Kapitel 4

Android: Grafische Benutzeroberflächen

Prof. Dr. Carsten Vogt

Technische Hochschule Köln

Fakultät Informations-, Medien- und Elektrotechnik

2019

4. Android: Grafische Benutzeroberflächen

Wichtige Abkürzung: GUI = Graphical User Interface

Grundlegende Webseite:
https://developer.android.com/guide/topics/ui/

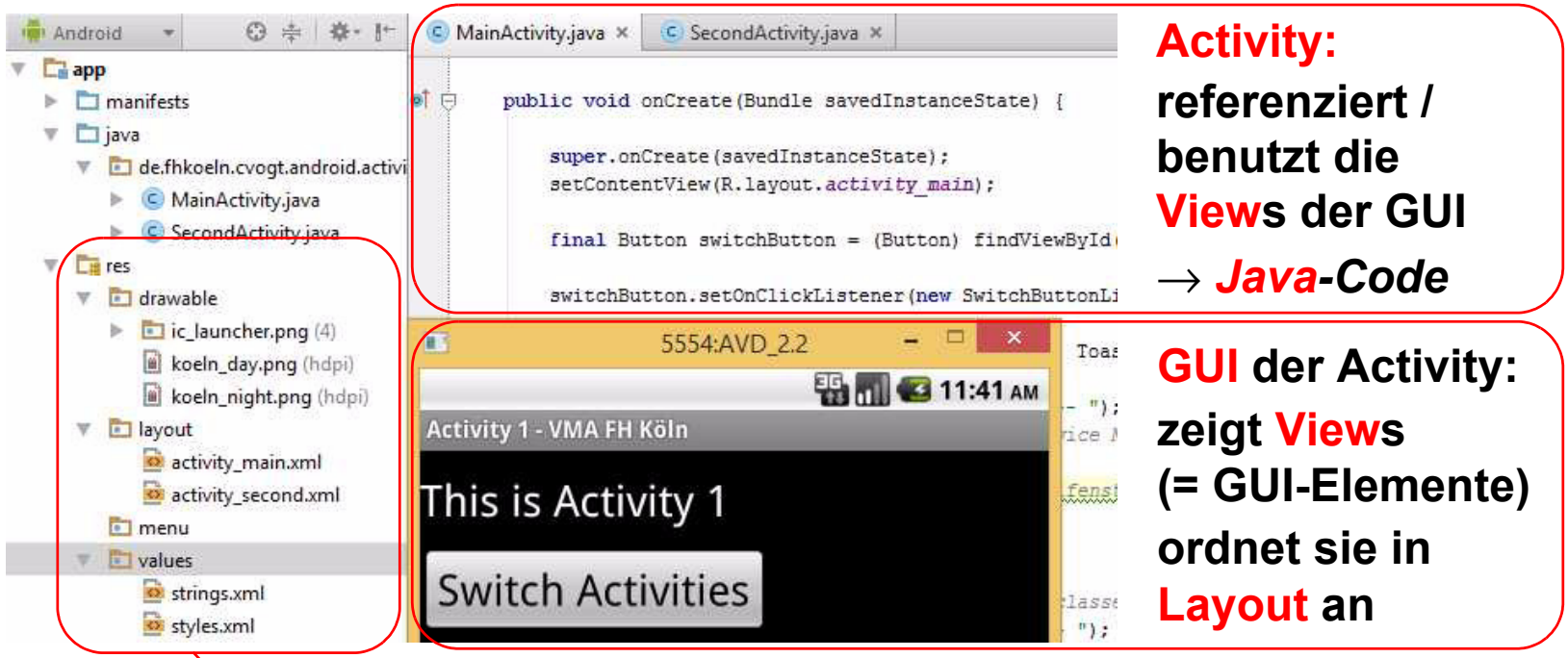
Fragestellungen dieses Kapitels:

- **4.1.) Wie sieht die allgemeine Struktur einer Android-Applikation mit grafischen Oberflächen (GUIs) aus?**
- **4.2.) Aus welchen Basiskomponenten bestehen GUIs und wie ordnet man diese Komponenten in einem Layout an?**
- **4.3./4.4./4.6./4.7.) Welche wichtige GUI-Elemente gibt es im Einzelnen?**
- **4.5.) Wie kann eine Applikation Berührungen und Gesten erkennen und auf sie reagieren?**
- **4.8.) Wie kann man Applikationen an die unterschiedlichen Bildschirmgrößen von Smartphones und Tablets anpassen?**
- **4.9.) Wie kann man eine App programmieren, die den Benutzer durch verschiedene Oberflächen navigieren lässt?**

4. Android: Grafische Benutzeroberflächen

Unterkapitel 4.1. gibt eine einführende Übersicht über die Struktur einer Android-Applikation mit grafischen Oberflächen (GUIs).

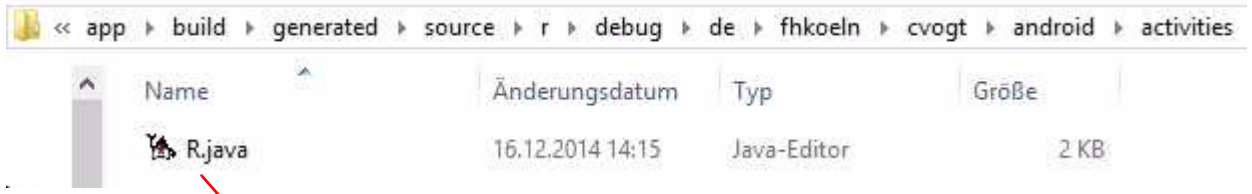
Software-Struktur: Activity, GUI, Ressourcen



Activity: referenziert / benutzt die **Views** der GUI → *Java-Code*

GUI der Activity: zeigt **Views** (= GUI-Elemente) ordnet sie in **Layout** an

Ressourcen: **Layouts** mit **Views**, **Konstanten**, **Bildern**, .. → *XML-Code* (u.a.)



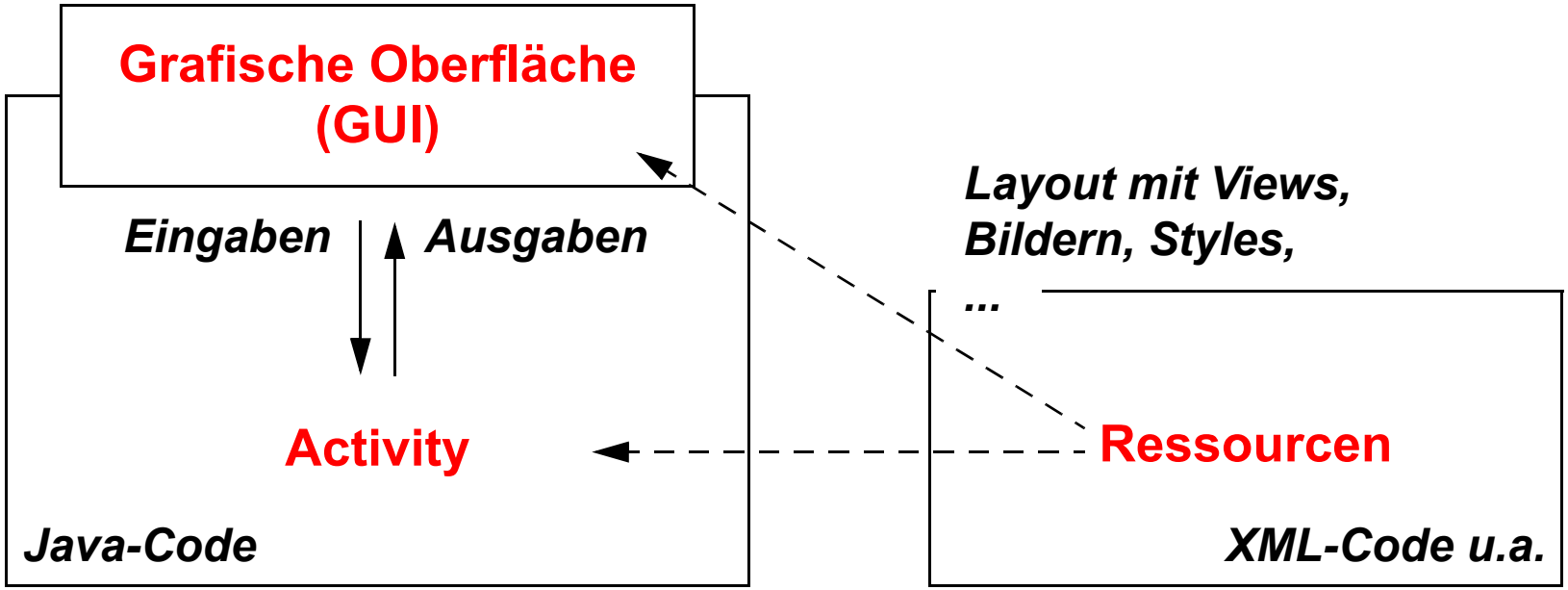
Datei **R.java** → automatisch generierter Java-Code: **Brücke zw. XML und Java**

Die Screenshot-Collage illustriert, was bei der GUI-Programmierung von Bedeutung ist:

- **rechts Mitte: Grafische Anzeigen (GUIs) sind Activities zugeordnet. Die Elemente einer solchen GUI (sogenannte „Views“: Textfelder, Buttons, ...) sind in einem Layout angeordnet.**
- **links: Eine GUI basiert auf Ressouren. Dazu gehören eine XML-Datei, die ihr Layout spezifiziert, und ggf. weitere Dateien.**
- **rechts oben: Activities greifen aus ihrem Java-Code auf die GUI und deren Ressourcen zu. Z.B. wird mit `setContentView()` ein Layout auf dem Display angezeigt.**
- **ganz unten: Eine Datei R.java definiert für jede Ressource eine Ganzzahlkonstante. Mit ihr kann aus Java heraus die Ressource identifiziert werden.**

- *Reproduzieren Sie in Android Studio mit der Beispielapplikation „ActivitiesAndroid“ das, was Sie im oberen Teil der vorherigen Folie sehen.*
- *Versuchen Sie für „ActivitiesAndroid“ herauszufinden,*
  - *wie das Layout der GUIs der beiden Activities definiert wird,*
  - *wie dabei festgelegt wird, welche Bilder angezeigt werden sollen, und*
  - *durch welchen Methodenaufruf die Layouts auf dem Display erscheinen.*

Software-Struktur: Activity, GUI, Ressourcen



Von Interesse:

1.) Definition der GUI

2.) Anzeige der GUI

3.) Kommunikation mit der GUI

Die Grafik skizziert die Organisation und das Zusammenspiel der Applikations-Bestandteile, die für grafische Oberflächen relevant sind:

- links: Eine Activity interagiert über die GUI mit ihren Benutzern. Sie ist in Java programmiert.
- rechts: Die Ressourcen der Applikation sind in XML und anderen, datenspezifischen Formaten codiert. Sie werden an der GUI angezeigt. Der Java-Code der Activity greift auf die Ressourcen zu – z.B. um ein bestimmtes Layout auf das Display zu bringen oder einen Listener (d.h. eine Methode zur Reaktion auf Clicks) an einen Button zu binden.

Die unter 1-3 genannten Punkte werden auf den folgenden Folien näher diskutiert.

1.) Definition der GUI: Views

View: **GUI-Element** in einer Bildschirmseite

- belegt einen **rechteckigen Ausschnitt** des Bildschirms
- definiert die **Grafik-Ausgabe** in diesem Ausschnitt
- reagiert auf **Eingaben / Ereignisse** in diesen Ausschnitt

Arten von Views:

- „**Widgets**“: zur Interaktion mit dem Benutzer
  - Texteingabefelder, Buttons, Checkboxen, ...
- „**ViewGroups**“: zur Organisation von Views
  - Layouts, Listen, Bildergalerien, ...

Klasse `View`: Oberklasse vieler Android-GUI-Klassen

„View“ ist der Sammelbegriff für alle Elemente, die in einer GUI angezeigt werden können (z.B. TextViews für statische Textausgaben oder Buttons).

Die grundlegenden Eigenschaften von Views (Höhe, Breite, Position und vieles mehr) werden in einer Klasse `View` definiert, von der speziellere View-Klassen (`Button`, `EditText`, ...) abgeleitet sind.

Vom programmtechnischen Standpunkt sind Widgets und ViewGroups nicht streng voneinander getrennt. `Widget` ist ein Paket, `ViewGroup` eine Klasse. Es gibt GUI-Elemente, die beides sind (z.B. `LinearLayout`s).

Man kann eigene Views definieren, indem man Unterklassen von `View` schreibt.

> *Verschaffen Sie sich in der Android-Dokumentation*
https://developer.android.com/reference/classes
einen (nur sehr groben) Überblick über die Klasse View und ihre Unterklassen.

1.) Definition der GUI: Layouts und ViewGroups

Ein Layout

- **definiert die Anordnung der Views auf der GUI**
  - **z.B.: lineare vertikale Liste, zweidimensionale Tabelle**
- **ist ein `ViewGroup`-Objekt**
  - **`ViewGroup`: Unterklasse von `View`**
  - **Unterklassen von `ViewGroup` z.B.: `LinearLayout`, `GridLayout`**
- **enthält einfache Views oder weitere ViewGroups**
  - **also: Layout-Schachtelung möglich**
  - **damit: hierarchische GUI-Struktur**
- **wird meist als XML-Ressource definiert**
  - **einschließlich der enthaltenen Views**

Ein Layout wird also durch ein Objekt einer Unterklasse von `ViewGroup` festgelegt.

LinearLayout: lineare Anordnung der Elemente nebeneinander (horizontal) oder untereinander (vertikal).
GridLayout: Anordnung der Elemente in einer zweidimensionalen Struktur.
weitere Layouts: siehe weiter unten.

einfache Views: Buttons, Textfelder, Bilder, ...

Layout-Schachtelung: z.B. ein vertikales lineares Layout, das mehrere horizontale lineare Layouts enthält.

XML-Ressource: im Unterverzeichnis res/layout (siehe Folie 4).

> *Öffnen Sie (in Android Studio mit der Beispielapplikation „ActivitiesAndroid“) die Layout-Datei „activity_main.xml“, indem Sie den Dateinamen im links angezeigten Dateibaum doppelklicken.*
Betrachten Sie den Inhalt der Layout-Datei und schalten Sie zwischen der textuellen und der grafischen Darstellung hin und her („Text“ bzw. „Design“ unterhalb des Teilfensters mit der Dateianzeige anklicken).
Sie können die Layout-Datei in beiden Darstellungsformen editieren – experimentieren Sie hier jeweils ein wenig.

1.) Definition der GUI: Java-Klassen und XML-Elemente

Für jede Art von View (und damit auch ViewGroup):

- **eine Java-Klasse:**
  - **Unterklasse von `View`**
  - **z.B. Java-Klassen `TextView`, `LinearLayout`**
    - **siehe z.B. *https://developer.android.com/reference/android/widget/TextView***
  - **zur Arbeit mit Views in Activities**
- **ein XML-Element**
  - **z.B. XML-Elemente `<TextView>`, `<LinearLayout>`**
    - **siehe z.B. *https://developer.android.com/reference/android/R.styleable.html#TextView***
  - **zur Definition von Views in einer Layout-Datei**

1:1-Beziehungen zwischen Java-Klassen und XML-Elementen

***1:1-Beziehungen*: Für jede Java-Klasse gibt es ein entsprechendes XML-Element und umgekehrt. Meist haben beide denselben Namen.**

Für jedes XML-Element in einer Layout-Datei einer Applikation wird ein entsprechendes Java-Objekt erzeugt.

