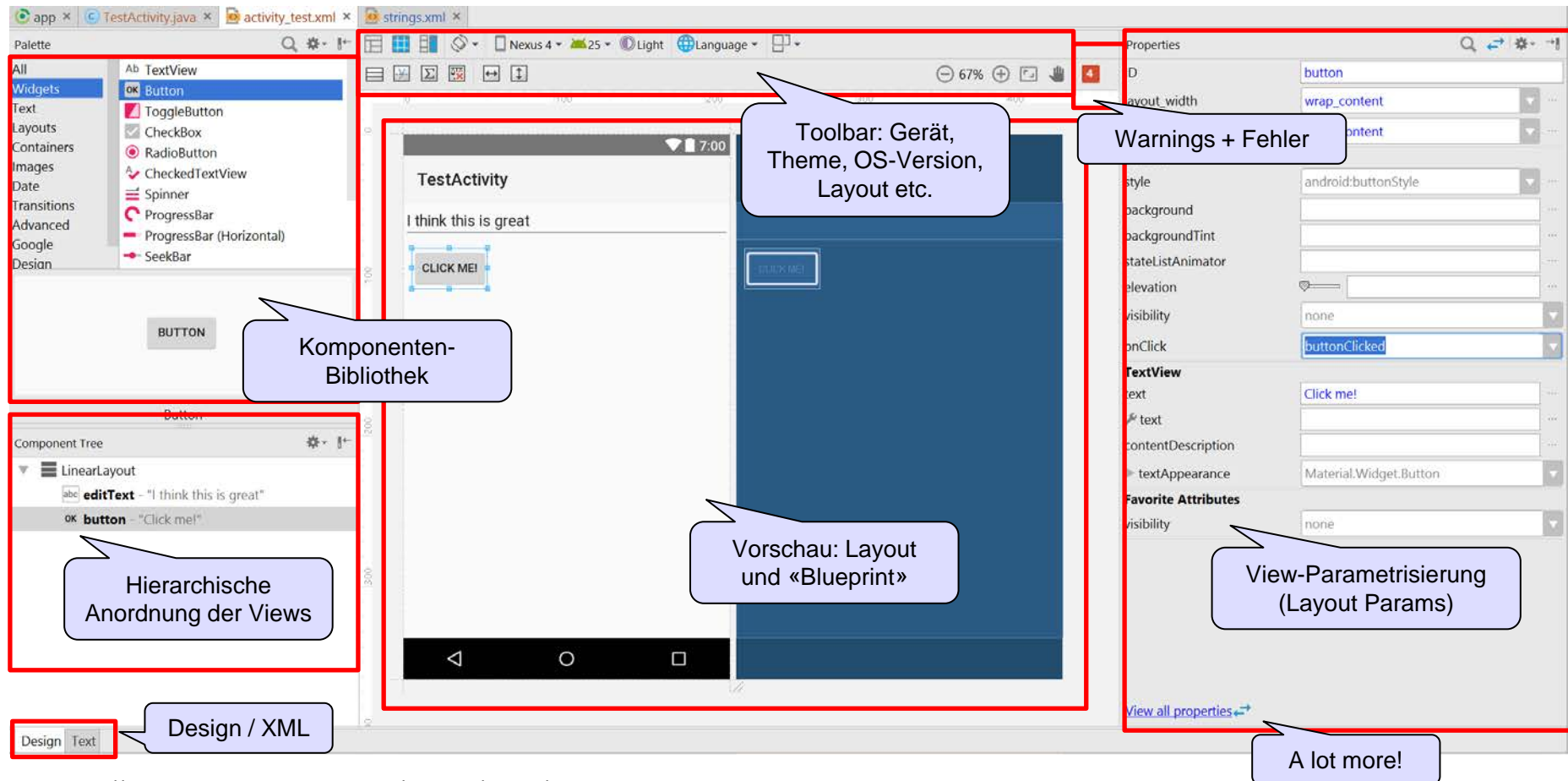
# Warum noch Linear Layout?

- Nach wie vor einfachste Lösung für
  - Button- oder Action-Bars («flow Semantik»)
  - Sehr einfache Screens
- Kaum Konfiguration nötig, robust
- Für scrollbare Listen mit dynamischer Anzahl Elemente besser ListView verwenden! (siehe Adapter-Views)
- Einsatz mit Bedacht durchaus sinnvoll

*Constraint layout does it all?!*

# Android Studio Layout Editor (Demo)

## Android Studio enthält einen mächtigen Layout-Editor



<https://developer.android.com/studio/write/layout-editor.html>

# Layout: Pixel-Angaben: dp, sp & px

Wir verwenden typischerweise Angaben in dp, ausser sp für Schriftgrößen

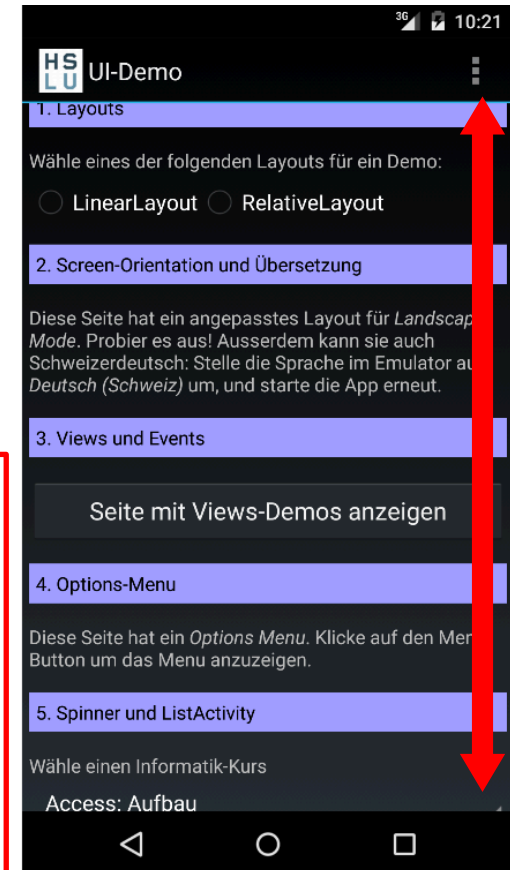
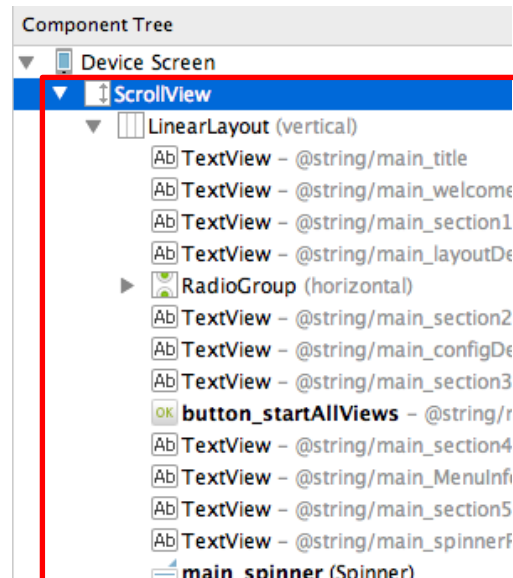
- **dp: Density-independent Pixels** - An abstract unit that is based on the physical density of the screen. These units are relative to a 160 dpi (dots per inch) screen, on which 1dp is roughly equal to 1px. [...] Using dp units (instead of px units) is a simple solution to making the **view dimensions in your layout resize properly** for different screen densities. In other words, it provides consistency for the real-world sizes of your UI elements across different devices.
- **sp: Scale-independent Pixels** - This is like the dp unit, but it is also scaled by the user's font size preference. It is **recommend you use this unit when specifying font sizes**, so they will be adjusted for both the screen density and the user's preference.
- **px: Pixels** - Corresponds to actual pixels on the screen. This unit of measure is **not recommended** because the actual representation can vary across devices; each devices may have a different number of pixels per inch and may have more or fewer total pixels available on the screen.

