TU Clausthal

Informatikwerkstatt

Android – Die grafische Nutzerschnittstelle (User Interface)

05.12.2018

Bildquelle: https://developer.android.com/images/brand/Android_Robot_200.png?hl=de

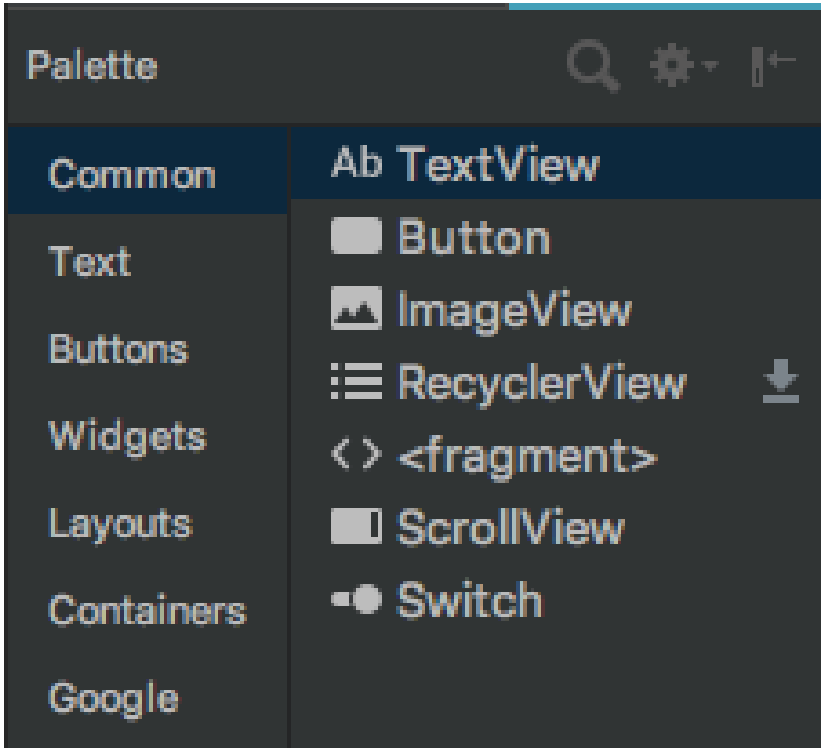
Rico Jimenez
TU-Clausthal

Literaturhinweis

- Folien und Inhalte beruhen teilweise auf
  - https://informatikwerkstatt.github.io/user-interface/#/
  - Informatikwerkstatt — Mobile App Entwicklung mit Android
  - Philipp Kraus — 23. Nov 2018 - 16:21
  - Folien von Dr.-Ing. A. Reinhardt, TU-Clausthal, 2016

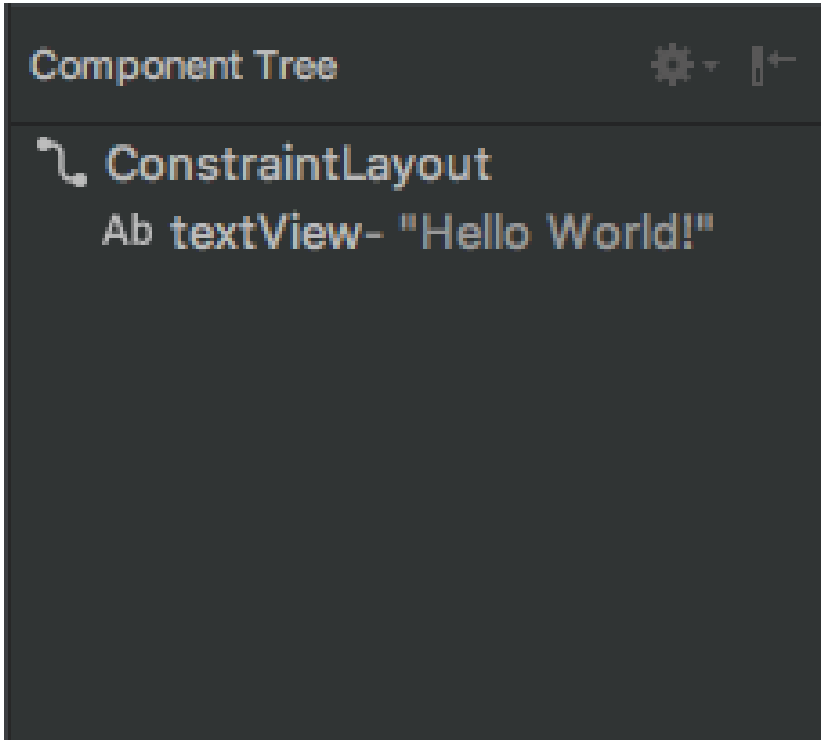
Palette

- Zum Design des Layouts befindet sich auf der linken Seite die *Palette*, die die verschiedenen Elemente enthält
- Man zieht die einzelnen Elemente aus der Palette an die Stelle im Layout, wo sie erscheinen sollen