> *Gehen Sie in der Android-Java-Dokumentation zur Beschreibung einer View-Unterklasse (z.B. TextView). Stellen Sie fest, dass hier sowohl die XML-Attribute als auch die Java-Attribute („Fields”) und -Methoden beschrieben werden.*

1.) Definition der GUI: XML vs. Java

Zwei Möglichkeiten zur GUI-Programmierung:

- **XML-Datei schreiben ...**
  - **mit geschachtelten XML-Elementen**
    → definieren Bestandteile der GUI und ihr Layout
- **... oder Java-Code schreiben**
  - **auf Grundlage der `View`-Unterklassen**
  - **ähnlich Swing**

Empfehlung: XML-Ansatz, da übersichtlicher

- **zusätzlich in Java-Code:**
  dynamische Erzeugung, Änderung und Löschung von GUI-Komponenten

Java-Code schreiben: GUI-Elemente als Java-Objekte erzeugen (z.B. `ll = new LinearLayout(..)`) und ineinander schachteln (z.B. `ll.addView(...)`).

dynamische Erzeugung und Löschung: Ein statisches „Grundlayout“, das durch eine XML-Datei definiert ist, wird dynamisch verändert, indem man zusätzliche GUI-Elemente als Java-Objekte erzeugt (z.B. `tv = new TextView(..)`) und einem vorhandenen Layout hinzufügt (mit `addView(tv)` – siehe oben).

1.) Definition der GUI: Ressourcen

Layout-XML-Dateien sind „Ressourcen“

- **können weitere Ressourcen nutzen**

Ressourcen sind nicht-Java-Bestandteile des Programms

- **Layouts, Styles, Menus, Texte, Bilder, Multimediadaten, ...**
- **auch: String-Konstanten!**

Ressourcen sind in XML- oder Binärdateien gespeichert

- `res/layout`: **Bildschirmseiten mit Layout**
- `res/values`: **konstante Werte, Styles**
- `res/drawable`: **insbes. Bilder**
- `res/menu`: **Menus, Action Bars (App Bars)**
- **und weitere**

Details dazu:
https://developer.android.com/guide/topics/resources/

In res/layout können für verschiedene Klassen von Bildschirmen (z.B. Bildschirme mit unterschiedlichen Größen) unterschiedliche Layouts definiert werden (Details siehe weiter unten).

***konstante Werte*: insbesondere Strings.**

***Styles*: siehe weiter unten.**

***weitere:* res/anim für Animationen, res/color für Farben.**

1.) Definition der GUI: Die Brücke zwischen XML und Java

Ressourcen haben Schlüsselnamen

- **z.B.** `<Button ... android:id="@+id/switchActivity">`
  - **definiert z.B. in** `res/layout/activity_main.xml`

Android Asset Packaging Tool (aapt)
erzeugt daraus den „Schlüsselwertspeicher“:

- **Klasse** `R`**: definiert alle Schlüssel als benannte Konstanten**
  - **in Java-Syntax!**
  - **z.B.**
```
public static final int
                    switchActivity=0x7f050000
```

Activity verschafft sich Ressourcen-Zugriff über die Klasse `R`**:**

- `findViewById(schlüssel)` **liefert Referenz auf** `View`**-Objekt**
- **z.B. oben:** `findViewById(R.id.switchActivity)`
  - **liefert Referenz auf Java-Objekt der Klasse** `Button`

Bedeutung von `@+`:
`@` = der folgende String ist eine ID-Ressource.
`+` = die ID wird hier neu definiert.
Bei IDs, die schon anderweitig definiert wurden, wird das `+` weggelassen. Z.B. bezieht sich `@android:id/empty` auf den vordefinierten *Android Namespace* mit der Klasse `android.R`.

***aapt*: wird automatisch bei der Übersetzung (beim „Build”) einer Applikation aufgerufen.**

- *Betrachten Sie die Layout-Dateien der Beispielapplikation „ActivitiesAndroid“ und stellen Sie fest, dass die dort definierten Buttons mit Namen (IDs) versehen sind.*
  - *Hinweis: Bei den anderen Elementen, die dort definiert sind, wurden keine Namen festgelegt, da auf sie aus dem Java-Code nicht zugegriffen wird.*
- *Suchen Sie die Datei R.java der Beispielapplikation „ActivitiesAndroid“ und öffnen Sie sie mit einem Editor.*
  - *Hinweis: Die Datei befindet sich tief unten im Verzeichnisbaum des Projekts. Um sie zu finden, orientieren Sie sich an der Pfadangabe, die Sie in der unteren Abbildung von Folie 4 sehen.*
  - *Betrachten Sie die Namen der dort definierten Klassen und der Konstanten, die in ihnen enthalten sind. Stellen Sie so fest, dass sie den Verzeichnis- und Dateinamen des Android-Studio-Projekts sowie Namen der dort definierten Elemente (Button, Stringkonstanten, ...) entsprechen.*

- *Betrachten Sie schließlich den Code einer Activity von „ActivitiesAndroid“ und suchen Sie dort nach dem Aufruf von findViewById().*
- *Schauen Sie sich schließlich an, welche weiteren Ressourcen zu „ActivitiesAndroid“ gehören.*

2.) Anzeige der GUI: Bezug zu Activities

Eine GUI ist einer Activity zugeordnet

- **entweder als gesamte Bildschirmseite**
- **oder als „Fragment“**
  - **Teil einer Bildschirmseite**

Activity kann sein:

- **Activity allgemeiner Art**
- **Activity für Spezialzweck**
  - **z.B. `android.app.ListActivity` für Listendarstellung**

Activity zeigt GUI an
durch Zugriff auf entsprechende Layout-Definition

- **z.B. Aufruf in `onCreate()`:**
  `setContentView(R.layout.`*`name_der_layoutdatei`*`)`

Die „GUI einer Activity” ist also ihre Bildschirmanzeige, über die der Benutzer mit der Activity kommuniziert.

Fragments: Auf großen Bildschirmen können mehrere Fragments gleichzeitig angezeigt werden – siehe Unterkapitel 4.8.

ListActivity: siehe Unterkapitel 4.3.

> *Suchen Sie in „ActivitiesAndroid“ nach den beiden Aufrufen von setContentView(). Warum stehen sie ausgerechnet an diesen Stellen?*

3.) Kommunikation mit der GUI:
Eingaben, Ereignisse, Ausgaben

Activity reagiert auf Eingaben in die GUI und GUI-Ereignisse, z.B.:

- Texteingaben
- Clicks
- Gesten

Activity schreibt Ausgaben in die GUI, z.B.:

- Textausgaben
- Grafiken
- Animationen

Ein-/Ausgaben beziehen sich auf Views (oder ViewGroups)

- siehe hierzu auch die `View`-Dokumentation:
  https://developer.android.com/reference/android/view/View

Details zu Ein- und Ausgaben, insbesondere auch zu Gesten, werden weiter unten besprochen.

***Ein-/Ausgaben beziehen sich auf Views*:**
Von einer Eingabe ist immer ein bestimmter View betroffen. Z.B. bezieht sich ein Click auf einen bestimmten Button, nicht auf die GUI als Ganzes, und aktiviert daher den Listener *dieses Buttons*.

3.) Kommunikation mit der GUI: Reaktion auf Ereignisse / Events

GUI-Ereignis führt zu Aufruf einer Methode `onXXX()`

- **„Callback-Methode“: wird durch den Programmierer definiert und durch das Laufzeitsystem aufgerufen**

Definition einer Callback-Methode `onXXX()`:

- **im View selbst**
  - **Methode wird in `View`-Unterklasse ausprogrammiert**
  - **z.B. `View`-Methode `onDraw()`**
    - **wird vom Laufzeitsystem aufgerufen, um den View zu zeichnen**
  - **z.B. `View`-Methode `onTouchEvent()`**
    - **wird bei Berührungsereignis aufgerufen**
- **oder in einem Listener**
  - **Methode wird in `OnXXXListener`-Klasse ausprogrammiert**
  - **z.B. `OnClickListener`-Methode `onClick()`**
    - **wird bei Click auf einen Button aufgerufen**

***durch das Laufzeitsystem aufgerufen*:**
Wenn das Laufzeitsystem auf dem Gerätedisplay ein Berührungsereignis bemerkt, stellt es zunächst fest, welcher View berührt wurde. Es ruft dann eine vom Programmierer geschriebene Callback-Methode auf, die diesem View zugeordnet ist.

Der Programmierer kann selbst Unterklassen von `View` schreiben und dort die Methoden `onDraw()` und `onTouchEvent()` ausprogrammieren (siehe später).

`OnClickListener()` ist ein Interface, das die Methode `onClick()` deklariert. Der Programmierer muss es in einer konkreten Klasse implementieren, also dort `onClick()` ausprogrammieren (siehe dazu die übernächste Folie).

- *Suchen Sie in der Dokumentation der Klasse View nach der Methode onTouchEvent(). Näheres zu dieser Methode lernen Sie in Unterkapitel 4.5.*
  - *Stellen Sie fest, welche weiteren Callback-Methoden für Views es gibt.*
- *Betrachten Sie in der Beispielapplikation „ActivitiesAndroid“ den Code einer Activity. Finden Sie heraus, wie dort der Button-Listener programmiert und beim Button registriert wurde. Stellen Sie dabei auch fest, wie der Listener im Einzelnen auf einen Click reagiert.*

3.) Kommunikation mit der GUI: Listener für Views

Listener = Event Handler =
Objekt mit Callback-Methoden zur Reaktion auf Ereignisse

- **wie in Java SE**

Listener werden ...

- **... definiert: durch Implementation eines Interface**

```
class MyListener implements OnClickListener  {
  public void onClick(View v) {
    Log.v("DEMO","---> Click on Button <--- ");
...
```

- **... erzeugt: durch Konstruktor-Aufruf**

```
MyListener lis = new MyListener()
```

- **... bei den GUI-Elementen registriert**

```
myButton.setOnClickListener(lis)
```

3.) Kommunikation mit der GUI: Listener für Views

Namen von Listener-Interface und -Methode	wird aktiv bei ...
`View.OnClickListener` → `onClick()`	Anklicken
`View.OnLongClickListener` → `onLongClick()`	längerem Anklicken
`View.OnKeyListener` → `onKey()`	Tastaturereignis
`View.OnTouchListener` → `onTouch()`	Touchscreen-Ereignis
`View.OnDragListener()` → `onDrag()`	„Drag“-Ereignis
`... und weitere ...`	

Die Tabelle zeigt einige Interfaces und ihre Methoden, die in der Android-Java-Klasse `View` definiert sind. Ein Programmierer kann diese Interfaces implementieren, d.h. ihre Methoden ausprogrammieren, und entsprechende Objekte bei Views registrieren.

3.) Kommunikation mit der GUI: Ausführung der Listener

Ausführung der Listener: durch den Haupt-Thread der App

- **„UI Thread“, „Main Thread“**

Problem: App muss reaktionsfähig bleiben

- **UI-Tread muss Listener-Ausführung rasch abschließen**
- **Gefahr bei zeitintensiven Listenern:**
  Fehler „ANR“ = „Application Not Responding“

Lösung: dedizierte Threads zur Listener-Ausführung

→ Details siehe Kap. 6.6

Standardmäßig wird eine Applikation durch einen Prozess (Task) mit einem einzigen Thread ausgeführt. Dieser Thread muss insbesondere auf GUI-Ereignisse (z.B. Clicks) reagieren, indem er die entsprechenden Listener und/oder Callback-Methoden ausführt. Er wird daher auch „UI Thread” genannt. Wird der Thread überlastet (z.B. durch zeitintensive Berechnungen in einem Listener / einer Callback-Methode), so kann er auf das nächste Ereignis nicht zeitgerecht reagieren. Die Oberfläche friert ein, und es erscheint schließlich eine ANR-Fehlermeldung. Um dies zu vermeiden, sollte der Programmierer lang andauernde oder gar blockierende Operationen in einen gesonderten Thread auslagern. Details dazu werden in Kap. 6 besprochen.

- *Die Programmierung von Java-Code zum Umgang mit GUI-Elementen, die in einer XML-Datei definiert sind, ist recht ermüdend, da viele Code-Muster immer und immer wieder erstellt werden müssen ("Boilerplate Code"). Unterstützung bieten hier Code-Generatoren, die solchen Code automatisch erzeugen.*

- *Wenn Sie Lust und Zeit haben, können Sie einen dieser Generatoren als Android-Studio-Plugin installieren und im Laufe dieses Kapitels ausprobieren. Nähere Informationen finden Sie unter*
  http://tmorcinek.github.io/
  android-codegenerator-plugin-intellij/

4. Android: Grafische Benutzeroberflächen

Unterkapitel 4.2. zeigt, aus welchen Basiskomponenten GUIs bestehen und wie man sie in einem Layout anordnet.

Detailinformationen zu Views:
https://developer.android.com/reference/android/view/View

Detailinformationen zu Layouts:
https://developer.android.com/guide/topics/ui/declaring-layout

Arten von Views

Widgets: zur Interaktion mit dem Benutzer

- `TextView`, `EditText`: Anzeige und Eingabe von Texten
- `ImageView`: Anzeige von Bildern
- `Button`, `ImageButton`, ...: Schaltflächen
- und viele weitere: siehe folgende Abschnitte

ViewGroups: zur Organisation von Views

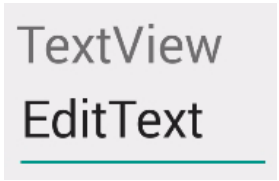
- `LinearLayout`, `GridLayout`, ...: Layouts
- `ScrollView`: scrollbares Feld
- `CalendarView`, `DatePicker`: Datums-Ein-/Ausgabe
- und viele weitere: siehe folgende Abschnitte
  - insbesondere Auswahl- und Ausgabelisten

Bei Views unterscheidet man zwischen *Widgets* und *ViewGroups*.

Button, *ImageButton*: Ein Button zeigt nur einen Text oder ein Icon mit Text an, ein ImageButton zeigt ein Icon oder ein Bild an.

Organisation von Views: Anordnung in ihrer GUI-Darstellung.

Views: Anzeige und Eingabe von Werten



Einfache Views zur Textein-/ausgabe

- Klassen `TextView`, `EditText`
  - Methoden `setText()`, `getText()` etc.
  - `TextView` ist nicht durch Benutzer editierbar
    - also: Textausgabe durch die Applikation
  - `EditText` ist editierbar
    - Texteingabe mit zahlreichen Möglichkeiten: Formatvorgaben, Autocomplete, Eingabelistener, ...
    - auch: Textausgabe
    - Details: *.../training/keyboard-input/style*

Einfache Views zur Anzeige von Bildern:

- Klasse `ImageView`
  - Methode `setImageResource()` etc.

Der Screenshot zeigt oben einen TextView und darunter einen EditText, die mit „TextView" bzw. „EditText" beschriftet sind.

In einen EditText kann der Benutzer Zeichenketten eingeben. Dies wird im Screenshot durch die waagerechte Linie angedeutet.