<http://developer.android.com/guide/topics/resources/more-resources.html#Dimension>



# ScrollView & Demo

- Spezielle ViewGroup, die (vertikales) Scrolling bei zu grossen Layouts erlaubt
  - Kann nur ein Kind haben (z.B. ein LinearLayout)
- Enthält typischerweise das Top-Level-Layout einer Bildschirmseite
- Siehe Übung 2





<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

## Ressourcen, Konfigurationen & Internationalisierung

# Q: Was sind Ressourcen?

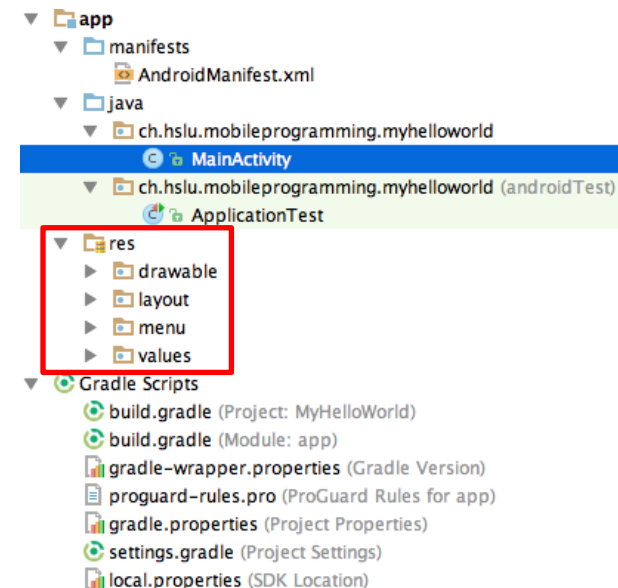
Im Ordner `res/`  
des Projekts

- A: Alle Nicht-Java Teile einer Applikation
  - Ausgelagerte Konstanten-Definitionen
  - Im Layout und Java-Code referenziert über **automatisch generierte R-Klasse** mit ID-Konstanten (int)
  - Kontextabhängige Ressourcen sind möglich
    - z.B. spezifisch für Sprache, Gerätetyp und Orientierung, ...

D.h. übersetzbar

## ■ Beispiele

- Strings, Styles, Colors, Dimensionen
- Bilder (drawables)
- Layouts (portrait, landscape)
- Array-Werte (z.B. für Spinner) und Menu-Items



# Referenzierung von Ressourcen

- Referenzierung in XML-Datei
  - Bsp.: Strings und Color im Layout (xml) mit @

```
<TextView  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:layout_marginBottom="@dimen/marginBottom"  
    android:background="@color/sectionBackground"  
    android:padding="@dimen/padding"  
    android:text="@string/main_section1"  
    android:textColor="@color/sectionText" />
```

- Referenzierung im Code: immer über die R-Klasse!
  - R-Klasse wird bei Build automatisch generiert

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
}
```

Für jede gefundene  
Resource ein Eintrag

# Spezifische Ressourcen

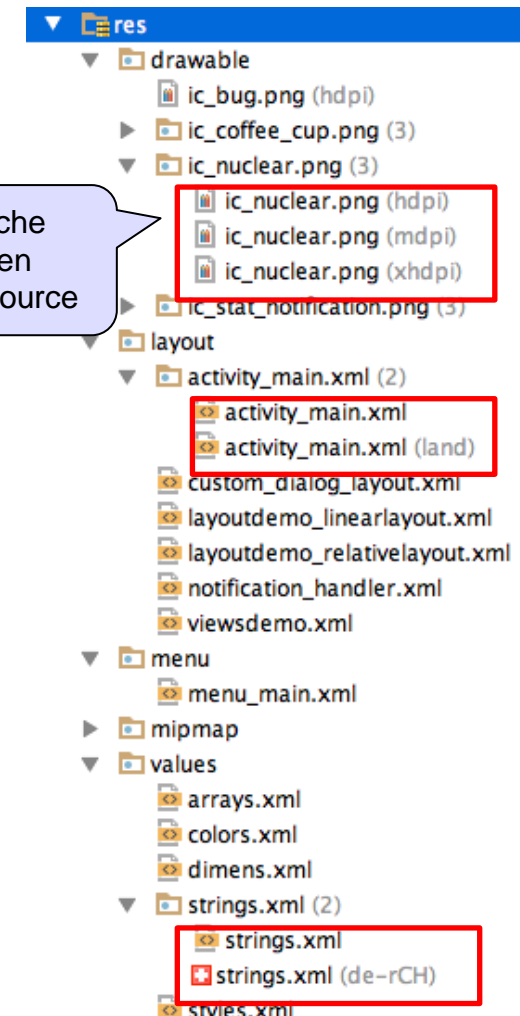
- Unterschiedliche Ausprägungen einer Resource für verschiedene Systemkonfigurationen
  - **Internationalisierung**
    - Komplette/teilweise Übersetzung (Strings + Bilder)
  - Bilder für unterschiedliche **Auflösungsklassen**
    - ldpi (~120dpi), mdpi (~160), hdpi (~240), xhdpi (~320),  
siehe [http://developer.android.com/guide/practices/screens\\_support.html#qualifiers](http://developer.android.com/guide/practices/screens_support.html#qualifiers)
  - Layouts für unterschiedliche **Orientierung** des Displays
    - landscape / portrait
  - Andere Ressource für ein bestimmtes **HW-Modell**
    - HTC, Samsung, LG, Sony, ...

Achtung! Die Konfiguration kann zur Laufzeit wechseln! (z.B. Sprache)

# Ressourcen-Organisation

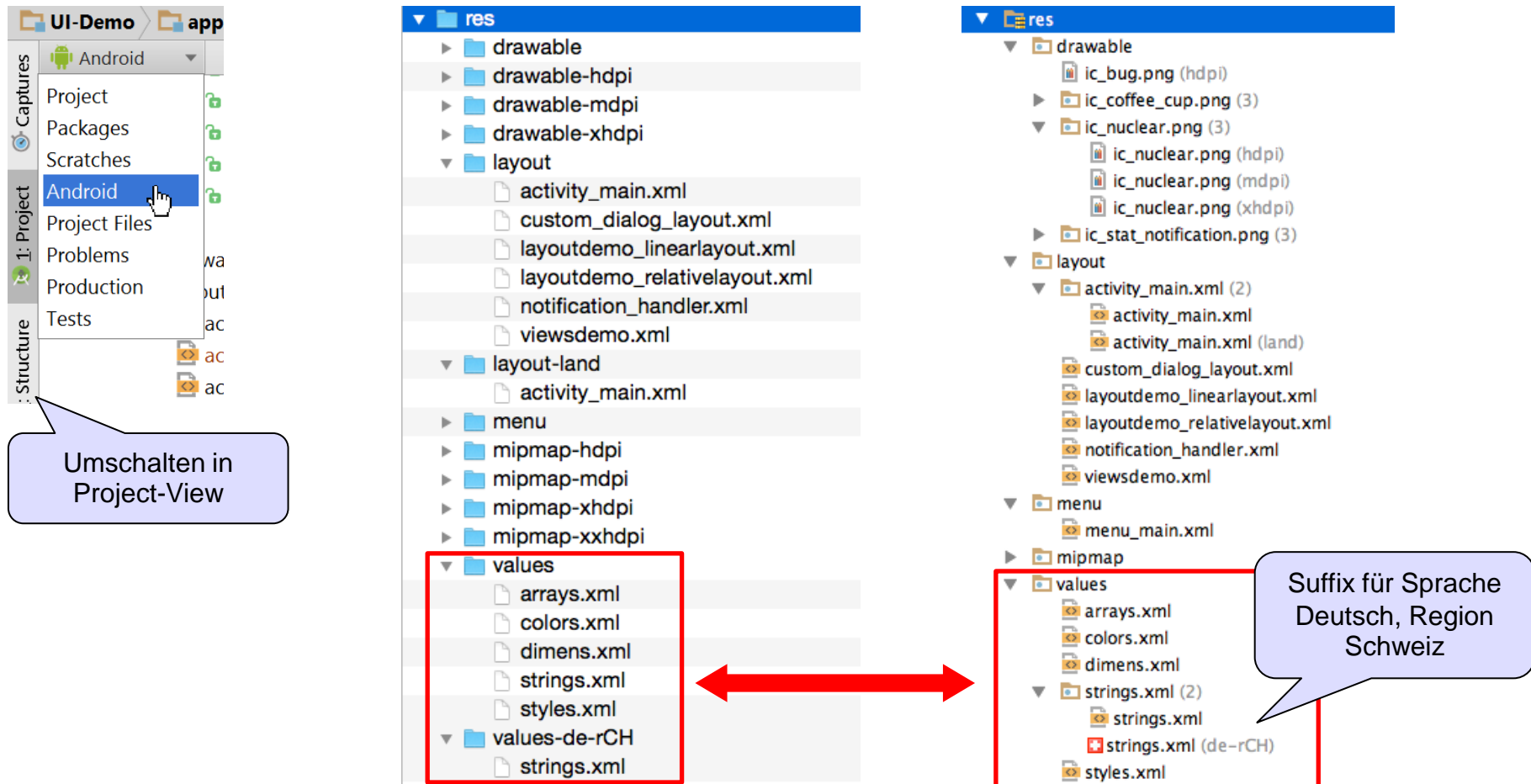
- Default-Verzeichnisse unterhalb von res/
  - drawable, layout, menu, values, ...
  - Inhalt gilt im allgemeinen Fall
- Spezifische Konfigurationen
  - Kopien der Default-Verzeichnisse
    - Name ergänzt durch Suffix, z.B. –de
  - Override für eine spez. Konfiguration
    - D.h. Inhalt ersetzt Default
    - Lookup: von spezifisch nach allgemein

Unterschiedliche  
Ausprägungen  
derselben Ressource



# Darstellung in Android Studio

Android Studio zeigt vereinfachte Sicht, entspricht nicht 1:1 dem Dateisystem:





# Konfigurations-Demo: Übersetzung (Siehe Übung 2)

- Ordner values-de-rCH

...oder auch  
values-fr, usw.

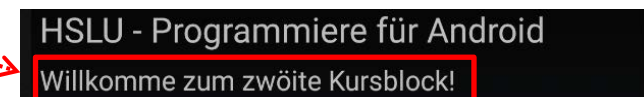
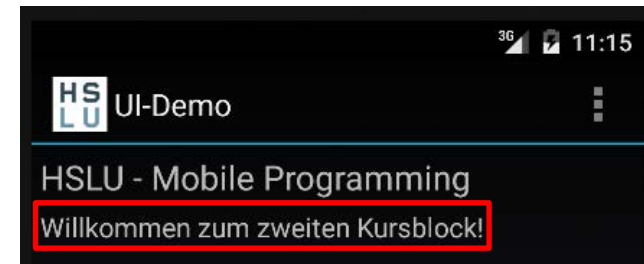
- Datei strings.xml:

```
<string name="main_welcomeText">Willkommen zum zweiten Kursblock!</string>
<string name="main_welcomeText">Willkomme zum zwöite Kursblock!</string>
```

- Zugriff aus layout.xml:

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="15dp"
    android:paddingBottom="12dp"
    android:text="@string/main_welcomeText"
    android:textSize="16sp" />
```

Gibt  
String  
gemäss  
aktueller  
Locale  
zurück

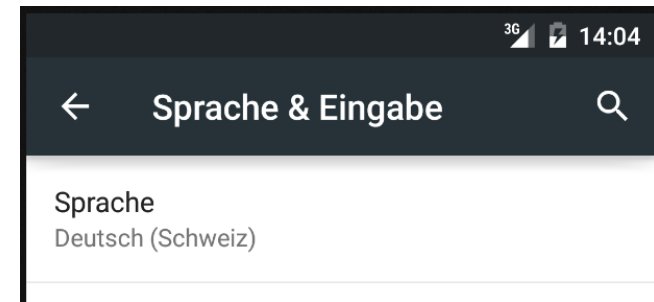


Zugriff aus Code: Resources.getString(id)

- z.B.: getResources().getString(R.string.main\_title);



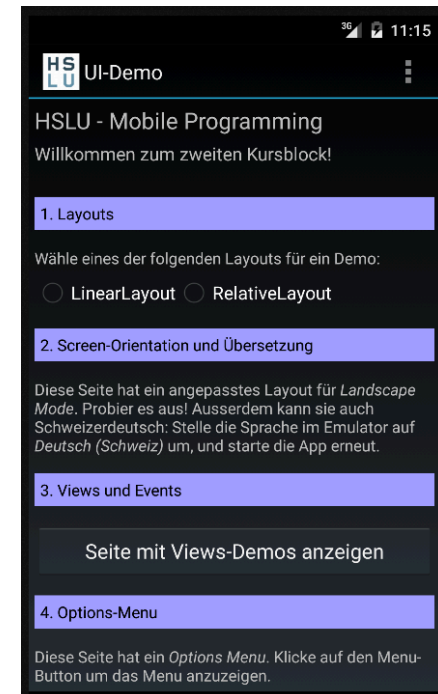
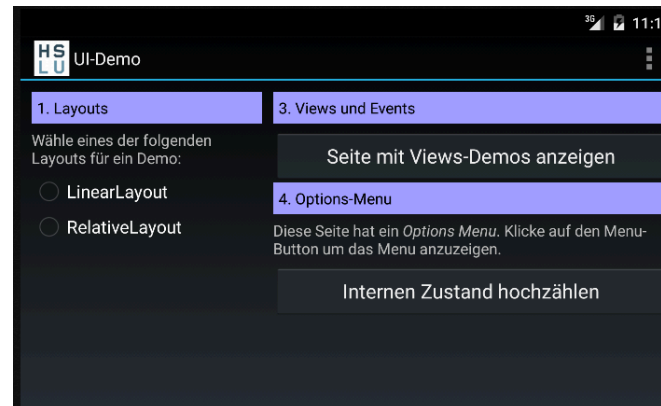
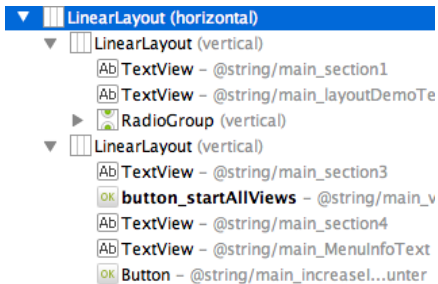
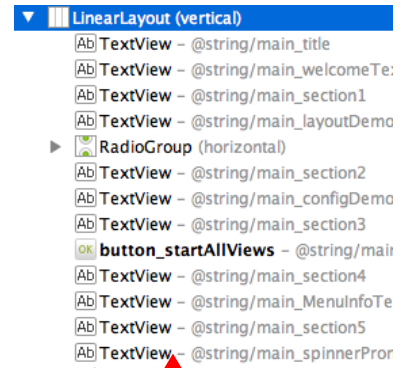
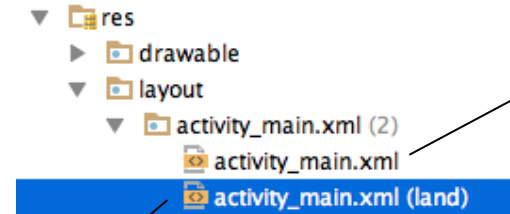
# Sprachwahl und -einstellung



- Im Emulator via Einstellungen die Sprache auf „Deutsch (Schweiz)“ stellen
  - > *Settings* > *System* > *Sprache & Eingabe* > *Sprache*
- ISO Country/Region und Language Codes:
  - [http://en.wikipedia.org/wiki/List\\_of\\_ISO\\_639-1\\_codes](http://en.wikipedia.org/wiki/List_of_ISO_639-1_codes) (Sprachen)
    - z.B. en, fr, it, de, ...
  - [http://en.wikipedia.org/wiki/ISO\\_3166-1](http://en.wikipedia.org/wiki/ISO_3166-1) (Länder)
- ...Auflösungsreihenfolge?
  - Siehe <http://developer.android.com/guide/topics/resources/providing-resources.html#BestMatch>

# Konfiguration-Demo: Landscape (Siehe Übung 2)

- Ordner layout-land
  - activity\_main.xml
- Screen-Orientierung umstellen
  - Emulator: Ctrl-Left/Right (oder Toolbar)
    - Siehe <http://developer.android.com/tools/help/emulator.html>
  - HW-Device: Gerät drehen





<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

## UI-Event-Handling

# Interaktion mit dem GUI

- Views in Java
  - Jedes View-Element hat eine entsprechende Java-Klasse
  - Gilt auch für ViewGroups!
  - Layout kann/könnte auch in Java programmiert werden
- Interaktion mit dem GUI im Code
  - Im Code können Views mit `findViewById(R.id.myView)` (Instanzmethode Activity) gesucht werden
  - Die APIs der einzelnen View-Klassen sind unter <http://developer.android.com/reference/android/widget/package-summary.html> ausführlich beschrieben

dynamisch

ID muss im Layout-XML definiert sein!  
`android:id="@+id/myView"`

# Beispiel: Zugriff auf TextView

- Voraussetzung: View ist im Layout (XML-Datei) mit einer ID versehen

```
<TextView  
    android:id="@+id/message_label"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content" />
```

- Zugriff auf View im Code (Activity-Klasse):

```
// Show message on dedicated text view  
private void displayMessage(String message) {  
    TextView label = (TextView)findViewById(R.id.message_label);  
    label.setText(message);  
}
```

# GUI-Events

- Entwurfsmuster: Observer / Listener
  - Listener für entsprechenden Event bei View registrieren, z.B. bei `Button myButton`:
    - `myButton.setOnClickListener(listener)`
- Event- und Listener-Typen
  - `OnClickListener`, `OnLongClickListener`, `OnKeyListener`, `OnTouchListener`, `OnDragListener`, ...
  - `public static Interfaces der Klasse View`

# Beispiel: Event-Handling im Code

- Ziel: Auf Klick-Event von Knopf reagieren
- Button muss ID haben im layout.xml

```
<Button  
    android:id="@+id/question_button_done"  
    android:text="@string/question_button_text"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content" />
```

- Registrierung eines Listeners an View im Code:

```
Button button = (Button) findViewById(R.id.question_button_done);  
button.setOnClickListener(new OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        // handler code  
        buttonClicked();  
    }  
});
```

Anonyme innere Klasse.  
– Andere Möglichkeiten?

# Spezialität: onClick-Event-Registrierung in XML

- Definition onClick-Handler im XML Layout

```
<Button  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:onClick="increaseInternalCounter"  
    android:layout_marginBottom="@dimen/marginBottom"  
    android:text="@string/main_increaseInternalCounter" />
```

Geht so nur für  
onClick-Events!