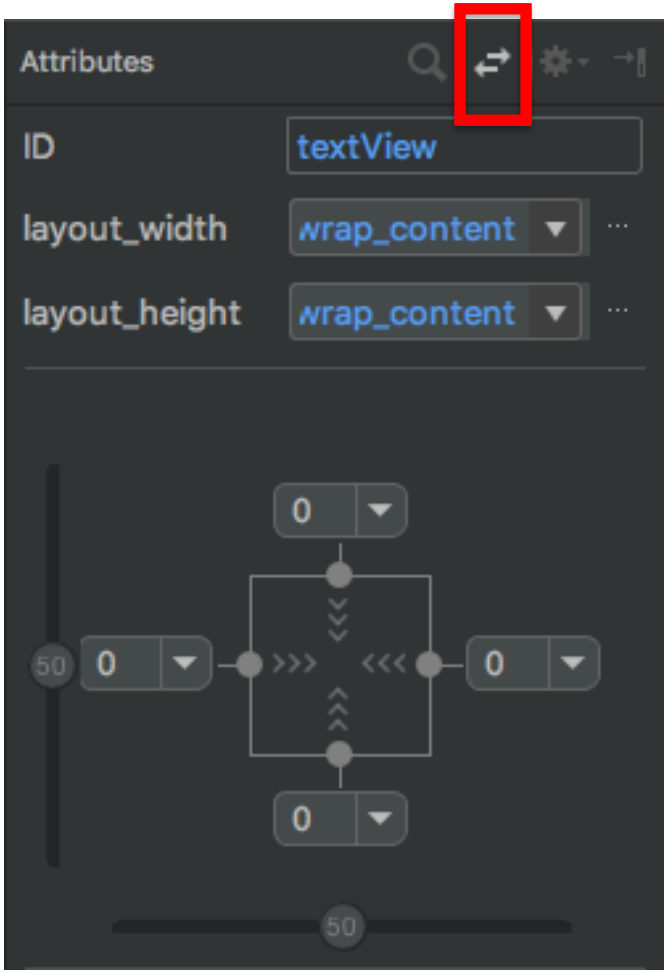
Component Tree

- Der *Component Tree* ist die aktuelle Struktur des Layouts
- Hiermit wird die Hierarchie des Layouts deutlich



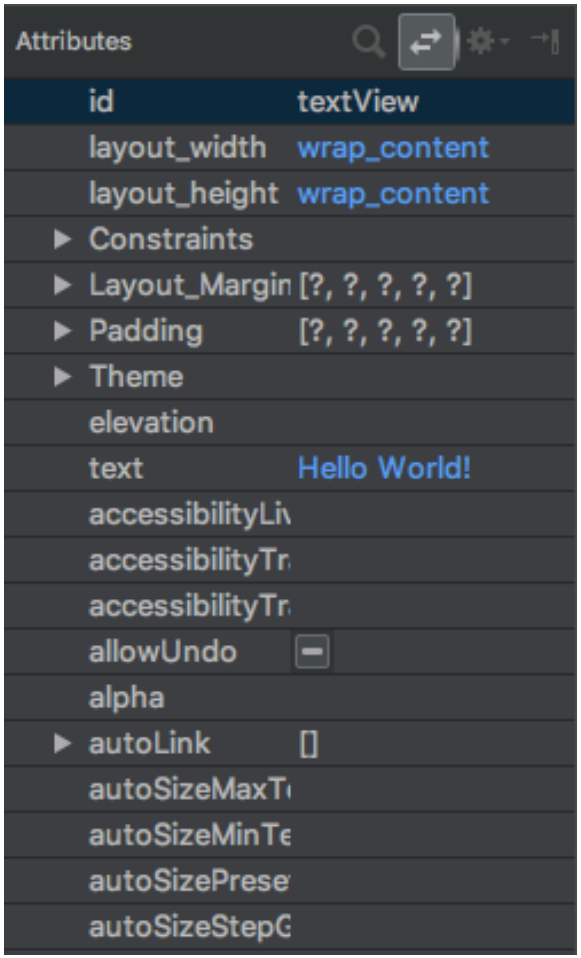
Elementattribute

- Über den Text „View all attributes“ ⇄ kann die Detailsicht zu einem Attribut aufgerufen werden