Die Textausgabe in einem TextView ist im Wesentlichen statisch. Allerdings kann das Programm sie mit `setText()` ändern. `getText()` liefert den aktuell angezeigten Text.

Einem EditText kann ein TextChangedListener zugeordnet werden, der bei einer Eingabe in den EditText aktiv wird.

> *Öffnen Sie die Beispielapplikation „ServicesAndroid“. Ermitteln Sie, wo die angezeigten Textfelder (TextView und EditTexts) definiert werden, und finden Sie die Anweisungen, die die Eingabewerte von den EditTexts einlesen bzw. das Resultat der Berechnung ausgeben.*

> *Stellen Sie fest, wo in der Beispielapplikation „ActivitiesAndroid“ der ImageView zur Anzeige des Bilds definiert wird und woher er das angezeigte Bild bezieht.*

Views: Anzeige und Eingabe von Werten (Forts.)

AdapterViews: „Datenträger-Views“

- Klasse `AdapterView`
  mit Unterklassen `ListView`, `Gallery`, `Spinner` u.a.

- stellen Daten aus Datenbanken, Dateisystem, ... dar
- Details siehe 4.3.

Die hier gezeigten Texte und Bilder sind nicht statisch in einer Layout-Datei o.ä. vorgegeben, sondern werden dynamisch aus einer Datenquelle (z.B. Array, Datenbank, Dateilistings) geholt. Dies ist charakteristisch für *AdapterViews*.

Die Klassen `ListView` (linkes Bild), `Gallery` (Mitte) und `Spinner` (rechts) werden weiter unten im Detail besprochen.

ViewGroups

Abstrakte Oberklasse `android.view.ViewGroup`

ViewGroup dient als „Container" für weitere Views

- **oder auch für ViewGroups → Schachtelung**

Aufgaben von ViewGroups:

- **Zusammenfassung von Views zu größeren Einheiten**
  - **z.B. geschachtelte Auswahlboxen**
- **Festlegung des Layouts**
  - **z.B. Anzeige mehrerer Views linear untereinander:**

```
<LinearLayout ...
  android:orientation="vertical">
  <TextView .../>
  <Button .../>
</LinearLayout>
```

Schachtelung von View-Groups z.B.: Ein vertikales lineares Layout, das mehrere horizontale lineare Layouts enthält.

Der links angegebene XML-Code steht in einer Layout-Datei.

> *Schauen Sie sich in der Beispielapplikation „ServicesAndroid“ an, wie ein geschachteltes Layout definiert wird.*

Layout-ViewGroups

Layout-Unterklassen von `ViewGroup`:

- `FrameLayout`: Anzeige eines einzelnen Views
- `LinearLayout`: horizontale oder vertikale Anordnung
- `GridLayout`: zweidimensionale Anordnung mit Scrolling
- `TableLayout`: Anordnung in Tabelle mit Zeilen und Spalten
- `RelativeLayout`: Anordnung relativ zu anderen Elementen
- `ConstraintLayout`:
  wie `RelativeLayout` + weitere Eigenschaften
  - siehe nächste Folie
- `ListView`: vertikale Liste mit Scrolling
- `Gallery`: horizontale Anordnung von Bildern
- `Spinner`: Drop-Down-Liste
- ...

Diese Folie listet einige Unterklassen von `ViewGroup` auf, mit denen Layouts spezifiziert werden können.

`FrameLayout`: Im Layout können mehrere Views enthalten sein. Einer davon wird im Display links oben angezeigt, die anderen werden von ihm verdeckt.

`RelativeLayout`, `ConstraintLayout`: z.B. Festlegungen der Art „View1 liegt links von View2" oder Festlegung der Größen einzelner Elemente abhängig von der Displaygröße.

Früher gab es noch das „AbsoluteLayout", in dem die View-Positionen durch absolute Angaben festgelegt wurden. Es wird aber nicht mehr unterstützt, da sich damit eine Applikation nicht automatisch an Geräte mit unterschiedlichen Bildschirmgrößen anpassen kann.

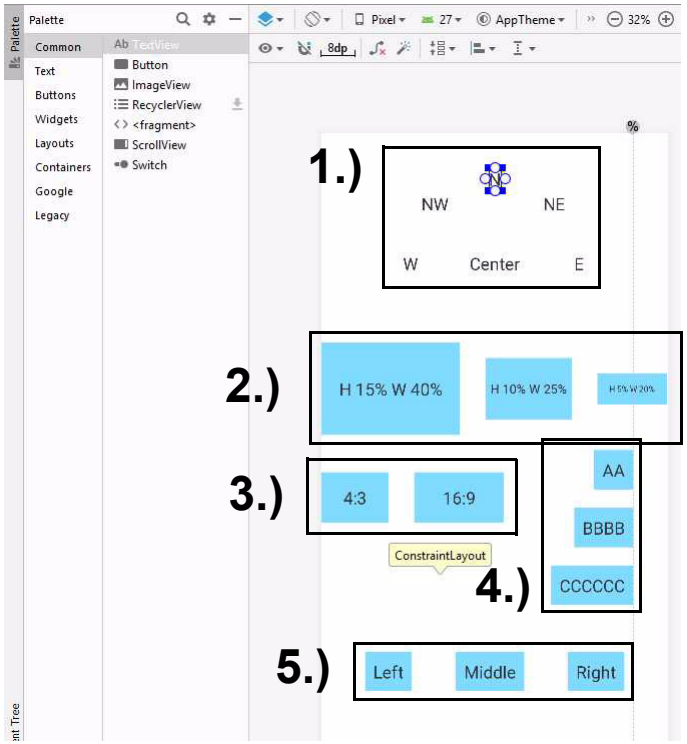
ConstraintLayout

Positionen / Größen von Elementen relativ zu

- **anderen Elementen**
- **dem Gesamtdisplay**

→ automatische Anpassung an unterschiedliche Displays

Erstellung mit grafischem Editor:



zum Beispiel:

1.) Anordnung mit Winkel / Radius

2.) Höhen / Breiten relativ zur Displaygröße

3.) festes Verhältnis Höhe / Breite

4.) Ausrichtung an Guideline

5.) gleichmäßige Verteilung

Mit ConstraintLayouts soll sich die Darstellung der Oberfläche einer Applikation möglichst gut automatisch an die Eigenschaften des Gerätedisplays anpassen. Dies wird dadurch erreicht, dass die Eigenschaften von Widgets (= Oberflächenelementen) relativ zu den Eigenschaften anderer Widgets und des Gesamtdisplays spezifiziert werden – insbesondere ihre Positionen und Größen.

Mit dem grafischen Editor von Android Studio kann man solche Layouts weitgehend per Drag&Drop aufbauen.

Der untere Teil der Folie illustriert einige Möglichkeiten von ConstraintLayouts.

> *Laden Sie die Beispielapplikation „LayoutAndroid“ nach Android Studio, führen Sie sie aus und betrachten Sie das Frame Layout, das Relative Layout, das Constraint Layout, das Grid Layout und das Table Layout (sowohl an der Benutzeroberfläche als auch die entsprechenden XML-Layout-Dateien und deren Anzeige im grafischen Layout-Editor). Experimentieren Sie etwas, indem Sie die Layout-Inhalte ändern. Weitere Informationen zu Layouts finden Sie unter https://developer.android.com/guide/topics/ui/declaring-layout.*
(ListViews und Spinner werden weiter unten näher betrachtet.)

> *Die Activity „JavaDefinedLayout“ (Auswahlpunkt „Linear Layout definiert in Java“) zeigt Ihnen, dass man ein Layout nicht unbedingt durch eine XML-Datei definieren muss, sondern auch im Java-Code selbst festlegen kann.*

> *Wenn Sie Zeit und Lust haben, dann informieren Sie sich unter https://developer.android.com/reference/android/support/constraint/ConstraintLayout*

sowie
https://developer.android.com/training/constraint-layout/
im Detail über die Möglichkeiten des ConstraintLayouts.

> *Betrachten Sie nochmals die Beispielapplikation „ActivitiesUeberlagertAndroid“ mit einem transparenten Layout und einem Layout, das das Display nur teilweise füllt.*

Layout mit Styles und Themes

Layout-Festlegungen für einzelne Views
(z.B. Textgröße, Schriftart, Farbe, ...)
innerhalb der XML-Elemente der Views
(in den Layout-Dateien in `res/layout/`):

- **direkt durch Attribute in den XML-Elementen**
- **durch Referenzen auf Style-Definitionen in `res/values`**

Alternativ: „Themes"-Festlegungen für *alle* Views einer Activity
oder sogar einer ganzen App

Details hierzu:

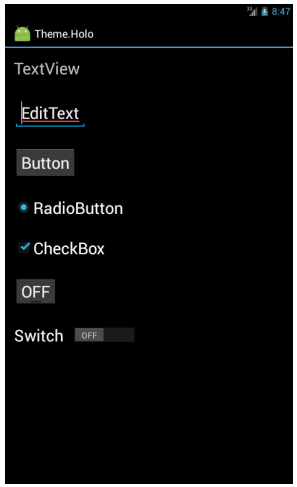
- ***https://developer.android.com***
  /guide/topics/ui/look-and-feel/themes
- ***https://developer.android.com***
  /guide/topics/resources/style-resource
- ***https://developer.android.com/reference/android/R.style***

***durch Attribute in den XML-Elementen*: z.B.**

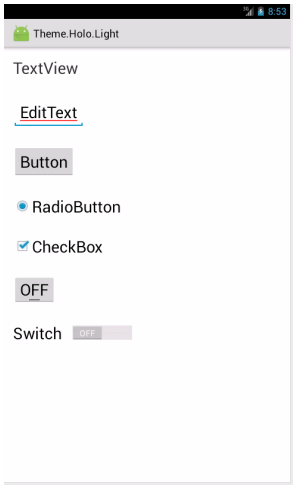
```
<TextView ...
android:textSize
            ="12pt"
...>
```

***durch Referenzen auf Style-Definitionen*:**
siehe Beispiel unten

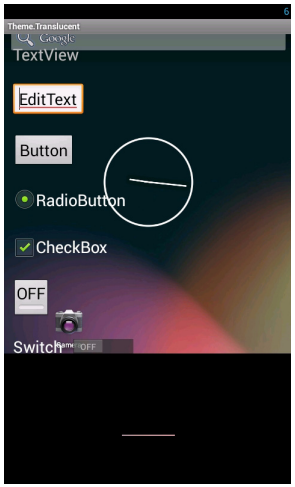
Einige vordefinierte Themes



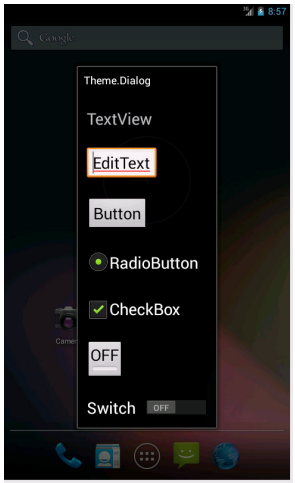
Holo



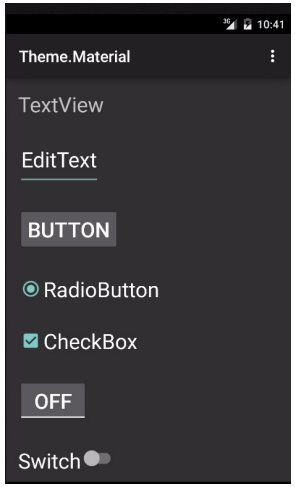
Holo Light



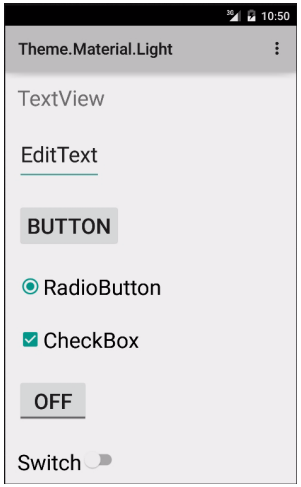
Translucent



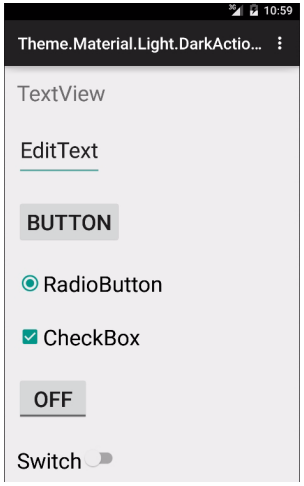
Dialog



Material



Material Light



Material Light Dark Action Bar

Es gibt eine Reihe vordefinierter Styles und Themes, um Applikationen ein einheitliches Erscheinungsbild zu geben – insbesondere das *Holo Theme* seinerzeit in Android 4.0 und das *Material Theme* aktuell seit Android 5.0 in verschiedenen Variationen. Das *Translucent Theme* realisiert transparente Layouts, bei denen das darunter liegende Layout durchscheint. Beim *Dialog Theme* füllt das Layout nur einen Teil des Displays.

Näheres zum Material Design findet man unter *https://developer.android.com/guide/topics/ui/look-and-feel/* und *https://material.io/develop/android/*. Zudem gibt es ausführlichste *Style Guidelines* mit Empfehlungen zur GUI-Gestaltung – siehe *https://developer.android.com/design/*.

- *Öffnen Sie mit Android Studio die Beispielapplikation „StylesThemesAndroid“*
  - *Stellen Sie fest, wie Styles definiert werden und wie man darauf Bezug nimmt.*
  - *Definieren Sie eigene Style-Festlegungen.*
  - *Gehen Sie zur Web-Seite https://developer.android.com/reference/android/R.style und schauen Sie sich an, welche vordefinierten Themes es gibt.*
    Probieren Sie insbesondere THEME_MATERIAL und THEME_MATERIAL_LIGHT praktisch aus.
- *Schauen Sie sich zudem in der Beispielapplikation „LayoutAndroid“ den Unterpunkt „Linear Layout mit Style und Shape“ mit dem zugehörigen Quellcode an.*

Layout für unterschiedliche Displays

Zu berücksichtigen: Unterschiedliche Geräte-Displays

- **unterschiedliche Bildschirmgrößen**
- **unterschiedliche Pixeldichten: „dpi“ = „dots per inch“**

Kategorien:

- **Bildschirmgrößen:**
  - **bis Android 3.1: `small`, `normal`, `large`, `xlarge`**
  - **ab Android 3.2: `sw<zahl>dp`, `w<zahl>dp`, `h<zahl>dp`**
    - **sw = smallest width, w = available width, h = available height**
- **Pixeldichten: `ldpi`, `mdpi`, `hdpi`, `xhdpi`, `xxhdpi`, `xxxhdpi`**
  - **entsprechen ~120 / ~160 / ~240 / ~320 / ~480 / ~640 dpi**

Bildschirmgrößen: Die Bildschirmgrößen von Android-fähigen Geräten differieren mittlerweile gewaltig (Smartphones, Tablets, TV-Geräte).

Pixeldichten: Bei einer hohen Pixeldichte liegen die Bildpunkte des physischen Displays dicht beieinander; die Auflösung ist damit hoch. Die Pixeldichte wird in dpi angegeben.

Bis Android 3.1 wurde nur grob zwischen vier verschiedenen Größenklassen unterschieden. Mittlerweile erlauben Zahlenangaben genauere Spezifikationen (z.B. `sw600dp` = Geräte mit einer Bildschirmbreite von mindestens 600 Pixeln).