- Implementierung onClick-Handler-Methode in Activity

```
public void increaseInternalCounter(View button) {  
    // ...handler code...  
}
```

Mehrere Views können auf  
dieselbe onClick-Handler-  
Methode verweisen!

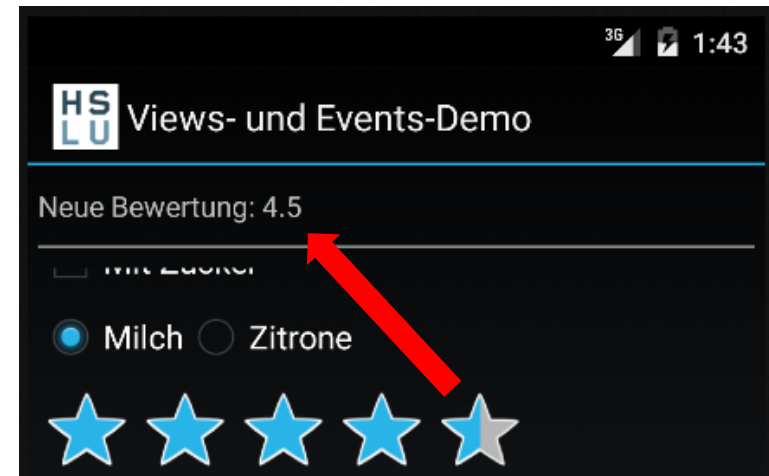
- Signatur-Vorgabe: `public void name(View v)`



# Demo: Event Handling (Siehe Übung 2)

- Beispiel: RatingBar
- Im Layout.xml:

```
<RatingBar  
    android:id="@+id/viewsDemo_ratingBar"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:numStars="5"  
    android:rating="4"  
    android:layout_marginBottom="5dp" />
```



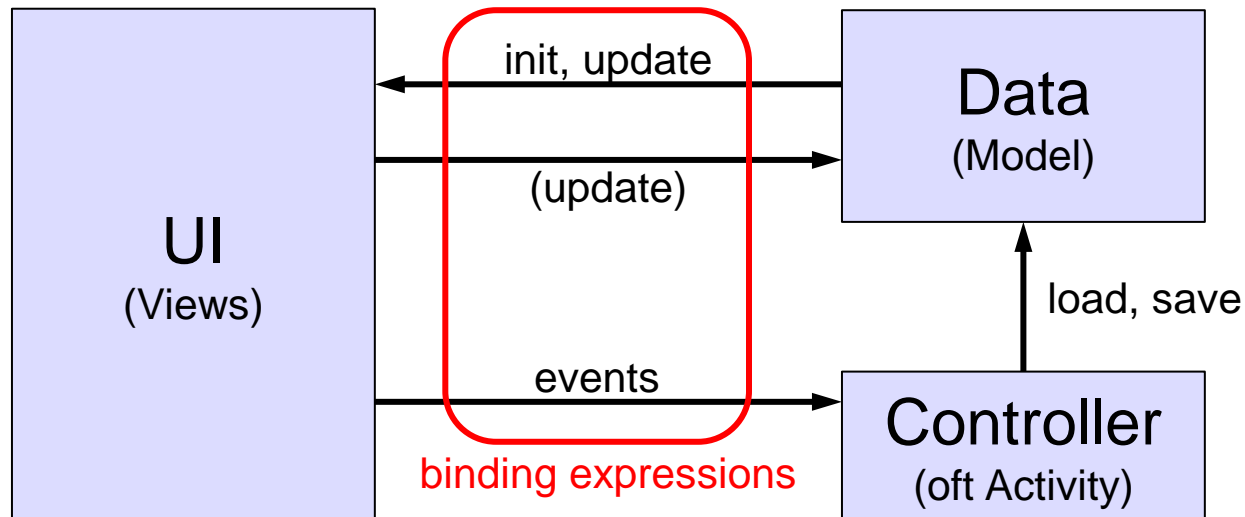
- Listener registrieren im Code der Activity-Klasse:

```
RatingBar ratingBar = (RatingBar) findViewById(R.id.viewsDemo_ratingBar);  
ratingBar.setOnRatingBarChangeListener(new RatingBar.OnRatingBarChangeListener() {  
  
    public void onRatingChanged(RatingBar ratingBar, float rating, boolean fromUser) {  
        logLabel.setText("Neue Bewertung: "+rating);  
    }  
});
```

Was dürfte logLabel sein?  
Wie kommt man zu dieser Referenz?

# Data Binding

- **Vorteil:** Separiert UI und Daten
- Synchronisiert UI mit Daten (1-, resp. 2-way-binding)
- Verwendet «binding expressions» mit `@{..}` Syntax im Layout-File, um View-Attribute zu initialisieren



# Data Binding: Beispiel

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android">
    <data>
        <variable name="model" type="org.example.MyModel"/>
    </data>
    <LinearLayout ...>
        <Button
            android:id="@+id/button"
            ...
            android:enabled="@{model.user.role == `admin`}"
            android:text="@{model.buttonText}"
            android:onClick="@{() -> model.increaseClickCount()}" />
        <EditText
            android:id="@+id/input"
            ...
            android:text="@={model.inputText}" />
    </LinearLayout>
</layout>
```

Definition Layout-Variablen

Data binding (1-way)

Event binding

Data binding (2-way)

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ActivityMainBinding binding = DataBindingUtil.setContentView(...);
    model = new MainModel();
    model.load();
    binding.setModel(model);
}
```

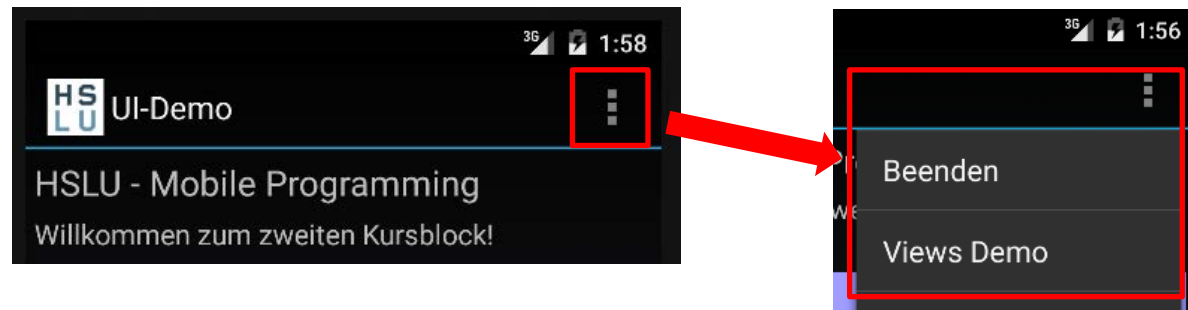
Binden der Layout-Variablen auf effektive Daten  
(z.B. **ViewModel** mit **Observables**)



<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

## Options-Menu

# Options-Menu



- Android-Apps können in der Action-Bar rechts oben ein Menu mit Optionen anbieten
- Erzeugung durch Aufruf Hook in der Activity-Klasse:
  - `onCreateOptionsMenu(Menu menu)`
    - Hier kann Menu mit Einträgen bestückt werden
    - `MenuInflater` + XML benutzen oder Java oder beides...
- Bei Klick auf Eintrag Aufruf eines anderen Hooks:
  - `onOptionsItemSelected(MenuItem item)`

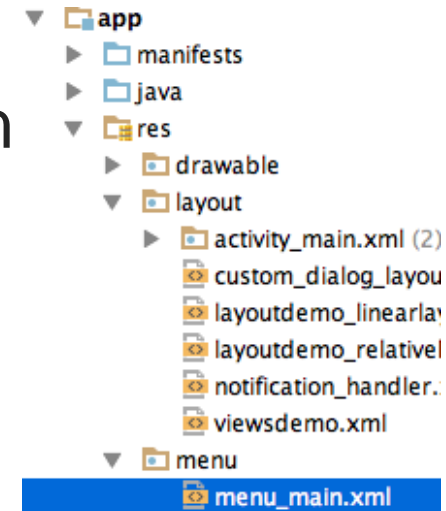
# Beispiel: Options-Menu & Inflater

## 1. Ordner `res/menu` mit `.xml`-Datei anlegen

- Dateiname z.B. `main_menu.xml`

## 2. Menu und Items in XML definieren

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:tools="http://schemas.android.com/tools" tools:context=".MainActivity">
    <item
        android:id="@+id/main_menu_finish"
        android:title="@string/menu_finish">
    </item>
    <item
        android:id="@+id/main_menu_startAllViews"
        android:title="@string/menu_startViewsDemo">
    </item>
</menu>
```



## 3. Menu „aufblasen“ mit `MenuInflater`

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.menu_main, menu);
    return true;
}
```