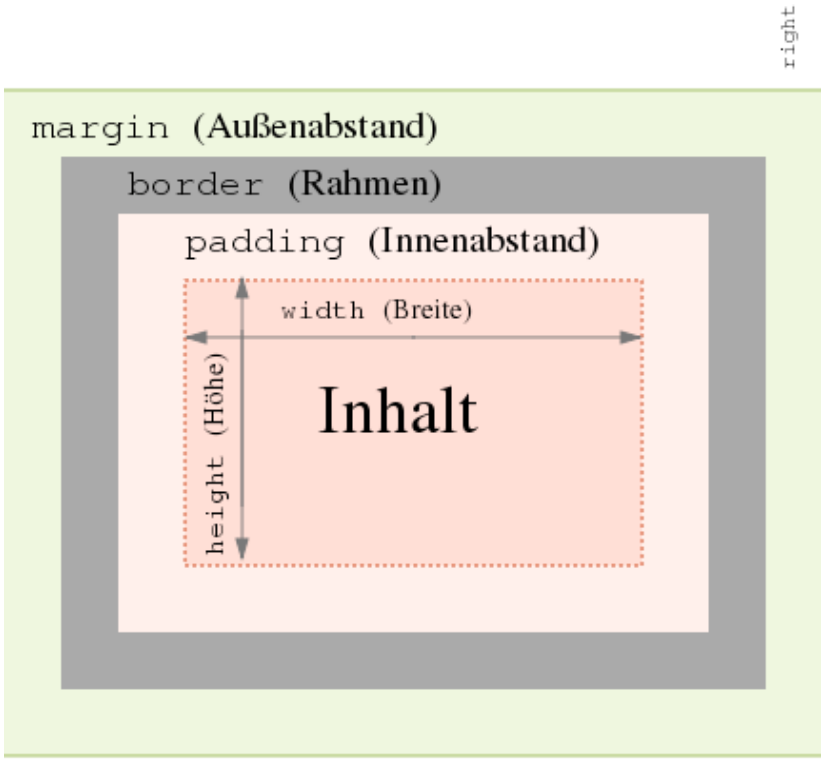
Elementattribute

- Erweiterte Eigenschaften zum Layout
- Abstände, Schrift, Farben, ...



Margin & Padding

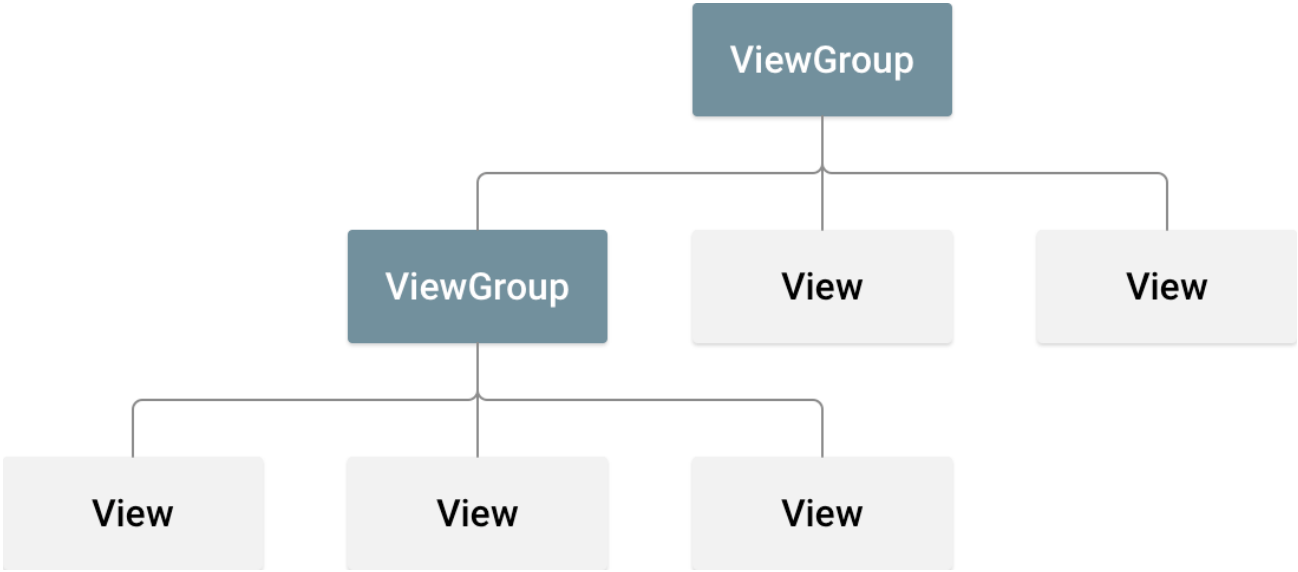
- Ein Element (Inhalt) besitzt eine Box
  - Die Box hat einen Innenabstand (*Padding*)
  - Einen optionalen Rahmen (*Border*)
  - Einen Außenabstand (*Margin*)
- Diese Eigenschaften verhindern das "Zusammenkleben" der Elemente