Layout für unterschiedliche Displays

Programmierer kann für jede Display-Konfiguration

- **eigene Layouts definieren**
  - **insbes. unterschiedlich für Smartphones und Tablets**
  - **z.B. in den Foldern `res/layout`, `res/layout-sw600dp`, ...**

> Betrachten Sie in der Beispielapplikation „CounterAndroid“ das Unterverzeichnis res/layout und suchen Sie nach den entsprechenden Layout-Dateien.

- **eigene Ressourcen bereitstellen**
  - **insbes. Bitmap-Bilder unterschiedlicher Auflösung**
  - **z.B. in den Foldern `res/drawable-mdpi`, ...**

Wird eine Applikation auf einem Gerät ausgeführt, so stellt das Laufzeitsystem dessen Bildschirmeigenschaften fest und verwendet dann automatisch die passenden Layouts und Ressourcen.

Der Programmierer stellt diese Layouts und Ressourcen in Unterverzeichnissen bereit. Die Namen der Unterverzeichnisse charakterisieren die Display-Konfiguration, für die die Layouts / Ressourcen verwendet werden sollen.

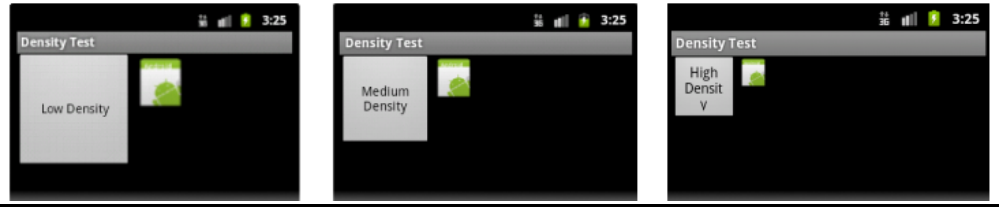
> *Betrachten Sie in der Beispielapplikation „CounterAndroid“ das Unterverzeichnis res/drawable. Sie werden feststellen, dass dort das Applikations-Icon in vier Versionen (d.h. für Bildschirme mit vier unterschiedlichen Auflösungsgraden) definiert ist.*

Layout für unterschiedliche Displays

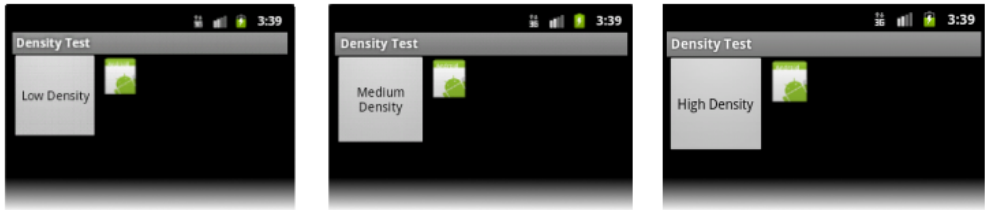
Empfehlung: Größen in „dp" statt „px" festlegen

- **px: absolute Pixelzahl**
- **dp: „density-independent pixel"**
  - **zur Laufzeit: Umrechnung in Pixel des realen Geräts**
  - **somit: auch bei unterschiedlichen Pixeldichten dieselbe absolute Größe**

bei px-Angabe:



bei dp-Angabe:



Quelle: https://developer.android.com

Weitere Details:

https://developer.android.com/guide/practices/screens_support

Gibt man die Länge und Breite eines View in Pixeln an (z.B. 300px breit, 200px hoch), so wird es auf Displays unterschiedlicher Pixeldichten unterschiedlich groß angezeigt – je höher die Pixeldichte, desto kleiner die Darstellung (die oberen drei Screenshots).
Es wird daher empfohlen, Größen in dp zu definieren. dp-Werte werden bei der Darstellung in reale px-Werte umgerechnet, abhängig von der realen Pixeldichte des Displays. Damit wird erreicht, dass der View auf allen Bildschirmen mit derselben Größe angezeigt wird (die unteren drei Screenshots).
Die Umrechnungsformel lautet
px = dp*(dpi/160),
wobei dpi die Pixeldichte des realen Displays angibt.

4. Android: Grafische Benutzeroberflächen

Unterkapitel 4.3. listet Techniken auf, mit denen man Auswahlangebote realisieren kann: Dem Benutzer wird eine Reihe von Wahlmöglichkeiten (z.B. in der Art eines Auswahlmenus) angeboten, und das Programm reagiert dann auf seine Auswahl.

Auswahlangebote – Übersicht

Auswahlen durch den Benutzer insbesondere über:

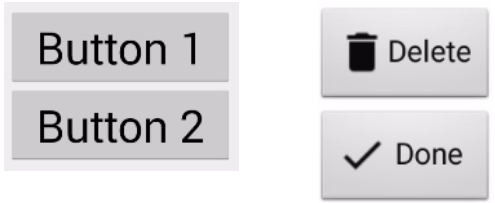
- einfache Buttons, Radiobuttons und Checkboxen
- Seekbars: Schieberegler
- Menus: Auswahllisten
  - Optionsmenu: bei Drücken des Menubuttons des Geräts
    - auf heutigen Geräten: permanente Action Bar (App Bar)
  - Kontextmenu: bei Drücken eines bestimmten Views
  - Popupmenu: aus Programm durch `show()`-Aufruf
- AdapterViews: Auswahlmöglichkeiten mit bestimmtem Layout
  - ListView: vertikal scrollbare Liste
  - Spinner: Dropdown-Liste
  - Gallery: horizontal scrollbare Liste
  - GridView: zweidimensionale Darstellung

Action Bars (auch *App Bars* genannt) wurden eingeführt, als Geräte ohne physischen Menu Button aufkamen. Eine Action Bar wird oben im Display permanent angezeigt und bietet dem Benutzer Auswahlmöglichkeiten. Die dahinterstehende Programmiertechnik ist aber i.w. dieselbe wie bei den traditionellen Optionsmenus.

Während Options- und Kontextmenus durch Drücken des Displays erscheinen (bzw. permanent angezeigt werden), erscheint ein Popupmenu, wenn im Programm die Methode `show()` aufgerufen wird.

Buttons

Button: Schaltfläche zum Anklicken



Reaktion auf Click:

- entweder über `OnClickListener`-Klasse:
  - dort Methode `onClick(View v)` ausprogrammieren
  - Listener-Objekt beim Button registrieren:
    `button.setOnClickListener()`
  - siehe auch 4.2
- oder durch eigenständige Methode:
  - Methode `void beliebigerName(View v)` programmieren
    - z.B. innerhalb der `Activity`-Klasse
  - Methode dem Button zuordnen:
    `android:onClick="beliebigerName"` in der Layout-Datei

→ *Beispielprogramm Android: Buttons*

Für Buttons programmiert man Callback-Methoden. Sie werden aufgerufen, wenn der Benutzer einen Button berührt. Die Methoden können auf zwei verschiedene Arten definiert werden:

- Wie bei Java Swing als Methode einer Listener-Klasse. Objekte dieser Klasse werden bei Buttons registriert.
- Durch eine Standalone-Methode. Solche Methoden werden durch XML-Attribute in Layout-Dateien den Buttons zugeordnet.

In beiden Fällen hat die Methode einen Parameter vom Typ `View`. Über ihn wird eine Referenz auf den View (hier: auf den Button) übergeben, der berührt wurde. Das erlaubt, bei mehreren Buttons dieselbe Methode zu registrieren und bei ihrer Ausführung abzufragen, um welchen Button es sich handelt.

- *Beispielapplikation „SelectionsAndroid“, Activity „ButtonsDemo“:*
  - *Führen Sie die Activity aus, um sich einen Eindruck aus Benutzersicht zu verschaffen (Auswahlpunkt „Buttons“).*
  - *Arbeiten Sie die entsprechenden kommentierten Quellcode-Auszüge durch (Überschrift „Android: Buttons“).*
  - *Schauen Sie sich die Quellcodes in Android Studio an und experimentieren Sie ein wenig damit.*
  - *Stellen Sie dabei insbesondere fest, auf welche zwei Arten man eine Listener-Methode (also eine Methode, die auf einen Click reagiert), an einen Button binden kann.*
  - *Stellen Sie auch fest, wie ein Button mit Icon und Text definiert wird.*

> *Welcher Parameter wird an die Listener-Methode übergeben, und wie wird er ausgewertet und genutzt?*
Auf welchen zwei Wegen stellt das Beispielprogramm fest, welcher Button geklickt wurde?

> *Gehen Sie zu https://material.io/tools/icons/ und betrachten Sie die Icons, die dort zum Download bereitstehen und für Buttons genutzt werden können.*

Buttons

Klassen für Buttons:

- **Klasse `Button`**
  - **Methoden `setText()`, `getText()` etc.**

> Beispielapplikation „SelectionsAndroid“, Activity „ButtonWithImageDemo“: Schauen Sie sich (in Android Studio) an, wie man einen Button mit Bild realisiert und wie sich das Bild bei einem Click auf den Button verändert.

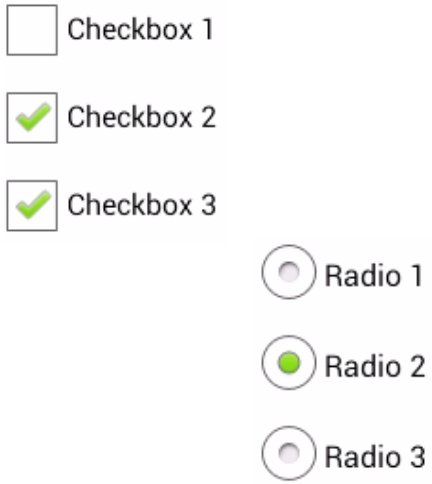
- **Klasse `ImageButton`**
  - **zeigt Bild an**
  - **Methoden `setImageResource()` etc.**

Details zu Buttons:
https://developer.android.com/guide/topics/ui/controls/button

`myButton.getText()` liefert den Text, mit dem der Button `myButton` beschriftet ist. So kann man z.B. in der Listener-Methode über deren `View`-Parameter (siehe vorige Folie) feststellen, welcher Button geklickt wurde. (Alternativ kann man mit `myButton.getId()` die ID des Buttons abfragen und so verschiedene Buttons unterscheiden.)

Statt der Klasse `ImageButton` kann man auch die Klasse `Button` benutzen, um einen Button mit Bild zu realisieren. Das Bild weist man dem Button über das `background`-Attribut zu. Dieses Bild kann sich (wie die Beispielapplikation zeigt) sogar dynamisch ändern, wenn der Button geklickt wird.

Checkboxen und Radiobuttons



Checkboxen:

- Auswahl beliebig vieler Elemente möglich
- Klasse `CheckBox`: für einzelne Checkbox

Radiobuttons in einer Gruppe:

- Auswahl genau eines Elements möglich
- Klasse `RadioButton`: für einzelnen Radiobutton
- Klasse `RadioGroup`: für Gruppe solcher Radiobuttons

Für beide: `OnCheckedChangeListener`

- **Listener für Zustandsänderungen**

→ *Beispielprogramm Android: Checkboxen und Radiobuttons*

Der Listener wird aktiviert, wenn der Benutzer eine Checkbox an- oder abhakt bzw. einen anderen Radiobutton auswählt.

Togglebuttons und Switches

Togglebutton:

- kann „On“ oder „Off“ sein
- zeigt jeweils unterschiedlichen Text an
- Klasse `ToggleButton`

auch hier: `OnCheckedChangeListener`

- **Listener** für Zustandsänderungen

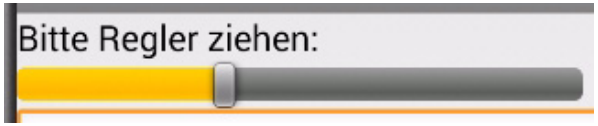
→ ***Beispielprogramm Android: Togglebuttons***

Switch:

- seit Android 4.0 / API-Level 14
- anderes Layout als Togglebutton,
  aber dieselbe Funktionalität
- Klasse `Switch`

- *Beispielapplikation „SelectionsAndroid“, Activity „CheckboxRadioToggleDemo“:*
  - *Führen Sie die Activity aus, um sich einen Eindruck aus Benutzersicht zu verschaffen (Auswahlpunkt „Checkbox/ Radio/Toggle/Switch“).*
  - *Arbeiten Sie die entsprechenden kommentierten Quellcode-Auszüge durch (Überschrift „Android: Checkboxen und Radiobuttons“ sowie „Android: Togglebuttons“).*
  - *Schauen Sie sich die Quellcodes in Android Studio an und experimentieren Sie ein wenig damit.*
  - *Wozu dient der Parameter isChecked der Listener-Methoden?*

Seekbar



Seekbar:

- Schieberegler zur Auswahl von Werten
- Werte sind ganzzahlig:
  - Minimum und Maximum können beliebig festgelegt werden
  - Default-Werte: 0 und 100
- Klasse `SeekBar`

Listener für Änderungen: `OnSeekBarChangeListener`

→ *Beispielprogramm Android: Seekbar*

Das Minimum (= der kleinstmögliche Wert, der gewählt werden kann) ist erst seit Android 8.0 / API-Level 26 frei festsetzbar. In der vorherigen Android-Versionen ist das Minimum stets 0.

Man kann Minimum und Maximum entweder in der XML-Layout-Datei durch die Attribute `android:min` und `android:max` setzen oder im Java-Programmcode durch die Methoden `setMin()` und `setMax()`.

- *Beispielapplikation „SelectionsAndroid“, Activity „SeekbarDemo“:*
  - *Führen Sie die Activity aus, um sich einen Eindruck aus Benutzersicht zu verschaffen (Auswahlpunkt „SeekBar“).*
  - *Arbeiten Sie die entsprechenden kommentierten Quellcode-Auszüge durch (Überschrift „Android: Seekbar“).*
  - *Schauen Sie sich die Quellcodes in Android Studio an und experimentieren Sie ein wenig damit.*
  - *Welche Listener-Methode reagiert auf Benutzereingaben in der Seekbar und welche Methoden gibt es noch?*
  - *Wie wird der Maximalwert einer Seekbar festgelegt?*

Menus

Drei Arten von Menus:

- **Optionsmenus / Action Bar (App Bar)**
  - werden über die Menutaste aktiviert
  - oder erscheinen permanent oben im Display
  - pro Activity / Bildschirmseite: *ein* Optionsmenu
- **Kontextmenus**:
  - werden durch Drücken von Views aktiviert
  - für verschiedene Views unterschiedliche Menus möglich
- **Popupmenus**:
  - seit Android 3.0 / API-Level 11
  - werden durch Aufruf von `show()` aktiviert
  - erscheinen nahe einem bestimmten View
  - für verschiedene Views unterschiedliche Menus möglich

Permanente Action Bars (App Bars) wurden eingeführt, als Geräte ohne Menu Button aufkamen.

Optionsmenus und Kontextmenus werden durch eine Benutzeraktion sichtbar, d.h. nach Drücken des Menu Buttons bzw. eines Views, oder sind permanent sichtbar. Ein Popupmenu wird dagegen durch eine Aktion des Programms, nämlich den Aufruf der Methode `show()`, angezeigt.

Während für jede Activity nur ein einziges Optionsmenu definiert werden kann, kann man für die einzelnen angezeigten Views unterschiedliche Kontext- und Popupmenus spezifizieren.