Auf jeder Activity verfügbar

# Demo: Options-Menu

- Einträge aus XML erzeugen

- Siehe letzte Folie

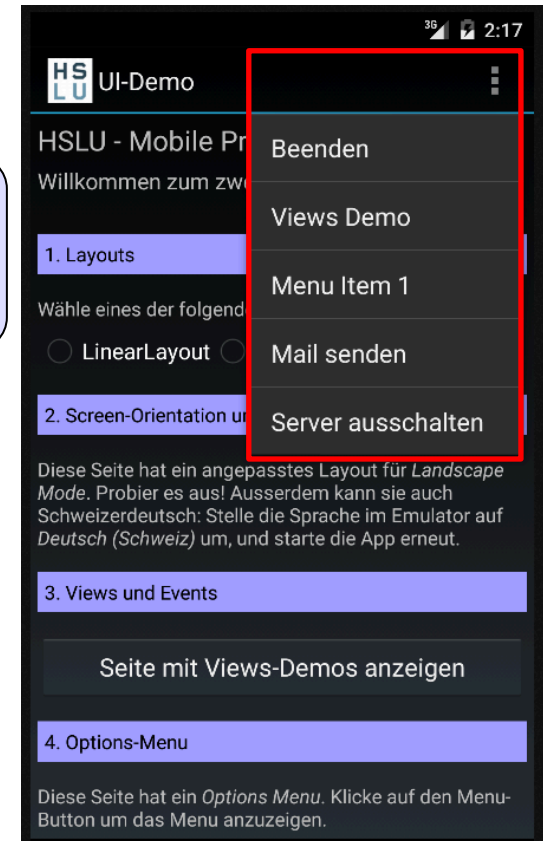
- Einträge programmatisch

```
menu.add(Menu.NONE, 239, Menu.NONE, "Menu Item 1");
menu.add(Menu.NONE, 333, Menu.NONE, getString(R.string.menu_mail));
menu.add(Menu.NONE, 923, Menu.NONE, R.string.menu_server);
```

...gesehen: Drei verschiedene Arten um zu einem String zu kommen ☺

- Event-Handling: Selektierung

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    if (super.onOptionsItemSelected(item)) {
        return true; // handled by super implementation
    }
    switch (item.getItemId()) {
        case R.id.main_menu_finish:
```





<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

## Adapter-Views



# Adapter

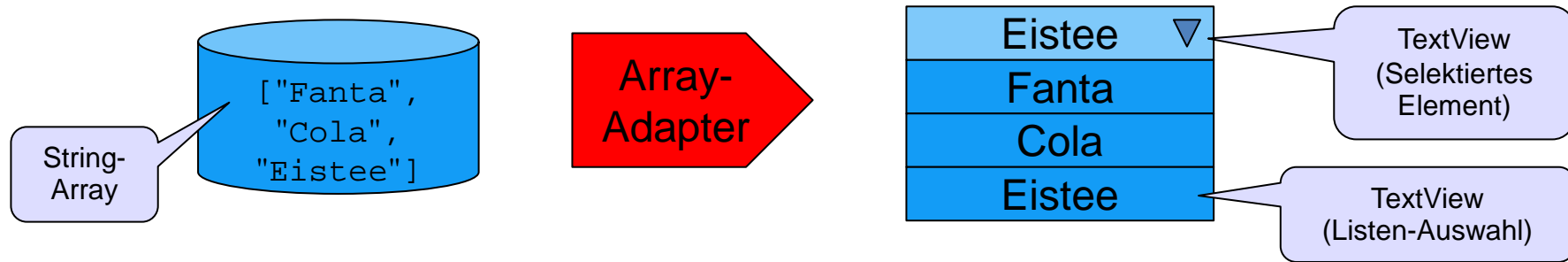
**Wichtiger Hinweis:** Wir behandeln hier nur das synchrone Laden von «kleinen», resp. «schnellen» Datenquellen. Für asynchrones Laden von «langsamen» oder «grossen» Datenquellen (z.B. vom Netz, oder potentiell grosse DB-Table) lese Doku über **Loaders**.

- Verbindung zwischen Datenquelle und GUI



- Zieht Datenquelle an und beliefert AdapterView
- Erzeugt (Sub-)Views *pro gefundenes Datenelement*
- Arten: **ArrayAdapter**, **ImageAdapter**, ...
- Transformiert Daten ggf. in benötigtes Ziel-Format
- Datenquellen:
  - String-Array, String-Liste, Bilder, Datenbank, ...

# Beispiel: ArrayAdapter



## ■ Aufgabe ArrayAdapter

- Anbindung von *irgend* einem Array oder einer Liste mit beliebig getypten Elementen an *irgend* eine AdapterView
- Für jedes Daten-Element wird eine SubView erzeugt
- Default: Erstellt TextView mit `element.toString()`-Wert

```
String[] myArray = new String[]{"Fanta", "Cola", "Eistee" };  
ArrayAdapter<String> adapter =  
    new ArrayAdapter<String>(this, android.R.layout.simple_list_item_checked, myArray);  
this.setListAdapter(adapter);
```

SubView-Template für Elemente  
(Aus Android Resources)  
Tipp: Ctrl-B in Android-Studio

Daten

Fanta  
Cola  
Eistee

# AdapterViews & ListActivity

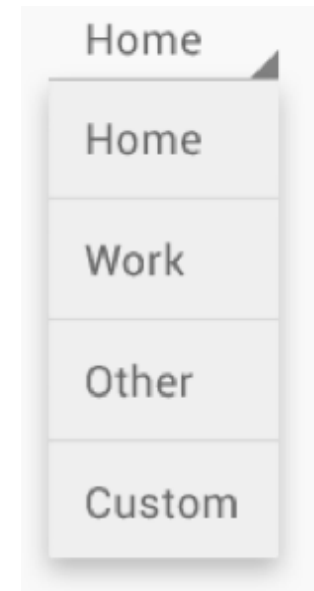
- **Adapter-Views:** Spezielle View-Klassen
  - Sind für die Zusammenarbeit mit Adaptern optimiert
    - Bsp: `ListView`, `GridView`, `Gallery`, `Spinner`, `Stack`, ...
  - Füllen Teile von sich mit von Adapter erzeugten Views
  - Leiten ab von  
`android.widget.AdapterView<T> extends android.widget.Adapter<T>`
- Spezielle Activity **ListActivity**:
  - Vordefiniertes Layout (enthält eine `ListView`)
  - Vordefinierte Callbacks (bei Auswahl einer List-Entry)
  - Bietet Zugriff auf aktuelle Selektion/Datenposition

D.h. kein XML nötig

# android.widget.Spinner

Als Variante gibt es noch die  
AutoCompleteTextView

- Auch: ComboBox oder DropDown-List genannt
- Zeigt ausgewähltes Element,  
bei Klick erscheint ein Menu mit Auswahl
- Daten auf Spinner setzen, 2 Varianten:
  - Im Code mit Adapter:  
`spinner.setAdapter(myAdapter)`
  - Im XML mit Angabe einer String-Array ID:  
`android:entries="@array/spinnerValues"`
- Listener setzen für Behandlung Auswahl:
  - `spinner.setOnItemClickListener(...)`



# Demo: Spinner (Siehe Übung 2)

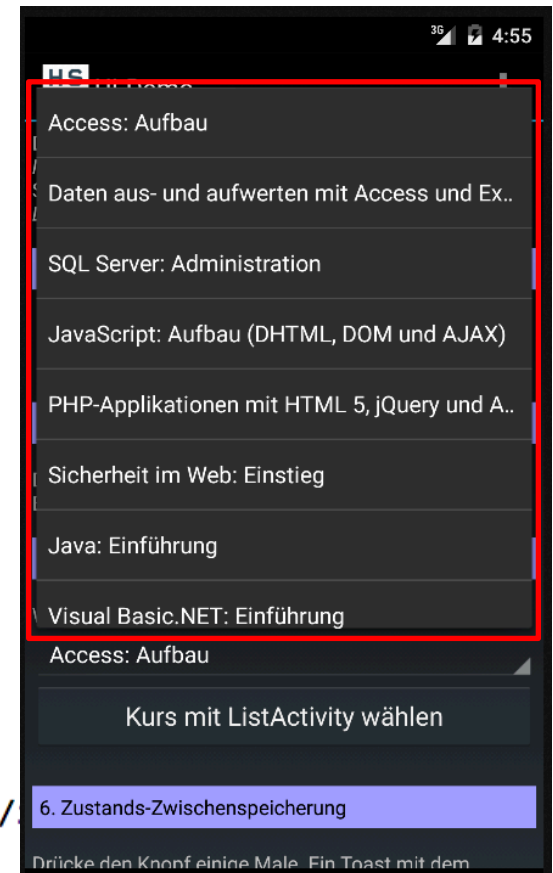
## ■ layout.xml

```
<Spinner
    android:id="@+id/main_spinner"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:entries="@array/itCourses"
    android:prompt="@string/main_spinnerPrompt" />
```

Beachte: Daten werden aus XML-Ressource geholt

## ■ arrays.xml