Bildquelle: https://wiki.selfhtml.org/images/thumb/f/f3/Box-Modell.svg/600px-Box-Modell.svg.png

View und ViewGroup

- Views stellen den grundlegenden Baustein für Komponenten der Benutzeroberfläche
- ViewGroups sind die Basisklasse für Layouts



Bildquelle: https://developer.android.com/images/viewgroup_2x.png

View und ViewGroup

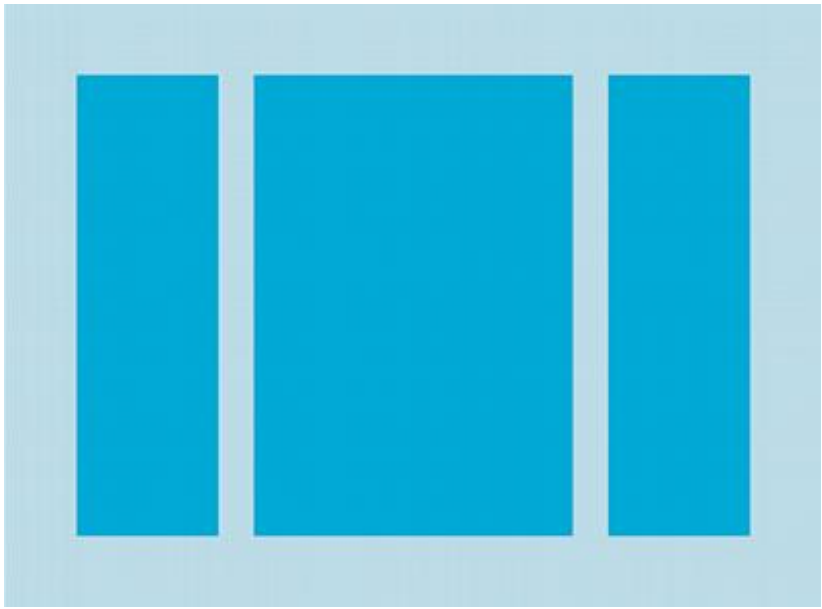
- Ein guter Vergleich ist eine Festplatte unter Windows:
  - Laufwerksbuchstabe bei Windows ≈ Activity
  - Verzeichnis ≈ ViewGroup
  - Datei ≈ View
  - Inhalt ≈ das konkrete Layout des Elements
- Eine ViewGroup ist eine Sammlung von Views und ViewGroups eines Layouts
- Der Aufbau über den Designer erzeugt Eltern – Kind – Verhältnisse

Layouts

- Wir werden uns auf das Erstellen der Layouts beschränken
- Details für komplexere Layouts müssen je nach Aufgabe entwickelt werden
- Ein gutes Design erhält man durch Kreativität und Feedback von Probanden
  - Sprechen Sie mit Ihren Kommilitonen oder weiteren Personen über Ihr Design
- Hilfestellung: Design Styleguides