> *Führen Sie die Beispielapplikation „MenusAndroid“ aus, um sich einen Eindruck aus Benutzersicht zu verschaffen. Tun Sie dies nicht nur mit einem Emulator, sondern möglichst auch mit einem realen Gerät.*

> *Sie sehen zunächst die Ausgabe „Alpha“. Wechseln Sie zur Ausgabe „Beta“, indem Sie den entsprechenden Menu-Auswahlpunkt anklicken (über den Menubutton des Geräts oder die Action Bar oben im Display).*

> *Wechseln Sie von „Beta“ zur Ausgabe „Gamma“. Pressen Sie dort auf die Grafik mit dem griechischen Buchstaben, bis ein Menu erscheint, und wählen Sie dann einen der Menu-Punkte.*

> *Kehren Sie zu „Gamma“ zurück, drücken Sie den Button „PopupMenu zeigen“ und wählen Sie wieder einen Menu-Punkt.*

Definition von Menus

Menus sind Ressourcen:

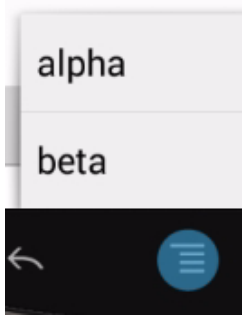
- **XML-Dateien in `res/menu`**
- **pro Menu-Punkt: ein `<item>`-Element**
  -
```
<menu ... >
    <item android:id="@+id/item01"
          android:title="My item 01" />
    <item android:id="@+id/item02"
          android:title="My item 02" />
    ...
</menu>
```

→ ***Beispielprogramm Android: Menus***

> Schauen Sie sich (im kommentierten Quellcode-Auszug und in Android Studio) an, wo und wie der Aufbau der Menus mit ihren Items festgelegt wird.

Bei der Definition von Menu-Ressourcen wird nur festgelegt, welche Items das Menu enthalten soll. Es wird hier nicht zwischen Options-, Kontext- und Popup-menus unterschieden. Alle drei Arten von Menus können also auf dieselbe Ressource zugreifen und damit auch dieselbe Liste von Auswahlitems anzeigen (siehe dazu die Beispielapplikation).

Optionsmenus / Action Bar



Erzeugung und Anzeige eines Optionsmenus:

- durch Drücken der **Menutaste** des Geräts
  - oder ständige Anzeige in der **Action Bar**
- beim ersten Anzeigen des Menus (oder bei Start der Activity): Ausführung von `onCreateOptionsMenu()`
  - zur Initialisierung des Menus
- bei jedem (erneuten) Anzeigen des Menus: Ausführung von `onPrepareOptionsMenu()`
  - zur Aktualisierung des Menus

Bei **Auswahl eines Menupunkts** *menuitem*:

- Ausführung von `onOptionsItemSelected(`*`menuitem`*`)`
  - zur Reaktion auf die Benutzerauswahl

Alle Methoden: definiert in der Klasse `Activity`

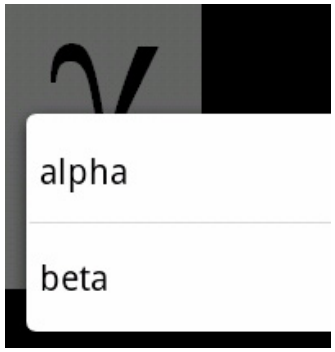
`onCreateOptionsMenu()` füllt das Menu bzw. die Action Bar mit den Items, die in der Menu-Ressource definiert sind. Das Laufzeitsystem ruft diese Methode auf, wenn der Benutzer in dieser Activity den Menubutton erstmals drückt bzw. die Activity gestartet wird und die Action Bar gefüllt werden muss.

In `onPrepareOptionsMenu()`, das bei jeder Anzeige des Menus aufgerufen wird, können die Menu-Einträge aktualisiert werden.

`onOptionsItemSelected()` ist eine Callback-Methode, die aufgerufen wird, wenn der Benutzer im Menu oder der Action Bar eine Auswahl getroffen hat.

Insgesamt sieht man, dass Android für Action Bars kein neues Konzept eingeführt, sondern die für Menus definierten Methoden übernommen hat.

Kontextmenus



Registrierung eines Views für ein Kontextmenu:

- durch `registerForContextMenu(`*`view`*`)`
  - damit Festlegung, dass bei Drücken des Views *`view`* ein Menu erscheinen soll

Anzeige eines Kontextmenus:

- durch „Long Click“ auf einen View
- dabei automatischer Aufruf von `onCreateContextMenu(...,`*`view`*`,...)`
- dort Auswahl eines Menus in Abhängigkeit von *`view`*
- und Anzeige nahe beim View oder als „Contextual Action Bar”

Bei Auswahl eines Menupunkts *menuitem*:

- Ausführung von `onContextItemSelected(`*`menuitem`*`)`

Alle Methoden: definiert in der Klasse `Activity`

Wird ein View gedrückt, so erscheint im Normalfall kein Menu. Wird dies gewünscht, so muss der View explizit durch `registerForContextMenu()` dafür angemeldet werden. Hier wird allerdings noch nicht festgelegt, *welches* Menu das ist!

Wird dann ein so angemeldeter View gedrückt, so wird `onCreateContextMenu()` aufgerufen. Diese Methode ist Activity-spezifisch – sie bezieht sich nicht auf einen bestimmten View, und für eine Activity mit allen ihren Views gibt es nur eine solche Methode. Über ihren View-Parameter kann dann festgestellt werden, welcher View gedrückt wurde, und ein entsprechendes Menu angezeigt werden.

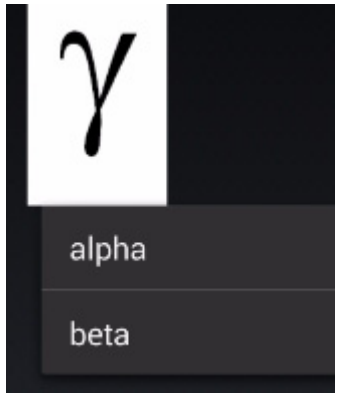
`onContextItemSelected()` ist eine Callback-Methode entsprechend `onOptionsItemSelected()`.

Popupmenus

Seit Android 3.0 / API-Level 11

Ein Popupmenu

- **ist einem View zugeordnet**
  - **wird nahe bei diesem View angezeigt**
- **erscheint bei Aufruf der Menu-Methode `show()`**



Ein Popupmenu wird auf dem Display wie ein Kontextmenu angezeigt. Es erscheint allerdings nicht als Reaktion auf eine Aktion des Benutzers, sondern wenn im Programmcode die Methode `show()` aufgerufen wird.

Zur Vorgehensweise siehe:

https://developer.android.com/guide/topics/ui/menus.html#PopupMenu

und Dokumentation der Klassen:

- `PopupMenu`
- `MenuInflater`
- `OnMenuItemClickListener`

- *Vollziehen Sie anhand der Beispielapplikation „MenusAndroid“ die Erklärungen der vorangehenden Folien nach (kommentierte Quellcode-Auszüge und vollständige Quellcodes in Android Studio).*
- *Warum erscheint in „Gamma“ ein Kontextmenu, wenn Sie die Grafik pressen, aber kein Menu, wenn Sie den TextView oberhalb der Grafik pressen?*
- *Beachten Sie auch (in menu_alpha.xml) den Kommentar zur Verwendung von Icons in der Action Bar. Probieren Sie es aus. Icons finden Sie, wie bereits gesagt, unter https://material.io/icons/*

Webseiten mit weiteren Informationen zu Menus und Action Bar

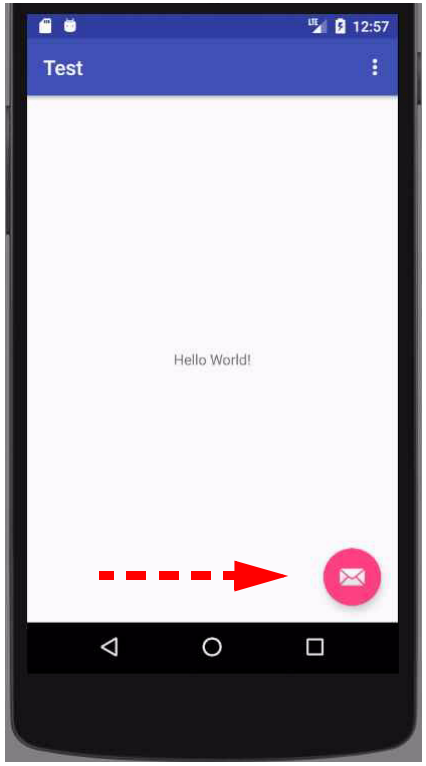
„Menus“:
https://developer.android.com/guide/topics/ui/menus

„Menu Resource“:
https://developer.android.com/guide/topics/resources/menu-resource

„Adding the App Bar“:
https://developer.android.com/training/appbar/

Floating Action Button

- Klasse `FloatingActionButton`
- zur Hervorhebung einer Auswahlmöglichkeit
- Details und Beispiel siehe
  http://www.journaldev.com/10318/android-floating-action-button-example-tutorial



Ergänzend zu den Menus oder anstelle von ihnen kann ein „FloatingActionButton" eingesetzt werden, um eine Auswahlmöglichkeit besonders hervorzuheben.

AdapterViews: Allgemeines Bild

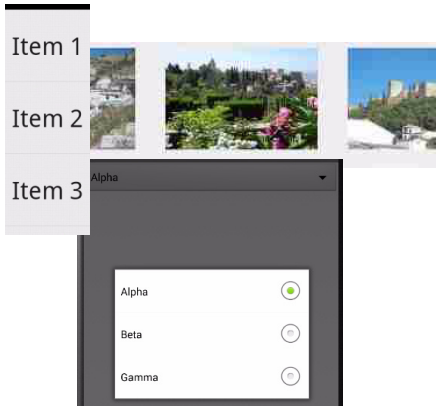
AdapterView
Darstellung von Daten

←

Adapter
Lieferung der Daten

←

Datenquelle
Bereitstellung von Daten



←

Arrays mit Strings

Dateien mit Bildern

Directories mit Dateinamen

...

Ein *AdapterView* ist ein GUI-Element, das die angezeigten Daten über einen Adapter aus einer Datenquelle bezieht. Der Adapter ist also gewissermaßen die Brücke zwischen der Datenquelle und der Anzeige.

Die Daten sind nicht unbedingt statische Ressourcen (wie z.B. String-Konstanten aus `res/values/string.xml`), sondern sie können sich dynamisch ändern (Arrayinhalte, Dateiverzeichnisse, Datenbankeinträge, ...). Details siehe folgende Folien.

AdapterViews: Details

AdapterViews: „Datenträger-Views“

- abstrakte Oberklasse `android.widget.AdapterView`
  - Unterklassen `ListView`, `Spinner`, `Gallery`, `GridView`, ...
- zur Darstellung von Daten aus einer Datenquelle
  - Array, Dateiverzeichnis, Dateiinhalt, Datenbank, ...
  - also nicht aus statischen Ressourcen-Definitionen

Adapter zwischen Datenquelle und AdapterView:

- Interface `android.widget.Adapter`
- füllt AdapterView mit Daten aus der Datenquelle
- kann Anzeige aktualisieren
  - bei Änderungen in der Datenquelle

Diese Folie stellt in Worten (und mit etwas mehr Details) dar, was auf der vorigen Folie in grafischer Form gezeigt wurde.

Details zu den Unterklassen: siehe Folie 68

Ein Beispiel für die Daten, die in einem AdapterView angezeigt werden: Liste der Namen der Dateien, die aktuell in einem bestimmten Verzeichnis stehen.

Ein AdapterView definert die Methode `notifyDataSetChanged()`. Ruft man diese Methode auf, so werden alle Änderungen der Daten in der Datenquelle in die Anzeige übernommen.

AdapterViews: Listener

Item-Listener:

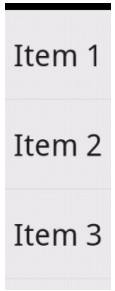
- implementieren vorgegebene Interfaces
- werden bei AdapterViews registriert
- reagieren auf Interaktion des Benutzers mit Items

Namen von Listener-Interface und -Methode	wird aktiv bei ...
`AdapterView.OnItemClickListener` → `onItemClick()`	Anklicken
`AdapterView.OnItemLongClickListener` → `onItemLongClick()`	längerem Drücken
`AdapterView.OnItemSelectedListener` → `onItemSelected()` → `onNothingSelected()`	Auswahl eines Datenelements

Die Klasse `AdapterView` definiert Interfaces für Listener, die auf Drücken von Items reagieren.

AdapterViews: Die wichtigsten `AdapterView`-Unterklassen

`ListView`: vertikale Liste

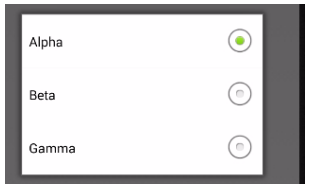


- **statt dessen oft: `ListActivity`**
  - **spezielle Activity zur Nutzung von ListViews**

→ ***Beispielprogramm Android: ListActivity***

→ ***Beispielprogramm Android: Activity mit Resultat (Kap. 6.)***

`Spinner`: Dropdown-Liste

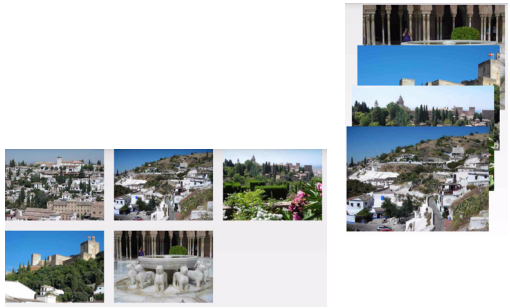


→ ***Beispielprogramm Android: Spinner***

`Gallery`: horizontale Anordnung

→ ***Beispielprogramm Android: Gallery***

`StackView`: schräg hintereinanderliegende Anordnung



`GridView`: zweidimensionale Anordnung

→ ***Beispielprogramm Android: GridView***

In einer `ListActivity` ist insbesondere das Layout vordefiniert (als Liste von Items, die untereinander angezeigt werden). Der Programmierer muss also keine entsprechende Layout-Datei schreiben, und auch der explizite Aufruf von `setContentView()` entfällt.

Eine Alternative zu `ListView`s sind `RecyclerView`s (siehe https://developer.android.com/guide/topics/ui/layout/recyclerview).

Der Name „Spinner" erklärt sich dadurch, dass in frühen Android-Versionen die Auswahl-Items in einem Auswahlkarussell dargestellt wurden. Dieses wurde später durch eine Dropdown-Liste ersetzt.

Die Klasse `Gallery` ist seit API-Level 16 „deprecated", kann aber weiterhin benutzt werden und ist recht bequem in der Nutzung.