```
<resources>
    <string-array name="itCourses">
        <item>Access: Aufbau</item>
        <item>Daten aus- und aufwerten mit Access und Excel</item>
        <item>SQL Server: Administration</item>
        <item>JavaScript: Aufbau (DHTML, DOM und AJAX)</item>
        <item>PHP-Applikationen mit HTML 5, jQuery und AJAX</item>
        <item>Sicherheit im Web: Einstieg</item>
        <item>Java: Einführung</item>
    
```



## ■ In der Activtiy-Klasse

```
spinner.setOnItemClickListener(new OnItemSelectedListener() {
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        String selectedItem = (String) parent.getItemAtPosition(position);
```

Position der View in „ParentView“

Zeilen-ID des gewählten Werts bei DB-Query

# android.widget.ListView

- Zeigt Liste von Views/Items zur Auswahl
- Achtung: Braucht viel Platz!
  - Erhält meistens fast den ganzen Bildschirm zugeteilt
  - Verwendung i.d.R. zusammen mit `ListActivity`
- Konzeptionell identisch zu Spinner
  - Aber andere Darstellung auf UI
  - Verwendungsentscheid
    - Kurze Listen → Spinner
    - (Sehr) lange Listen → `ListView/ListActivity`
    - Falls User die möglichen Auswahlwerte kennt → `AutoCompleteTextView`
  - Adapter-/Datendefinition grundsätzlich gleich (d.h. im Code oder durch XML-Array)
  - Auswahlmodus: `setChoiceMode(ListView.CHOICE_MODE_*)`

Verwendung:

- Navigiere zu eigener `ListActivity`
- Auswahl > Resultat setzen > finish
- Auswertung Rückgabewert in Caller

Single/Multiselection

# android.app.ListActivity

- Spezielle Activity zur Darstellung einer ListView
- Vordefiniertes Layout (full-screen Liste)
  - `setContentView(...)` muss nicht aufgerufen werden
  - Aufruf i.d.R. mit `startActivityForResult(...)`
  - Vordefinierte vererbte Konfigurationsmethoden
    - `setListAdapter(adapter)` setzt die Daten für die Liste
    - `getListView()` erlaubt Zugriff auf die ListView-Instanz
- Callback bei Auswahl

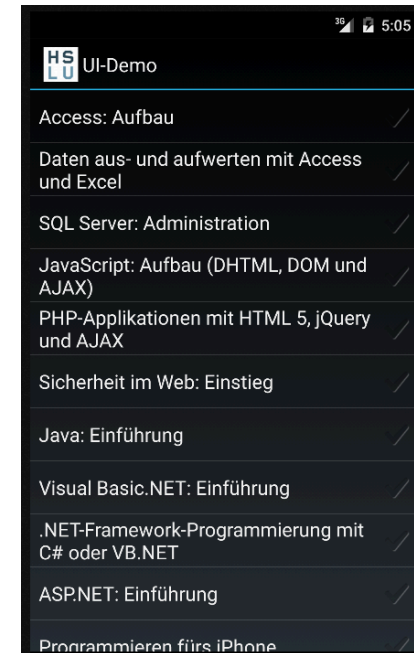
Anstatt `findViewById(..)` + Cast

  - `onListItemClick(parentView, view, position, id)` wird bei Auswahl aufgerufen (muss in Subklasse überschrieben werden, keine Listener-Registrierung nötig)

# Demo: ListView & ListActivity (Siehe Übung 2)

- Activity-Klasse (erbt von ListActivity!)
- Initialisierung der ListActivity mit Daten

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Attention: We do NOT set a layout! - ListActivity has already defined a layout.
    String[] courses = getResources().getStringArray(R.array.itCourses);
    ArrayAdapter<String> adapter =
        new ArrayAdapter<String>(this, android.R.layout.simple_list_item_checked, courses);
    this.setAdapter(adapter);
    ListView listView = getListView();
    listView.setChoiceMode(ListView.CHOICE_MODE_SINGLE);
}
```



- Reagieren auf Selektion

```
@Override
protected void onListItemClick(ListView parent, View view, int position, long id) {
    // define return value
    Intent result = new Intent();
    String selectedItem = (String) parent.getItemAtPosition(position);
    result.putExtra(EXTRA_CLASS_KEY, selectedItem);
    // set return value
    setResult(RESULT_OK, result);
    // finish the activity
    finish();
}
```

Position der View in  
„ParentView“

Zeilen-ID des  
gewählten Werts  
bei DQ-Query





<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

## Konfigurations- Wechsel & temporäre Datenspeicherung mit ViewModel

# Datenverlust bei Konfigurationswechsel

Was war das nochmal?

- Tatsache: Bei jedem Konfigurationswechsel wird die aktuelle Activity-Instanz zerstört und neu aufgebaut!
  - Typischer Fall: Wechsel Bildschirmorientierung
- Problem: Zustandsverlust!
  - Der Zustand aller Views mit einer ID wird automatisch gesichert und wieder hergestellt
  - Aber: Inhärenter Zustand der Activity (nicht auf Bildschirm sichtbar, in Feldern gespeichert) geht verloren!
    - Beispiele: Undo-Cache, Dropdown-Auswahl, internes Log
- Lösung: Verwendung eines **ViewModels**

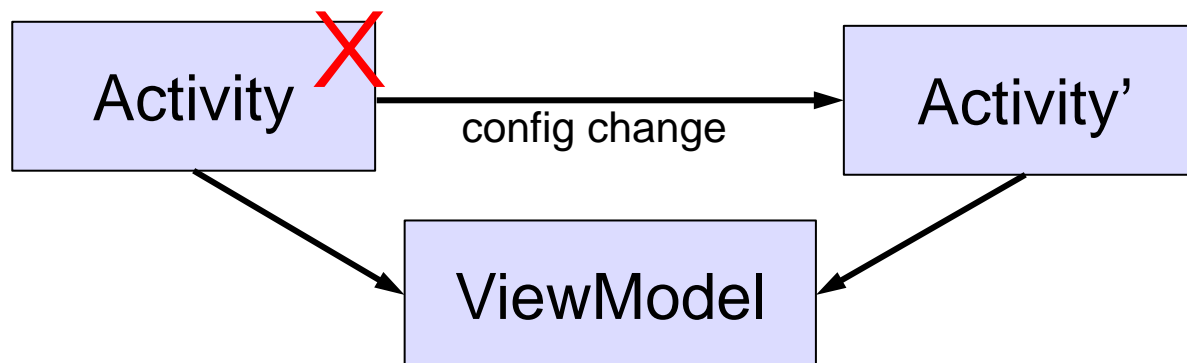
Mit Einschränkungen!

# ViewModel

- Kapselt UI-Daten so, dass sie bei Konfigurationsänderung einer Activity in-memory erhalten bleiben.
- **Vorteil:** Weniger Aufwand für Behandlung von Konfigurationsänderungen
- Lebensdauer mit Activity gekoppelt.

z.B. in onStop()

Wichtig: Für den Fall eines App-Kills durch OS muss immer noch persistiert werden!  
Hier ist VM keine Hilfe.



# ViewModel (2)

```
dependencies {
```

```
    implementation 'androidx.lifecycle:lifecycle-extensions:2.0.0' // ViewModel and LiveData
```

Zusätzliche Gradle dependency für  
ViewModel und Lifecycle Management

```
}
```

```
public class MainViewModel extends ViewModel {  
    private int counter = 0;  
  
    public int incrementCounter() { return ++counter; }  
  
    public int getCounter() { return counter; }  
}
```

ViewModel = normales POJO,  
ggf. mit Handler-Methoden

Wäre noch viel einfacher mit  
DataBinding! (out-of-scope)

```
// in MainActivity  
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    viewModel = ViewModelProviders.of(this).get(MainViewModel.class);  
  
    counterLabel = findViewById(R.id.main_label_counter);  
    updateCounterLabel();  
}  
  
// called on button click (see main.xml)  
public void increaseInternalCounter(View button) {  
    viewModel.incrementCounter();  
    updateCounterLabel();  
}
```

Erzeuge oder hole  
ViewModel-Instanz  
für diese Activity-  
Lebenszyklus-Instanz

Initialisierung UI aus ViewModel

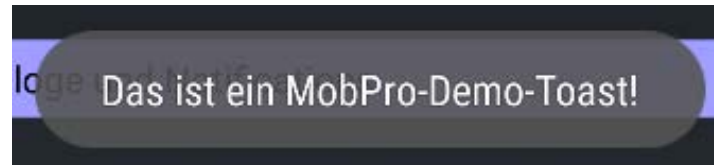
**Demo**



<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

## Rückmeldungen an den Benutzer: Toasts, Dialoge & Notifications

# Toast



- Kurze Rückmeldung (Popup) an Benutzer
  - Keine Interaktion möglich, verschwindet von selber nach einer gewissen Zeit
- Vorteil: Sehr einfach zu erstellen
- Nachteil: Wenn Benutzer nicht aufs Display schaut, während Toast angezeigt wird, geht die Information verloren
- Konfiguration
  - Text, Layout, Anzeigedauer (kurz/lang), Ort (gravity)

# Toasts Anzeigen: Code-Bsp.

- Default-Toast: Ein-Zeiler

...langer ;-)

```
Toast.makeText(getApplicationContext(), "Das ist ...", Toast.LENGTH_LONG).show();
```

Nur LENGTH\_LONG oder  
LENGTH\_SHORT möglich

Nicht  
vergessen!