Linear Layout

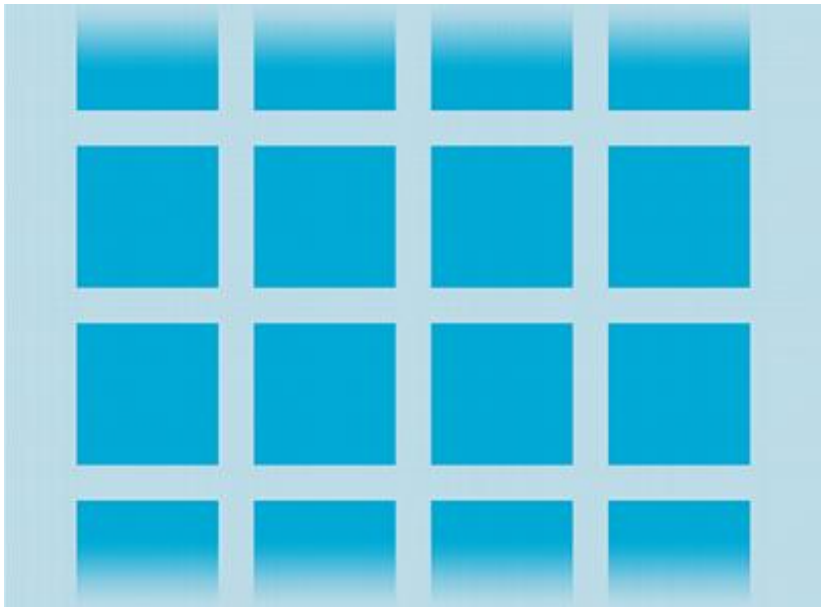
- Das Linear Layout ermöglicht es, geradlinige Strukturen wie z.B. Listen darzustellen
- Die Breite ist flexibel, aber auf eine Spalte begrenzt
- Die Höhe orientiert sich an der Größe des Inhaltes



Bildquelle: https://developer.android.com/images/ui/linearlayout.png

Grid View Layout

- Das Grid View Layout erlaubt den Aufbau schachbrettformiger Strukturen



Bildquelle: https://developer.android.com/images/ui/gridview.png

Binär Taschenrechner - Hilfestellung

- Wie wandele ich eine Binärzahl in eine Dezimalzahl um?

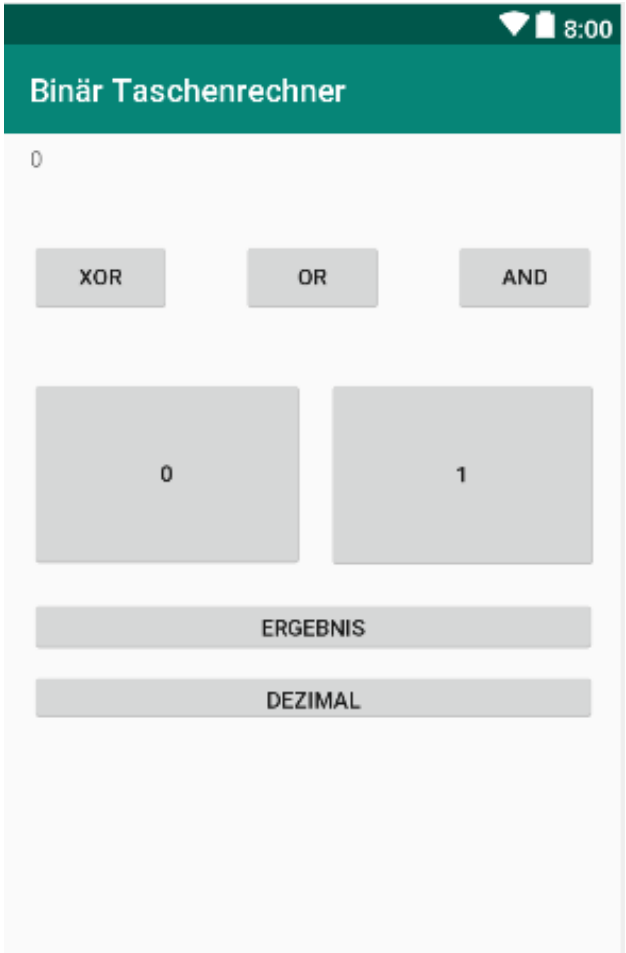
```
int zahl = Integer.parseInt("1111", 2);
```

- Logische Verknüpfungen in Java
  - Sei `a := Integer.parseInt("1000", 2);`
  - Sei `b := Integer.parseInt("0100", 2);`
  - XOR: `int ergebnis = a ^ b;`
  - OR : `int ergebnis = a | b;`
  - AND: `int ergebnis = a & b;`
  - Negation: `int ergebnis = ~a;`