> *Beispielapplikation „SelectionsAndroid“, Activities „ListActivityDemo“, „ListViewDemo“, „SpinnerDemo“, „GalleryDemo“, „StackViewDemo“, „GridViewDemo“:*

> *Führen Sie die Activities aus, um sich einen Eindruck aus Benutzersicht zu verschaffen (Auswahlpunkte „ListActivity“, „ListView“, „Spinner“, „Gallery“, „StackView“, „GridView“).*

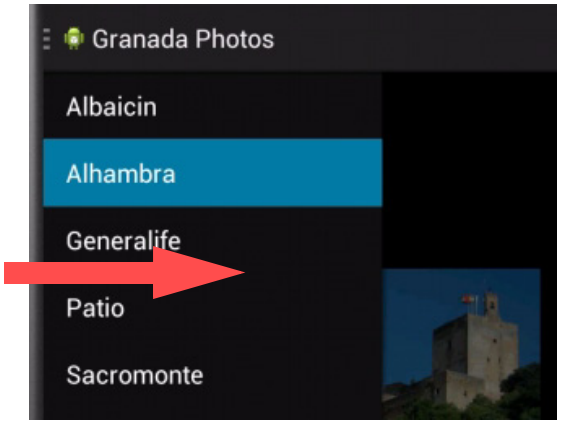
> *Arbeiten Sie die entsprechenden kommentierten Quellcode-Auszüge durch (Überschriften „Android: ListActivity“, „Spinner“, „Gallery“, „GridView“).*

> *Stellen Sie (unter dem Auswahlpunkt „ListView“) insbesondere fest, wie die Anzeige nach Änderungen in der Datenquelle aktualisiert wird.*

> *Vergleichen Sie „ListActivityDemo“ und „ListViewDemo“. Was ist der allgemeine Unterschied? Betrachten Sie dabei die Körper der onCreate()-Methoden: Welchen Aufruf sehen Sie in „ListViewDemo“, der in*

„ListActivityDemo“ fehlt? Warum kann „ListActivityDemo“ darauf verzichten?

ListView und Navigation Drawer



ListViews: auch mit Navigation Drawers

- **Auswahllisten, die man von der Seite in die Oberfläche ziehen kann**

Lehrvideo und weitere Details:

- ***http://www.nt.th-koeln.de/vogt/vma/videos.html***
- ***https://developer.android.com/training/implementing-navigation/nav-drawer***

> Exkurs: Sehen Sie sich das entsprechende Lehrvideo und den zugehörigen Programmcode an.

4. Android: Grafische Benutzeroberflächen

Unterkapitel 4.4. zeigt, wie eine Applikation dem Benutzer Benachrichtigungen und Dialogfenster anzeigen kann.

Benachrichtigungen und Popup-Fenster

Nachrichten an den Benutzer

- **Toast: kurze Nachricht ohne Benutzerreaktion**
  - **erscheint als kleines Fenster ohne Fokus**
- **Notification: Nachricht mit Benutzerreaktion**
  - **erscheint in der Status Bar / Notification Area**
- **Dialog: Nachricht mit Benutzerreaktion**
  - **erscheint als Fenster mit Fokus**
  - **Alert Dialog, Progress Dialog, Date Picker Dialog, Time Picker Dialog**

Darüber hinaus: allgemeine Popup-Fenster

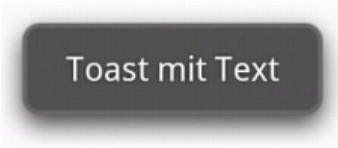
ohne Benutzerreaktion, ohne Fokus:
Der Benutzer kann keine Eingaben machen und auch sonst nicht mit dem Fenster interagieren. Das Fenster verschwindet nach einer bestimmten Zeit von selbst wieder.

Die *Status Bar* mit den Notifications wird oben im Display angezeigt. Der Benutzer kann die hier angezeigten Nachrichten anklicken.

Dialog mit Benutzerreaktion /mit Fokus:
Der Benutzer kann mit dem Fenster interagieren, also Eingaben machen.

Details, insbesondere auch zu Alert-Dialogen etc., folgen auf den nächsten Folien.

Toasts



Klasse `android.widget.Toast`

Kurzzeitige Anzeige einer Information in eigenem Fenster

- **Aufruf aus Activity oder Service heraus**
  - **Activity behält dabei den Fokus**
- **keine Interaktion möglich**

Verwendungsmöglichkeiten:

- **Anzeige einfacher Texte:**
  - `Toast.makeText(myActivity,ausgabetext, Toast.LENGTH_LONG).show();`
- **Anzeige etwas komplexerer Layouts:**
  - **mit** *`myToast`*`.setView(`*`ViewGroup mit Layout`*`)`

→ ***Beispielprogramm Android: Toasts***

keine Interaktion: Der Benutzer kann keine Eingaben machen.

- *Beispielapplikation „NotifDialogsAndroid“, Activity „ToastDemo“:*
  - *Führen Sie die Activity aus, um sich einen Eindruck aus Benutzersicht zu verschaffen (Auswahlpunkt „Toasts“).*
  - *Arbeiten Sie die entsprechenden kommentierten Quellcode-Auszüge durch (Überschrift „Android: Toasts“).*
  - *Schauen Sie sich die Quellcodes in Android Studio an und experimentieren Sie ein wenig damit.*

Notification

Anzeige in Status Bar /
Notification Area (Icon)
und Notifications Window (Expanded Message)

- dynamisch generierbar
- auch aus Service heraus

Benutzer kann Notification anklicken

- dabei Auslösung eines Intents

Java-Klassen im Paket `android.app`:

- `Notification`: definiert Eigenschaften einer Notification
- `NotificationManager`: verwaltet Notifications
  - Methode `notify()`: registriert Notification beim Manager
    → Manager zeigt sie an

→ *Beispielprogramm Android: Status Bar Notifications*

Die Status Bar wird ständig angezeigt. Das Notifications Window kann vom Benutzer nach unten aufgezogen werden.

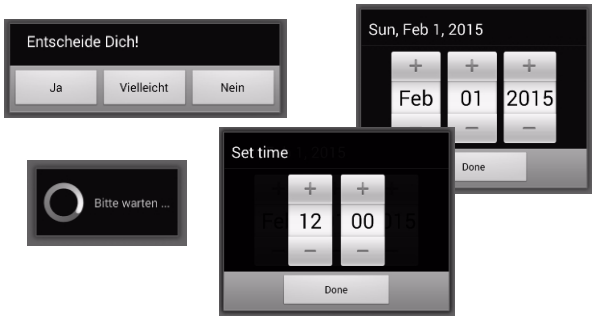
dynamisch generierbar: Notification werden aus dem Java-Code heraus erzeugt und angezeigt

auch aus Service heraus: Services haben keine GUI wie Activities, können also nicht über das volle Display mit dem Benutzer interagieren. Sie können allerdings Benutzer durch Notifications benachrichtigen.

Klickt der Benutzer eine Notification an, so werden dadurch ein Intent und damit Aktionen der Applikation ausgelöst.

- *Beispielapplikation „NotifDialogsAndroid“, Activity „StatusBarDemo“:*
  - *Führen Sie die Activity aus, um sich einen Eindruck aus Benutzersicht zu verschaffen (Auswahlpunkt „Status Bar Notification“).*
  - *Arbeiten Sie die entsprechenden kommentierten Quellcode-Auszüge durch (Überschrift „Android: Status Bar Notifications“).*
  - *Schauen Sie sich die Quellcodes in Android Studio an und experimentieren Sie ein wenig damit.*

Dialog



Anzeige in eigenem Fenster

- Aufruf aus Activity heraus
  - Activity verliert den Fokus
  - damit: Eingaben in das Dialog-Fenster möglich

Basisklasse `android.app.Dialog` mit Unterklassen

- `AlertDialog`: allgemeiner Ein-/Ausgabedialog
  - Titel, Text
  - max. 3 Buttons: positiv, negativ, neutral
  - Radiobuttons, Checkboxen
  - aber auch frei gestaltetes Layout möglich
- `ProgressDialog`: Fortschrittsanzeige
- `DatePickerDialog`: Datumseingabe
- `TimePickerDialog`: Zeiteingabe

Die Screenshots zeigen Beispiele für die vier Arten von Dialogen (Alert Dialog, Progress Dialog, Date und Time Picker).

Activity verliert den Fokus:
Der Fokus geht in das Dialog-Fenster über.

Mit den drei Standard-Buttons können Benutzerreaktionen wie „Ja", „Nein", „Cancel" abgefragt werden. Diese Buttons können angezeigt werden, müssen es aber nicht. Der Programmierer spart sich damit die Arbeit am Layout.

Darüber hinaus kann der Programmierer mit Radio- und Checkbuttons arbeiten, und er kann das Layout seiner Dialoge auch völlig frei gestalten.

Alert Dialog

`Activity`-Methoden:

- `onCreateDialog()`: **Erzeugung** eines Dialogs
- `onPrepareDialog()`: **Aktualisierung** eines Dialogs
- `showDialog()`: **Anzeige** eines Dialogs
- `dismissDialog()`: **Löschen der Anzeige** eines Dialogs

`showDialog(`*`dialogID,Bundle`*`)`:

- ruft `onCreateDialog(`*`dialogID,Bundle`*`)` auf
  - nur beim ersten Mal; Dialog-Objekt wird dann gespeichert
  - Bundle zur Übergabe von Parametern
- ruft `onPrepareDialog(`*`dialogID,Dialog,Bundle`*`)` auf
  - bei jeder (Neu-)Anzeige des Dialogs
  - z.B. zur Aktualisierung übergebener Werte
- zeigt Dialog an

Die Klasse `Activity` sieht vier Methoden für Alert-Dialoge vor:

- Der Programmierer programmiert die Call-back-Methode `onCreateDialog()` aus, um das Aussehen von Dialog-Fenstern festzulegen. Das Laufzeitsystem ruft die Methode auf, wenn ein Dialog erzeugt werden soll. Dabei wird eine ganzzahlige ID für den zu zeigenden Dialog übergeben. Zusätzlich kann man `onPrepareDialog()` ausprogrammieren.
- `showDialog()` ist eine vorgegebene Methode, wird also nicht vom Programmierer erstellt. Sie kann mit einer ID aufgerufen werden und ruft ihrerseits mit dieser ID eine oder beide der obigen Methoden auf, wodurch der Dialog erscheint.
- `dismissDialog()` lässt einen Dialog verschwinden.

Alert Dialog (Forts.)

`onCreateDialog(dialogID,Bundle)`:

- Programmierer legt hier das Aussehen der Dialoge fest
  - in `switch-case`-Anweisung, abhängig von `dialogID`
- Vorgehensweise:
  - Erzeugung eines `AlertDialog.Builder`-Objekts `builder`
  - Festlegung der Dialog-Eigenschaften
    - z.B. `builder.setMessage()`: Beschriftung des Dialogs
    - z.B. `builder.setNegative/Positive/NeutralButton()`: Beschriftung und Listener der Buttons
  - Erzeugung des Dialogs: `builder.create()`
  - Rückgabe des Dialogs per `return`

→ *Beispielprogramm Android: Alert Dialog*

Die typische Vorgehensweise von `onCreateDialog()` wird aus der Beispielapplikation klar (siehe nächste Folie). Zentral ist hier eine `switch-case`-Anweisung. Sie enthält für jeden Dialog der Activity (identifiziert durch eine ganzzahlige `dialogID`) einen `case`-Zweig, in dem sie das Aussehen des Dialogs festlegt.

- *Beispielapplikation „NotifDialogsAndroid“, Activity „AlertDemo“:*
  - *Führen Sie die Activity aus, um sich einen Eindruck aus Benutzersicht zu verschaffen (Auswahlpunkt „Alert Dialog“).*
  - *Arbeiten Sie die entsprechenden kommentierten Quellcode-Auszüge durch (Überschrift „Android: Alert Dialog“).*
  - *Schauen Sie sich die Quellcodes in Android Studio an und experimentieren Sie ein wenig damit.*
  - *Wo wird das Aussehen eines Dialogs definiert?*
  - *Wodurch wird ermöglicht, dass verschiedene Dialoge definiert und angezeigt werden können?*

Alert Dialog: Unterklassen

Progress Dialog: Fortschrittsanzeige

- Klasse `android.app.ProgressDialog`

  → *Beispielprogramm Android: Progress Dialog*

DatePicker Dialog: Eingabe eines Datums

- Klasse `android.app.DatePickerDialog`

TimePicker Dialog: Eingabe einer Uhrzeit

- Klasse `android.app.TimePickerDialog`

  → *Beispielprogramm Android: Date and Time Picker Dialog*

Zur Klasse `AlertDialog` sind die hier genannten Unterklassen vordefiniert, die man für spezielle Aufgaben einsetzt.

- *Beispielapplikation „NotifDialogsAndroid“, Activities „ProgressDemo“ und „DateTimePickerDemo“:*
  - *Führen Sie die Activities aus, um sich einen Eindruck aus Benutzersicht zu verschaffen (Auswahlpunkt „Progress Dialog“ und „Date and Time Picker Dialog“).*
  - *Arbeiten Sie die entsprechenden kommentierten Quellcode-Auszüge durch (Überschriften „Android: Progress Dialog“ und „Android: Date and Time Picker Dialog“).*
  - *Schauen Sie sich die Quellcodes in Android Studio an.*

PopupWindow

Temporäres Fenster
oberhalb der aktuellen Oberfläche

- **Klasse `android.widget.PopupWindow`**
- **zur Anzeige beliebiger Views**
  - **z.B. Layout-ViewGroup als Container für mehrere Views**

Methoden u.a.:

- **`setContentView()`: legt den anzuzeigenden View fest**
- **`showAtLocation()`: zeigt das Fenster an**
- **`update()`: aktualisiert die Anzeige**
- **`setFocusable()`:**
  erlaubt, den Fokus auf das Fenster zu setzen
- **`dismiss()`: löscht das Fenster**

***PopupWindows* kann man alternativ zu Alert-Dialogs einsetzen, um frei gestaltete temporäre Fenster anzuzeigen.**

PopupWindow

OnDismissListener:

- Klasse `PopupWindow.OnDismissListener`
- Activity kann Listener bei PopupWindow registrieren
  - Beim Löschen des Fensters:
    Aufruf der Listener-Methode `onDismiss()`
  - Activity kann dort z.B.
    eine Eingabe aus dem Fenster übernehmen

→ *Beispielprogramm Android: Popup Window*

Ein *OnDismissListener*, der bei einem PopupWindow registriert ist, wird aktiv, wenn das Fenster gelöscht wird. In diesem Listener kann der Programmierer die Aktionen ausprogrammieren, die dann geschehen sollen.

Für das PopupWindow, das im Screenshot auf der vorigen Folie gezeigt wird, kann beispielsweise der eingegebene Name in einer Variablen der Activity gespeichert oder dort dem Benutzer angezeigt werden (siehe die nächste Beispielapplikation).

- *Beispielapplikation „NotifDialogsAndroid“, Activity „PopupWindowDemo“:*
  - *Führen Sie die Activity aus, um sich einen Eindruck aus Benutzersicht zu verschaffen (Auswahlpunkt „PopupWindow“).*
  - *Arbeiten Sie die entsprechenden kommentierten Quellcode-Auszüge durch (Überschrift „Android: Popup Window“).*
  - *Schauen Sie sich die Quellcodes in Android Studio an und experimentieren Sie ein wenig damit.*

Webseiten mit weiteren Informationen

„Notifications“:
https://developer.android.com/guide/topics/ui/notifiers/notifications

„Dialogs“:
https://developer.android.com/guide/topics/ui/dialogs

Einen Effekt wie mit Dialogen kann man durch Activities erzielen, deren Anzeige nicht das gesamte Display füllen, so dass die darunterliegende Activity sichtbar bleibt – siehe hierzu das Programmbeispiel *ActivitiesUeberlagert-Android* aus Kapitel 3.