- Toast mit anderem Anzeigort

- z.B. oben links:

```
Context context = getApplicationContext();  
Toast toast = Toast.makeText(context, "Toast links oben!", Toast.LENGTH_LONG);  
toast.setGravity(Gravity.TOP|Gravity.START, 0, 0);  
toast.show();
```

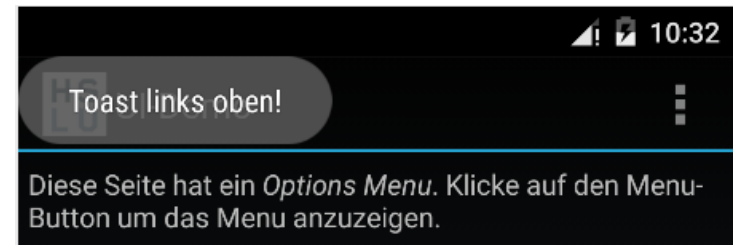
(x,y) Offset

- Toast mit eigenem Layout

- Siehe API-Doku:

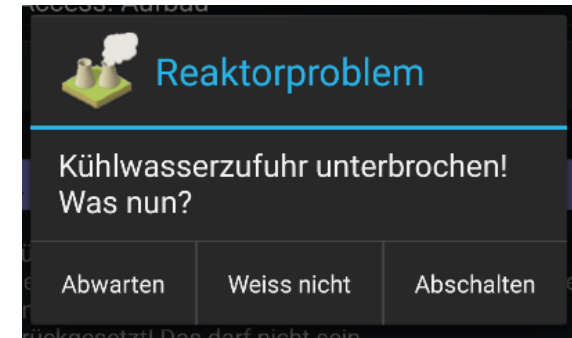
<http://developer.android.com/guide/topics/ui/notifiers/toasts.html#CustomToastView>

Das ist ein MobPro-Demo-Toast!



# Alert-Dialog

- Fenster mit Aktion für Benutzer
  - Information oder Eingabe von Daten
  - Interaktion möglich
  - Buttons: Positive, Neutral, Negative
- Vorteile
  - Kaum Einschränkungen punkto Darstellung
  - Vorbereitet für Anzeige von Daten
  - Verschwindet erst, wenn von Benutzer quittiert
- Konfiguration
  - Buttons, Titel, Icon, Nachricht
  - Inhalt: Liste von Items oder eigene View





# Alert-Dialog: Builder

- Builder-Muster für Erstellung von Alert-Dialog
- Vorgehen
  1. Builder erstellen: `new AlertDialog.Builder(this)`
  2. Builder konfigurieren: `setXXX` + Registrierung von `ClickListeners`
  3. Dialog erstellen: `Dialog dialog = builder.create()`
  4. Dialog anzeigen: `dialog.show()`
- Achtung! Anzeige von Dialogen ist **immer asynchron!**
  - Bei `show()` wird nicht gewartet (kein Rückgabewert)  
→ Behandlung von Benutzerselektion mit Listener

# Bsp. 1: Einfacher Alert-Dialog



## Reaktorproblem

Kühlwasserzufuhr unterbrochen!  
Was nun?

Abwarten

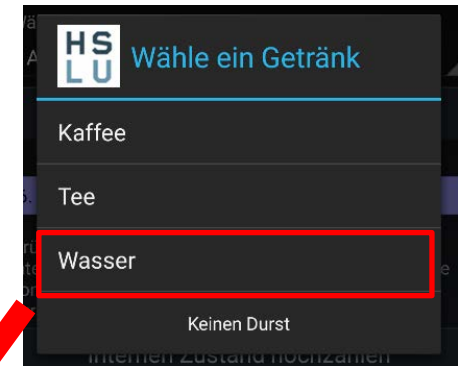
Weiss nicht

Abschalten

```
AlertDialog.Builder dialogBuilder = new AlertDialog.Builder(this);
dialogBuilder.setTitle("Reaktorproblem")
    .setIcon(R.drawable.ic_nuclear)
    .setMessage("Kühlwasserzufuhr unterbrochen!\nWas nun?")
    .setPositiveButton("Abschalten", new OnClickListener() {
        public void onClick(DialogInterface dialog, int which) {
            Toast.makeText(getApplicationContext(),
                "Reaktor wird abgeschaltet...",
                Toast.LENGTH_LONG).show();
        }
    }).setNeutralButton("Weiss nicht", new OnClickListener() {
        public void onClick(DialogInterface dialog, int which) {
            Toast.makeText(getApplicationContext(),
                "Problem an Support weitergeleitet...",
                Toast.LENGTH_SHORT).show();
        }
    }).setNegativeButton("Abwarten", new OnClickListener() {
        public void onClick(DialogInterface dialog, int which) {
            // do nothing
        }
    });
return dialogBuilder.create();
```

## Bsp. 2: Alert-Dialog mit Auswahl-Daten

- Titel, Icon, usw. wie gehabt
- Neu: Daten (Array) setzten
  - Methode `setItems(...)`
  - Inkl. ClickListener
  - Toast mit Wahl anzeigen!

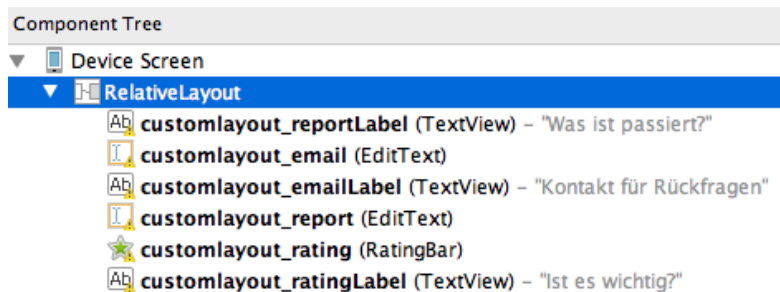


```
final String[] items = {"Kaffee", "Tee", "Wasser"};
AlertDialog.Builder dialogBuilder = new AlertDialog.Builder(this);
dialogBuilder.setTitle("Wähle ein Getränk")
    .setIcon(R.mipmap.ic_launcher)
    .setItems(items, new OnClickListener() {
        public void onClick(DialogInterface dialog, int itemPos) {
```

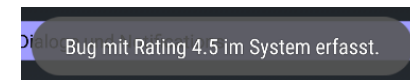
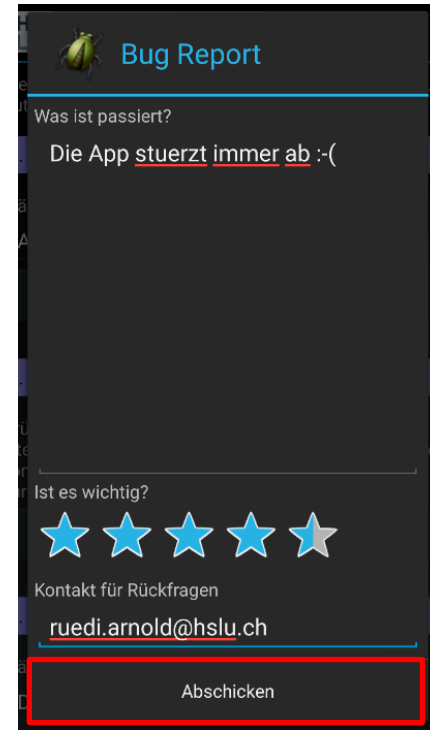
Handler für  
Selection

# Bsp. 3: Alert-Dialog mit eigenem Layout

- Layout.xml „aufblasen“ & setzen

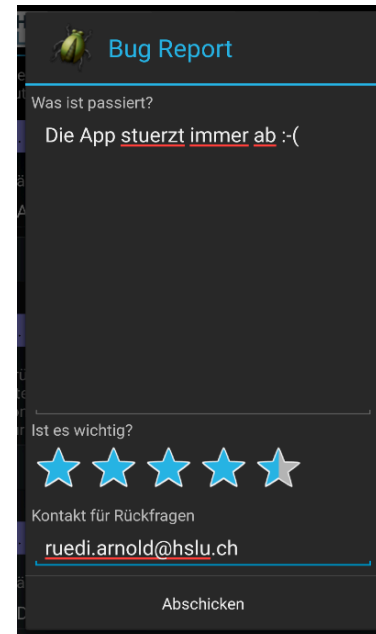
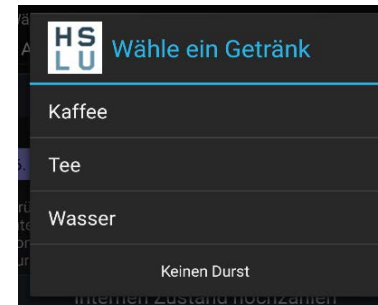
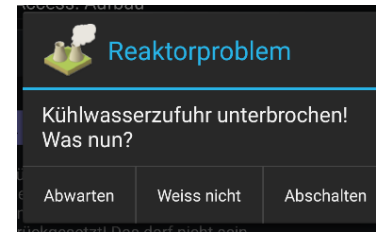


```
private Dialog createCustomLayoutDialog() {  
    final View customView = LayoutInflater.from(this).inflate(  
        R.layout.custom_dialog_layout, null);  
    AlertDialog.Builder dialogBuilder = new AlertDialog.Builder(this);  
    dialogBuilder.setTitle("Bug Report").setIcon(R.drawable.ic_bug)  
        .setView(customView)  
        .setPositiveButton("Abschicken", new OnClickListener() {  
            public void onClick(DialogInterface dialog, int which) {
```