Binär Taschenrechner - Hilfestellung

- Folglich haben wir in *ergebnis* die Dezimalzahl der Operation
- Wir benötigen eine Binärzahl als Ausgabe

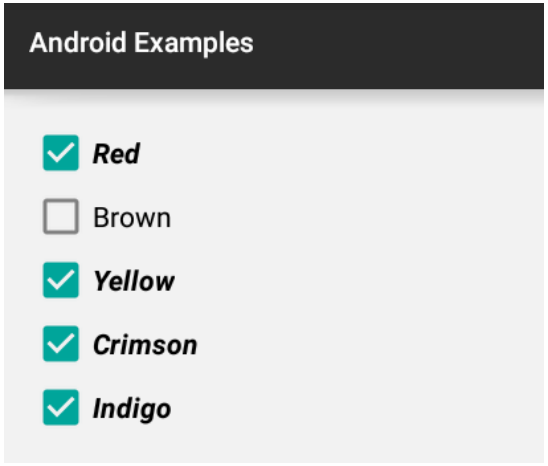
```
String loesung =
Integer.toBinaryString
(ergebnis);
```



@LETS TRY – PFLICHTABGABE 5 ZUM 11.12.2018

1. Es soll ein Binär-Taschenrechner entwickelt werden, mit dem die Operatoren (AND, OR, XOR) durchgeführt werden können.
2. Entwickeln Sie ein sinnvolles und benutzer-freundliches Layout!
3. Es soll die Möglichkeit bestehen, das Ergebnis als Dezimalzahl auszugeben
4. Geben Sie bei fehlerhafter Bedienung (z.B. Operator vor Zahl ausgewählt, zwei Operatoren hintereinander) einen sinnvollen Toast mit einer Fehlermeldung aus

Eingabetypen

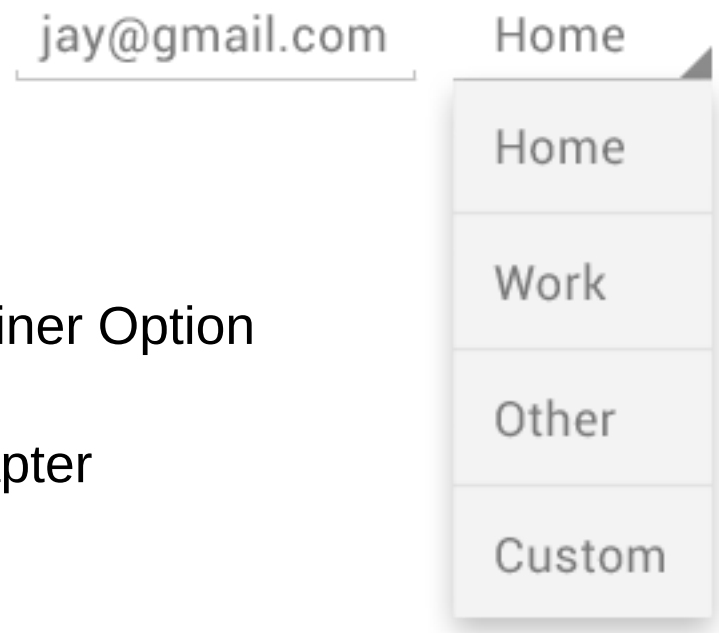


- CheckBox
  - Markierungsfeld (beispielsweise für AGB‘s)
  - `onClick()` – Methode wie beim Button (siehe VL4, Folie 18)
  - Der Zustand wird mittels `.isChecked()` – Methode abgerufen

```
CheckBox box =(CheckBox)findViewById(R.id.checkbox);
boolean checked = box.isChecked();
```

Bildquelle: https://2.bp.blogspot.com/-eNVn0wDWy0I/VR0lwYzem4I/AAAAAAAAWmE/c67purm14Kc/s640/CheckBox2.png

Eingabetypen



- Spinner
  - DropDown-Menü zur Auswahl einer Option
    - z.B Liste von Städten
  - Auswahlmöglichkeiten über Adapter
    - Verbindet Daten mit Elemente