4. Android: Grafische Benutzeroberflächen

Unterkapitel 4.5. zeigt, wie eine Applikation Berührungen und Gesten erkennen und wie sie auf sie reagieren kann.

Touchscreen-Ereignisse

Ziel: Aktionen / Gesten des Benutzers erkennen

- Benutzer vollführt die Aktionen / Gesten mit seinen Fingern auf dem Touchscreen
  - oder mit einem Stift

Zu unterscheiden:

- Einzelereignisse:
  - einzelne Berührungen mit einem Finger
- Gesten:
  - Folgen von Berührungen
  - möglicherweise mit mehreren Fingern
    - „single-touch“ vs. „multi-touch“

Der Benutzer interagiert mit einer Applikation, indem er das Display berührt und auf ihm Gesten ausführt. Wie eine Applikation auf Standardberührungen (z.B. Click auf einen Button) reagiert, wurde bereits besprochen. Hier geht es nun um den allgemeinen Ansatz zum Umgang mit Berührungen und Gesten.

single-touch: ein Finger,
multi-touch: mehrere Finger (z.B. Auseinanderziehen zum Zoomen)

- *Starten Sie die Beispielapplikation „TouchGesturesAndroid“ auf einem realen Gerät und gehen Sie die einzelnen Auswahlpunkte durch:*
  - *„Einfache Berührungen“: Berühren Sie das Display und lassen Sie es wieder los – entweder an derselben Stelle oder nach Verschieben des Fingers.*
  - *„Multitouch“: Setzen Sie vier bis fünf Finger nacheinander auf das Display.*
  - *„Einfache Gesten“: Ziehen Sie das Icon mit dem Finger über den Bildschirm. Drücken Sie etwas länger auf das stehende Icon.*
  - *„Skalierung“: Machen Sie eine Skalierungsgeste, d.h. ziehen Sie auf dem Display zwei Finger auseinander und schieben Sie sie wieder zusammen.*

> *„Animation mit Wischgeste“: Machen Sie eine (nicht zu starke) Wischbewegung auf dem Bildschirm. Tippen Sie anschließend kurz auf den Bildschirm und/oder machen Sie eine neue Wischbewegung.*
> *[Dieser Teil der Applikation wird im nachfolgenden Unterkapitel erläutert.]*

MotionEvent-Objekte

Ein Objekt der Klasse `android.view.MotionEvent`:

- **beschreibt eine Berührung des Bildschirms**
  - **genauer: die Berührung eines Views**
- **enthält Daten, die einen „Pointer“ betreffen**
  - **Pointer = „active touch point on the screen“**
    - **i.a. Finger oder Stift**
  - **X-Position, Y-Position, Zeitpunkt, Druckstärke, ...**
    - **abfragbar mit get-Methoden: `getX()`, `getY()`, ...**
    - **Gesamtübersicht: *https://developer.android.com/reference/android/view/MotionEvent***
  - **insbesondere: `getAction()`**
    - **Rückgabewert beschreibt die Art des Ereignisses**
    - **„Art” bezieht sich auf Ablauf einer Geste → nächste Folie**

`MotionEvent` ist die zentrale Klasse bei der Arbeit mit Berührungen und Gesten. Ein `MotionEvent`-Objekt beschreibt ein einzelnes Berührungsereignis und enthält dafür Informationen wie Position und Zeitpunkt des Ereignisses. Die Applikation kann diese Informationen auslesen und auswerten.

Eine Geste (z.B. eine Wischbewegung) entspricht einer Folge von Berührungsereignissen und somit einer Folge von `MotionEvent`-Objekten. Wie solche Gesten erkannt und behandelt werden, wird weiter unten dargestellt; hier geht es zunächst einmal hauptsächlich um den Umgang mit einzelnen Berührungsereignissen.

MotionEvent-Objekte: Arten von Ereignissen

Rückgabewerte der `MotionEvent`-Methode `getAction()`:

- `ACTION_DOWN`: eine Geste hat **begonnen**
  - das Objekt enthält die Anfangsposition dieser Geste
- `ACTION_MOVE`: die Geste wird **fortgeführt**
  - das Objekt enthält die aktuell letzte Position dieser Geste und Zwischenpositionen seit dem letzten `ACTION_DOWN` oder `ACTION_MOVE`
- `ACTION_UP`: die Geste ist **beendet**
  - das Objekt enthält die Endposition dieser Geste und Zwischenpositionen seit dem letzten `ACTION_DOWN` oder `ACTION_MOVE`
- `ACTION_CANCEL`: die Geste wurde **abgebrochen**

Wie schon gesagt, setzt sich eine Geste aus mehreren `MotionEvent`-Objekten zusammen. Das `action`-Attribut eines dieser `MotionEvent`-Objekte beschreibt dabei die Rolle, die die entsprechende Berührung in dieser Geste spielt.

`ACTION_CANCEL`: Beispielsweise, wenn der Finger einen vorgegebenen Bereich verlässt.

Views: onTouchEvent()

Berührungsereignisse beziehen sich auf `View`-Objekte

Ein `View`-Objekt wird über ein Ereignis informiert:

- **durch automatischen Aufruf seiner `View`-Methode `onTouchEvent()`**
  - **also: einer Callback-Methode**
- **Parameter dabei: `MotionEvent`-Objekt**
  - **beschreibt das Ereignis (→ vorherige Folie)**

→ Die Reaktionen eines Views auf ein Ereignis programmiert man in seiner Methode `onTouchEvent()` aus

→ ***Beispielprogramm Android: Einfache Berührungen***

Alternativ: Listener beim View registrieren

- **Interface `View.OnTouchListener` mit Methode `onTouch()`**

Jedes `View`-Objekt besitzt eine Callback-Methode `onTouchEvent()`. Das Laufzeitsystem ruft sie bei einer Berührung des Views auf. Bei den vorgegebenen `View`-Klassen (`Button`, `ListView`, ...) ist die Methode implizit vorhanden.

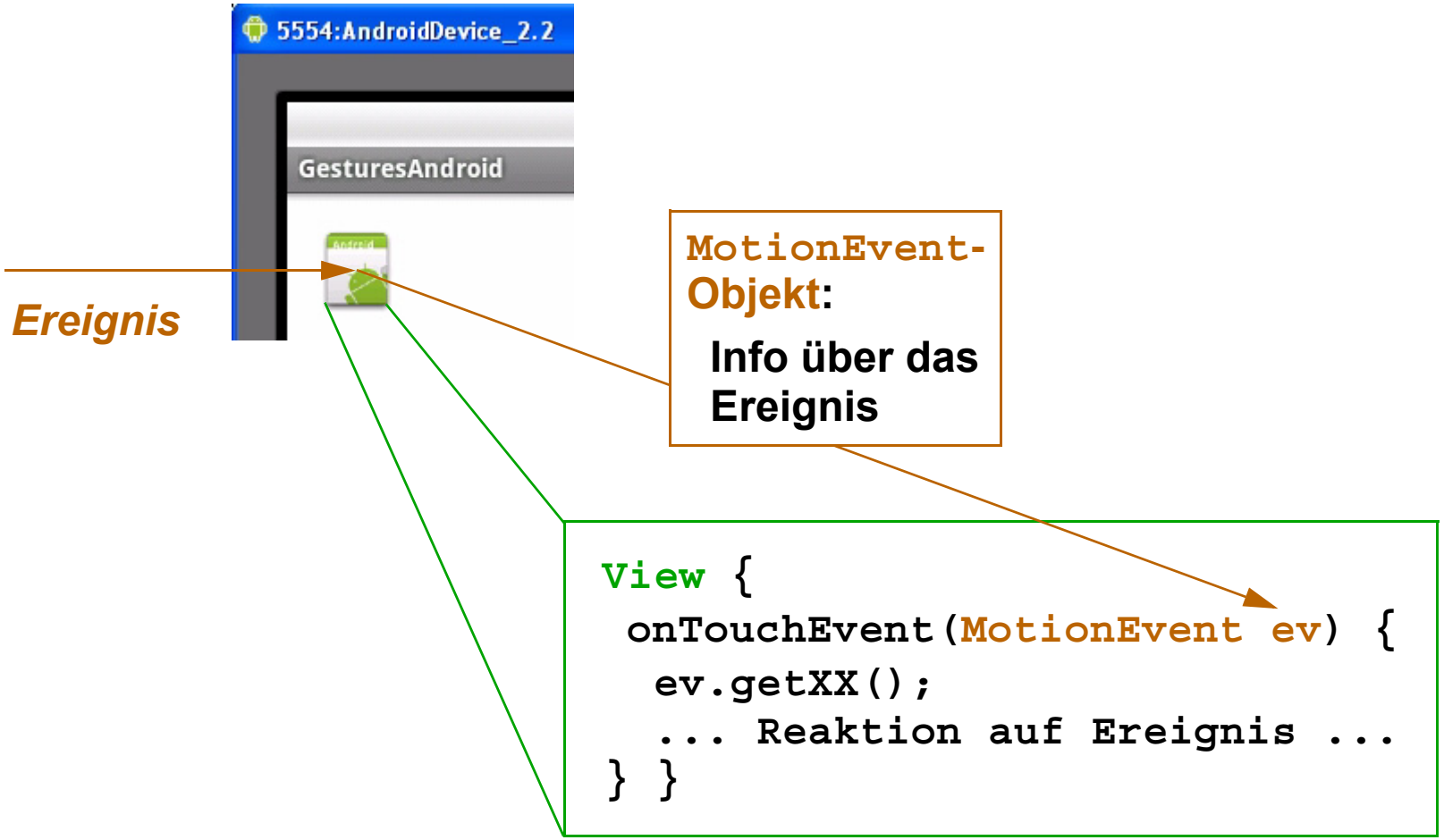
Der Programmierer kann eigene Unterklassen von `View` schreiben und für sie `onTouchEvent()` ausprogrammieren, also explizit festlegen, was bei der Berührung eines solchen Views geschehen soll. In der Beispielapplikation, die auf der nächsten Folie angesprochen wird, wird das so gemacht.

Alternativ dazu kann man eine Klasse schreiben, die das Interface `View.OnTouchListener` mit der Methode `onTouch()` implementiert, und ein Objekt dieser Klasse beim View per `setOnTouchListener()` registrieren.

- *Vollziehen Sie anhand der Beispielapplikation „TouchGesturesAndroid“ die Erklärungen der vorangehenden Folien nach:*
  - *kommentierte Quellcode-Auszüge: Überschrift „Android: Einfache Berührungen“.*
  - *Android Studio: Activity „SingleTouch“.*
  - *Was macht hier die Methode onTouchEvent() und was macht die Methode onDraw()?*
  - *Vollziehen Sie nach, dass sich die Berührungsereignisse auf einzelne Views, also nicht auf das Display als Ganzes beziehen.*

Touchscreen-Ereignisse: Das Gesamtbild

Zusammengefasst: Reaktion auf ein Touchscreen-Ereignis



Das Gesamtbild zeigt den Ablauf bei der Reaktion auf ein Berührungsereignis:

- **Pfeil von links: Der Benutzer berührt auf dem Display einen View (hier z.B. das grün-weiße Icon).**
- **grünes Rechteck: Die Klasse, zu der dieser View gehört, definiert eine Methode `onTouchEvent()` mit einem Parameter der Klasse `MotionEvent`.**
- **Pfeil vom View zum Methoden-Parameter: Das Laufzeitsystem erzeugt ein `MotionEvent`-Objekt, das das Berührungsereignis beschreibt. Es ruft `onTouchEvent()` des betroffenen Views auf und übergibt dabei das `MotionEvent`-Objekt.**
- **`onTouchEvent()` reagiert auf das Ereignis und wertet dafür das `MotionEvent`-Objekt durch `getXX()`-Aufrufe aus.**

Multi-Touch Events

Ereignisse mit mehreren Pointern:

- „Multi-Touch Events“
  - z.B. mehrere Finger auf dem Bildschirm
- entsprechende Komponenten der Klasse `MotionEvent`:
  - zusätzliche -Konstanten `ACTION_XX`:
    - `ACTION_POINTER_DOWN`, `ACTION_POINTER_UP`
      = ein *weiterer* Pointer geht runter / hoch
  - `int`-Parameter bei `getX()` und `getY()`:
    Index des Pointers, dessen Position abgefragt wird

→ ***Beispielprogramm Android:***
Berührungen mit mehreren Pointern

Android kann erkennen, ob ein *weiterer* Pointer auf den Bildschirm gesetzt wird. Im Anschluss daran wird erkannt, welcher dieser Pointer bewegt wurde bzw. das Display wieder verlassen hat. Hiermit kann man Applikationen programmieren, die durch mehrere Finger gesteuert werden.

Ein `MotionEvent`-Objekt enthält Informationen über alle aktiven Pointer. Die Information über einen bestimmten Pointer erhält man dann über dessen Indexnummer.

> *Vollziehen Sie anhand der Beispielapplikation „TouchGesturesAndroid“ die Erklärungen der vorangehenden Folien nach (auf einem realen Gerät, da der Emulator im Normalfall keine Multitouch-Events unterstützt):*

> *kommentierte Quellcode-Auszüge: Überschrift „Android: Berührungen mit mehreren Pointern“.*

> *Android Studio: Activity „Multitouch“.*

Gesten

Beispiele für Gesten:

- **Single-Touch-Gesten:**
  - **Long Press**
  - **Double Tap**
  - **Scrolling**
  - **Swiping / Flinging (= Wischen)**
  - **Dragging**
- **Multi-Touch-Gesten:**
  - **Scaling / Pinch Zooming (= Skalierung mit zwei Fingern)**
  - **(Rotation = Drehbewegung mit zwei Fingern)**

Eine Geste ist, wie schon gesagt, eine Folge von Berührungsereignissen. Daran ist entweder nur ein Pointer / Finger („Single-Touch“) oder mehrere Pointer / Finger beteiligt („Multi-Touch“).

Zum Erkennen einer Skalierungsgeste stellt Android einen entsprechenden Gesture Detector bereit (siehe weiter unten), für Rotationsgesten jedoch nicht (siehe z.B. http://stackoverflow.com/questions/8570900/2-finger-rotation-gesture-listener-in-android).