# Demo: Dialoge

- Dialog
  - Ohne Daten
  - Mit Daten (Array)
  - Mit eigenem Layout



# How-To: DialogFragment

- Ein (offener) Dialog gehört zum Zustand einer Activity
  - Falls Konfigurationswechsel wenn Dialog offen, dann wird nicht gespeichert und auch nicht wieder hergestellt!
  - Dialoge sollten deshalb als `DialogFragment` implementiert werden. Zustand des Dialogs wird dann vom `FragmentManager` korrekt mit Lifecycle der Activity synchronisiert (save/restore)
- Mehr dazu unter <https://developer.android.com/guide/topics/ui/dialogs.html#DialogFragment> und später in der Vorlesung (Fragments)

Für den Moment nur soviel: Ein Fragment ist ein wiederverwendbarer «UI Schnippsel» mit eigenem Zustand und Lifecycle

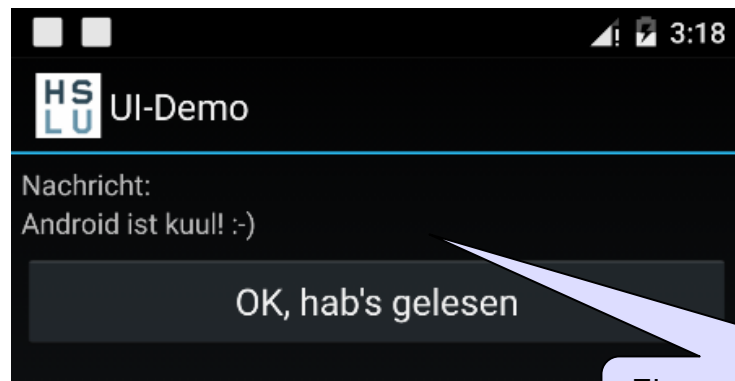
# Notifications (Status-Bar)

- Persistente Nachricht
  - Kurze Ticker-Nachricht in der Status-Bar
  - Danach persistente Anzeige im Notification Window
  - Bei Auswahl erfolgt Aufruf einer definierten Activity
- Vorteile
  - Nachricht bleibt erhalten, bis von Benutzer quittiert
  - Beliebig komplexe Behandlung, da Start einer Activity
- Nachteile
  - Etwas komplexere Mechanik wegen `PendingIntent`



# Demo: Notification

- Code verwendet u.a.:
  - `AlertDialog.Builder`
  - `Notification.Builder`
  - `PendingIntent`
  - `NotificationManager`
  - Eigene dedizierte Activity für Darstellung der Nachricht

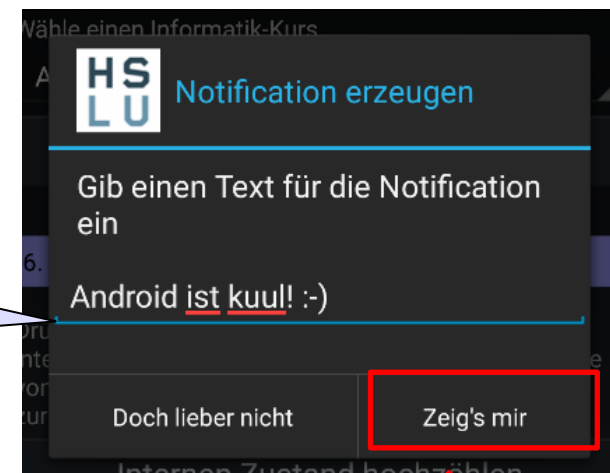


Eigene (!) Activity (inkl. layout.xml), in der die Nachricht angezeigt wird

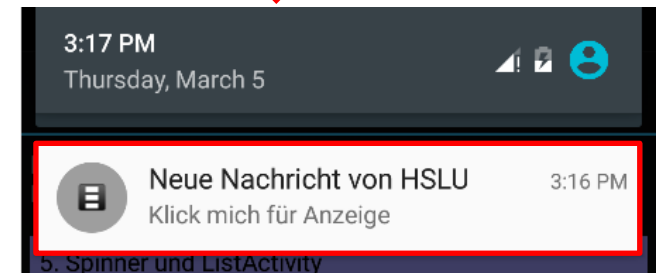
Nicht vergessen: Registrieren im Manifest!..

Icon zeigt neue Notification an

Dialog zum Notification auslösen



Durch „Hinunterziehen“ wird „Notification Window“ angezeigt





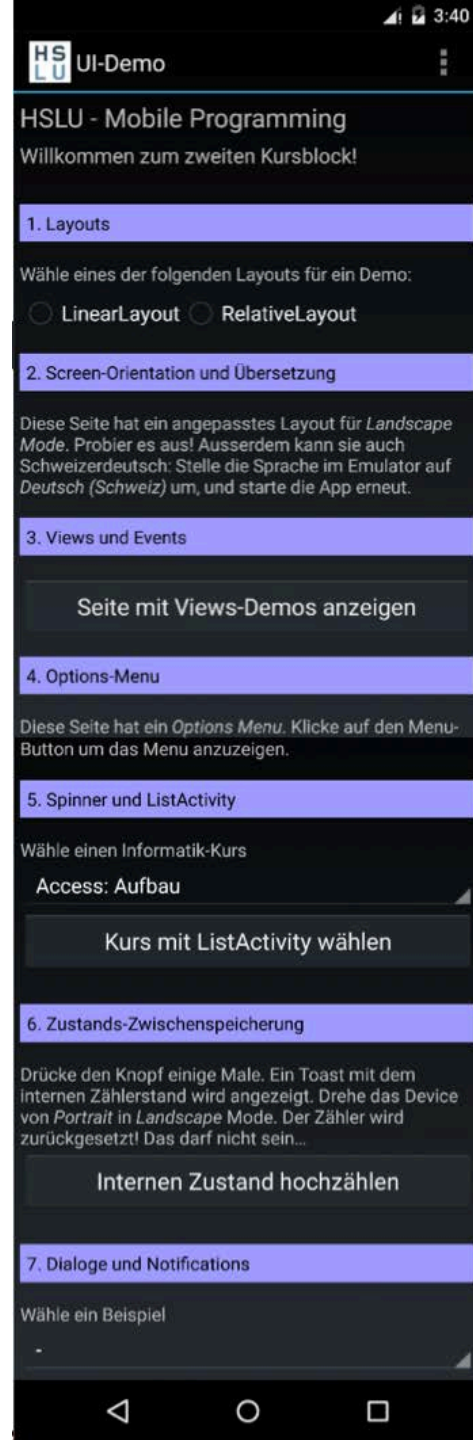
# Zur Übung 2

## Selber Views, Layouts, Events einsetzen

### 7 Teilaufgaben, analog zu Demos

1. Linear- & ConstraintLayout
2. Bildschirm-Orientierung
3. Übersetzung nach Schweizerdeutsch
4. Views & Event-Demo: RatingBar
5. Spinner (& ListActivity)
6. Zustands-Zwischenspeicherung
7. Dialoge (& Notifications)

**Anrechnung für Testat:  
mindestens 4 Teilaufgaben vorzeigen**



# ...den Dozenten/Assistenten Übungen präsentieren?

Wiederholung

- Voraussetzung: Sie haben die Wochenübung im 2er-Team zusammen gelöst und sich auf Vorzeigliste eingetragen
- Ablauf Vorzeigen: Ass./Doz. wählt die zu präsentierenden Aufgaben aus, Studierende zeigen ihre Lösung und beantworten **beide je individuelle** Fragen vom Ass./Doz.
  - Falls Lösungen ok und Fragen von beiden zufriedenstellend beantwortet wird Übung als Teil vom Testat akzeptiert
- Übungspräsentation ist möglich bis max. 1 Woche nach Ausgabe der Übung
  - z.B.: die Übung der SW4 muss bis SW5 präsentiert sein

Heute zeigen Sie Übung 1!

# Kontrolle Übung 1

Reihenfolge gemäss Einträgen auf

<https://tinyurl.com/mobpro-2019-pflicht>

(von unten nach oben)