Bildquelle: https://developer.android.com/images/ui/spinner.png

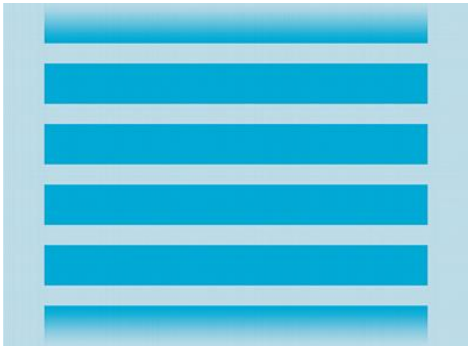
Spinner – Beispiel Implementierung

```
String[] meinArray = { "Berlin", "Hannover",
"Clausthal", "Buxtehude" };

Spinner spinner = (Spinner)findViewById(R.id.spinner);
ArrayAdapter<String> adapter = new
ArrayAdapter<String>(this,
android.R.layout.simple_spinner_item, meinArray);

//Layout des Spinners für den Adapter festlegen
adapter.setDropDownViewResource(android.R.layout.simple_
spinner_dropdown_item);

// Adapter an spinner setzen
spinner.setAdapter(adapter);
```

Adapter

- *AdapterView* - Elemente wie *ListView* und *Spinner* nutzen Adapter, um auf Daten zuzugreifen
- Adapter sind Brücken zwischen Daten und Darstellung
- Sie entscheiden, was in der begrenzten Ansicht angezeigt wird, bauen es und löschen es wieder
  - Um Elemente in einer Ansicht anzupassen, wird der bauende Adapter angepasst
- Adapter können verschiedene Daten und Ansichten nutzen

ListView



Bildquelle: https://developer.android.com/images/ui/listview.png

- Wird durch einen Adapter gefüllt
- Scrollt selbstständig Daten, wenn diese nicht „passen“
- ListView bekommt den Adapter mitgeteilt
- Dem Adapter werden die Daten übermittelt
- Die Daten in einer ListView bezeichnet man als *Item*

ListView – Beispiel Implementierung

```
ArrayList<String> = new
ArrayList<>(Arrays.asList("Null", "Eins", "Zwei",
"Drei", "Vier"));

ListView mListView = findViewById(R.id.listview);

ArrayAdapter<String> adapter = new ArrayAdapter<>(this,
android.R.layout.simple_list_item_1, list);

mListView.setAdapter(adapter);
```

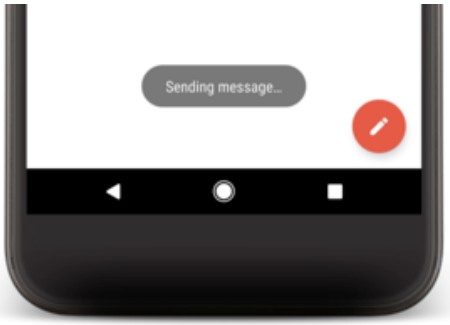
ListView – Beispiel Implementierung

- ListView `onItemClickListener`
  - Registrierbar, ob Elemente angeklickt wurden
- Interface `AdapterView.OnItemClickListener` muss implementiert werden
- Alternativ können kleine Listener hinzugefügt werden
  - Ein Beispiel dazu folgt auf nächster Seite
- Zusätzlich wird ein Parameter „position“ übergeben
  - Dadurch wird erkennbar, welches Item ausgewählt wurde

ListView – Beispiel Implementierung

```
mListView.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View
view, int position, long id) {
      Log.i("TAG", list.get(position));

      Toast.makeText(getApplicationContext(),
      "Position: " + position,
      Toast.LENGTH_LONG).show();
    }
});
```

Toast Notifiers



Bildquelle: https://developer.android.com/images/toast.png

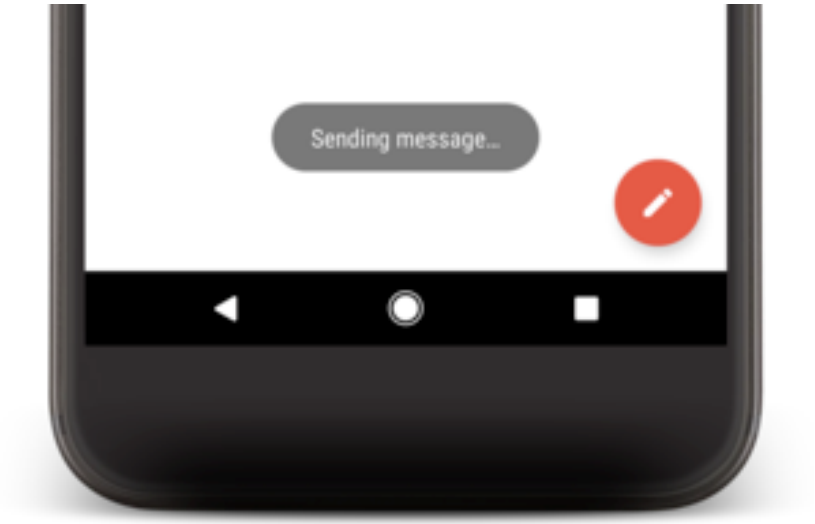
- Toast Notifiers sind kleine Textboxen
- Sie werden in App eingeblendet und nach ein paar Sekunden wieder automatisch entfernt
- Der Aufruf, eine solche Box zu erzeugen, benötigt den *Application-Context*
  - *Hinweis*: Diesen erhält man aus der Activity mit der Methode `getApplicationContext()`
- Dadurch ist es möglich, dass die Nachricht im Vordergrund angezeigt wird
- Anzeigedauer für diese Nachricht kann angegeben werden (siehe Beispiel nächste Seite)

Toast Notifiers: Beispiel

Source Code

```
Toast.makeText(
 activity
 .getApplicationContext(),
  "eine Hallo-Nachricht",
 Toast.LENGTH_LONG |
 Toast.LENGTH_SHORT
).show();
```

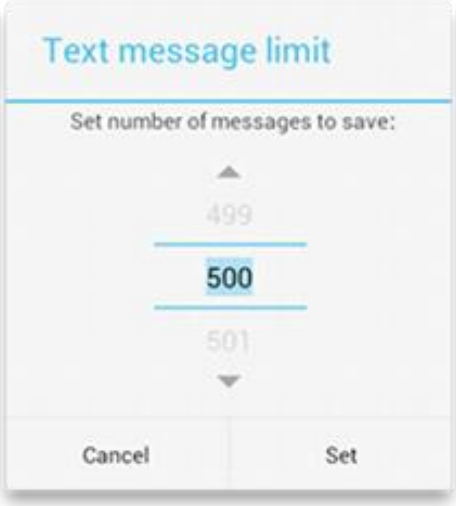
Darstellung

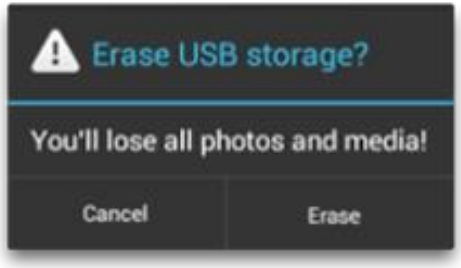


Bildquelle: https://developer.android.com/images/toast.png

Dialog

- Dialoge sind *kleine Fenster* zur Interaktion mit dem Nutzer
- Wichtig bei Dialogen ist die Eigenschaft modal / nicht-modal
- Modal bedeutet blockierend, d.h. so lange der Dialog offen ist, wartet die Anwendung auf die Eingabe





Bildquelle:
https://developer.android.com/images/ui/dialogs.png

XML Darstellung

- Intern wird XML für das Layout verwendet
- Über die Tabs *Design* und *Text* am unteren Rand wechselt man die Ansicht
- Über die XML Struktur können Eigenschaften verändert werden, die nicht über den *Designer* zugänglich sind

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android=
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

@HOME / ÜBUNG: PFLICHTABGABE 6 ZUM 11.12.2018

- Fügen Sie eine CheckBox der „Hello World“ App hinzu
- Verwenden Sie Ihren bestehenden Button dazu, zu überprüfen ob die CheckBox ausgewählt ist
- Geben Sie jeweils einen Toast über den Zustand der CheckBox aus
- Implementieren Sie einen Spinner, überlegen Sie sich dafür passende Elemente

- BONUSAUFGABE
  - Implementieren Sie eine ListView, überlegen Sie sich dafür passende Elemente

Quellen

- https://informatikwerkstatt.github.io/user-interface/
- https://developer.android.com/guide/topics/ui/dialogs
- https://developer.android.com/guide/topics/ui/notifiers/toasts#java
- https://developer.android.com/guide/topics/resources/layout-resource
- https://developer.android.com/training/basics/firstapp/building-ui
- https://developer.android.com/guide/topics/ui/layout/listview