Gesture Detectors

Eine Geste ist eine Folge mehrerer Events

- dargestellt durch Folge von `MotionEvent`-Objekten

Problem: Aufwendige Programmierung, um aus MotionEvents eine komplexere Geste zu erkennen

Lösung: GestureDetector

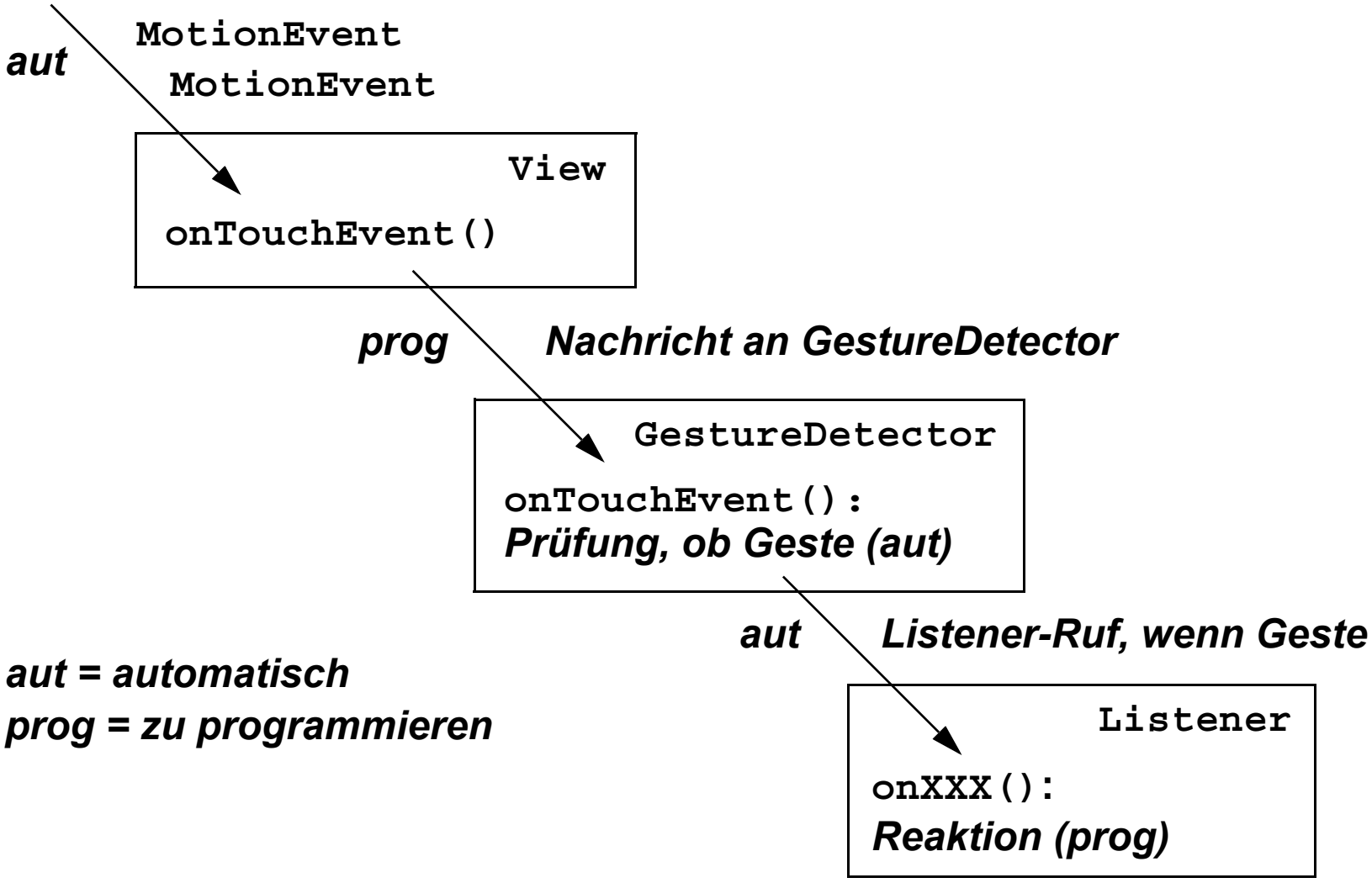
- dient zur automatischen Erkennung komplexerer Gesten
- arbeitet dazu als Filter:
  - wird über alle MotionEvents informiert
    - durch expliziten Aufruf seiner Methode `onTouchEvent()`
  - löst „Higher-Level Gesture Event“ aus, wenn er aus den MotionEvents eine Geste erkannt hat
  - ruft dafür entsprechende Listener-Methode auf
    - Listener muss zuvor beim Detector registriert worden sein

Aufwendige Programmierung, um ... Geste zu erkennen:
Der Programmierer müsste Code schreiben, der die gemeldeten `MotionEvent`-Objekte speichert und sie anhand ihrer zeitlichen und räumlichen Beziehungen zueinander analysiert.

Um dem Programmierer eine solche Arbeit zu ersparen, stellt Android *Gesture Detectors* bereit, die diese Analyse übernehmen. Das Programm meldet einem Detector jeden Event, und der Detector ruft eine Listener-Methode (= Callback-Methode) auf, sobald er aus den gemeldeten Event eine Geste erkannt hat. Dieser Listener-Aufruf ist der „Higher-Level Gesture Event“, der in der Folie genannt ist.

Gesten: Das Gesamtbild

Behandlung von Gesten also:



aut = automatisch

prog = zu programmieren

Die Grafik zeigt die Erkennung von Gesten und die Reaktion darauf:

- Das Laufzeitsystem meldet automatisch jeden Event, indem es `onTouchEvent()` des betroffenen Views aufruft.
- Aus `onTouchEvent()` des Views wird `onTouchEvent()` des GestureDetectors aufgerufen. Der `MotionEvent`-Parameter der `View`-Methode wird dabei weitergereicht. Der Aufruf muss explizit im Körper der `View`-Methode stehen!
- Der Detector prüft automatisch, ob sich aus dem gemeldeten Event zusammen mit den vorherigen eine Geste ergibt.
- Wurde eine Geste erkannt, so wird ein zugehöriger Listener aktiviert. Dieser Listener muss vom Programmierer geschrieben werden.

Vordefinierte Gesture Detectors

`android.view.GestureDetector`:

- für Single-Touch-Events
- zugehörige Listener-Klasse:
  `GestureDetector.SimpleOnGestureListener`
  - implementiert die Interfaces
    `OnGestureListener` und `OnDoubleTapListener`
  - Methoden `onFling()`, `onLongPress()`, `onDoubleTap()`, ...
    - Gesamtübersicht: *https://developer.android.com/reference/android/view/GestureDetector.SimpleOnGestureListener*
  - GestureDetector ruft diese Listener-Methoden auf, wenn er die entsprechenden Gesten erkannt hat

→ *Beispielprogramm Android: Einfache Gesten*

Detektoren der Klasse `GestureDetector` können Gesten wie Wischen, langer Druck und Doppelclick erkennen. Sie rufen dann entsprechende Listener-Methoden der Klasse `SimpleOnGestureListener` auf, sofern ein solcher Listener bei ihnen registriert ist. Programmierer können Unterklassen von `SimpleOnGestureListener` schreiben, dort einige oder alle Listener-Methoden überschreiben und somit individuelle Reaktionen auf Gesten programmieren.

- *Vollziehen Sie anhand der Beispielapplikation „TouchGesturesAndroid“ die Erklärungen der vorangehenden Folien nach:*
  - *kommentierte Quellcode-Auszüge: Überschrift „Android: Einfache Gesten“.*
  - *Android Studio: Activity „GesturesOhneScale“.*
  - *Was machen also Gesture Detector und Gesture Listener zusätzlich zu dem, was onTouchEvent() macht?*

Vordefinierte Gesture Detectors (Forts.)

`android.view.ScaleGestureDetector`:

- für „Pinch Zooming“ (= Skalierung mit zwei Fingern)
- zugehörige Listener-Klasse:
  `ScaleGestureDetector.SimpleOnScaleGestureListener`
  - implementiert das Interface
    `ScaleGestureDetector.OnScaleGestureListener`
  - Methoden `onScale()`, `onScaleBegin()`, `onScaleEnd()`

→ *Beispielprogramm Android: Skalierungsgesten*

→ *Beispiel auch: http://android-developers.blogspot.com/2010/06/making-sense-of-multitouch.html*

- *Vollziehen Sie anhand der Beispielapplikation „TouchGesturesAndroid“ die Erklärungen der vorangehenden Folien nach:*
  - *kommentierte Quellcode-Auszüge: Überschrift „Android: Skalierungsgesten“.*
  - *Android Studio: Activity „ScaleGestures“.*
- *Betrachten Sie das Lehrvideo zum Thema „Drag&Drop“: http://youtu.be/l-5Px-7a_hE*

Selbstdefinierte Gesten

Paket `android.gesture`

- **ermöglicht die Definition und Erkennung beliebiger Gesten**

Zur Vorgehensweise siehe z.B.:

- ***http://androidresearch.wordpress.com/2012/01/10/working-with-gesture-api-in-android/***
- ***http://www.vogella.com/tutorials/AndroidGestures/article.html***

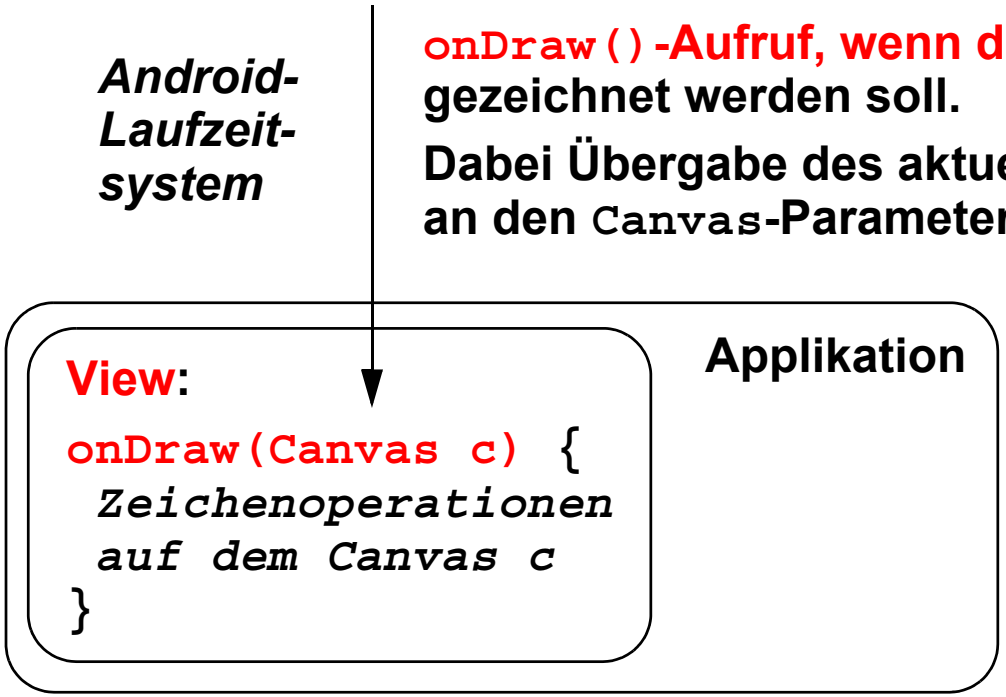
Der Benutzer definiert eine Geste, indem er über eine spezielle App eine Fingerbewegung auf dem Display aufzeichnet und mit einem Namen versieht. Details dazu findet man auf den angegebenen Web-Seiten.

Eine Android-Applikation kann dann eine solche Geste nach demselben Schema erkennen und verarbeiten wie bei vorgegebenen Gesten.

4. Android: Grafische Benutzeroberflächen

Unterkapitel 4.6. befasst sich mit der Integration von einfachen Grafiken, Animationen und Multimedia-Clips in Applikationen.

Grundprinzip der Grafikausgabe

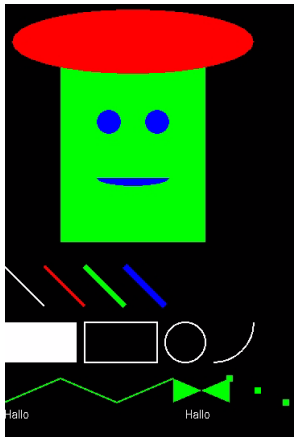


- Ein View definiert durch `onDraw(Canvas c)`, wie er zu zeichnen ist
- Das Laufzeitsystem ruft `onDraw(`*`aktueller Bildschirm`*`)` bei Bedarf auf

Die Grafik zeigt die grundlegende Vorgehensweise beim Zeichnen von Views auf dem Bildschirm (dem Display):

- Jeder View definiert eine Methode `onDraw()`, die festlegt, wie der View gezeichnet werden soll. `onDraw()` bekommt dabei die Oberfläche, auf der gezeichnet werden soll, als `Canvas`-Parameter übergeben (canvas = „Leinwand“). Im `onDraw()`-Körper werden dann auf dem Canvas Methoden wie `drawLine()` oder `drawRect()` aufgerufen.
- Entscheidet das Laufzeit-System, dass ein View gezeichnet werden soll (z.B. wenn eine Activity gestartet wird), so ruft es dessen `onDraw()` auf und übergibt als Canvas-Parameter eine Referenz auf das Gerätedisplay.

Grafikausgabe auf Canvas



Klasse `android.graphics.Canvas`

- definiert Methoden zum Zeichnen
  - geometrische Formen, Texte, Bilder, ...
- Details siehe *https://developer.android.com/reference/android/graphics/Canvas*

Methoden zum Zeichnen: `drawLine()`, `drawOval()`, `drawRect()`, ... Die Grafik, die der Screenshot zeigt, wurde durch Aufrufe solcher Methoden erzeugt (siehe Beispielapplikation weiter unten).

Canvas-Objekt:

- dient u.a. als `onDraw()`-Parameter
- repräsentiert eine Bitmap
  - realer Bildschirm
    - siehe vorige Folie
  - oder Bitmap innerhalb des Programms
    - kann später auf realem Bildschirm angezeigt werden
- Canvas-Methoden zeichnen auf dieser Bitmap

Schritte der Programmierung von Grafiken

1.) Unterklasse von `View` definieren

- insbesondere `onDraw(Canvas c)` ausprogrammieren
  - dort: Aufruf der Zeichenfunktionen auf dem Canvas `c`
  - Details siehe nächste Folie

2.) Objekte dieser Klasse erzeugen

- durch Konstruktor-Aufrufe

3.) Objekte auf der Oberfläche anzeigen

- durch `setContentView()`
- oder Einbettung in eine ViewGroup in der GUI

Weitere Details zur Definition von Views: Abschnitt 4.7

Die Folie skizziert die Schritte, mit denen man Grafiken durch selbstgeschriebene Views definieren und anzeigen kann.

Zeichnen in onDraw()

`onDraw()` erhält `Canvas c` als Parameter

oben gezeigt: Zeichnen mit `Canvas`-Zeichenoperationen

- im Körper von `onDraw()`:
  Aufruf von `c.drawRect()`, `c.drawOval()`, `c.drawLine()`, ...

→ *Beispielprogramm Android: Canvas*

alternativ: Zeichnen mit `Drawable`-Objekten

- Klasse `android.graphics.drawable.Drawable`
  - für Objekte, die gezeichnet werden können
  - Objekte können „Shapes“ haben: Rechteck, Oval, ...
- im Körper von `onDraw()`:
  Aufruf von `myDrawable.draw(c)`

→ *Beispielprogramm Android: Drawables*

Die Folie nennt zwei Möglichkeiten, wie man selbstgestaltete Grafiken auf dem Display anzeigen kann. Details sieht man in den Beispielapplikationen.

- *Beispielapplikation „GrafAnimMMAndroid“, Activities „GrafikMitCanvas“ und „GrafikMitDrawable“:*
  - *Führen Sie die Activities aus, um sich einen Eindruck aus Benutzersicht zu verschaffen (Auswahlpunkte „Grafik mit Canvas-Methoden“ und „Grafik mit Drawables“).*
  - *Arbeiten Sie die entsprechenden kommentierten Quellcode-Auszüge durch (Überschriften „Android: Canvas“ und „Android: Drawables“).*
  - *Schauen Sie sich die Quellcodes in Android Studio an und experimentieren Sie ein wenig damit.*

Zeitpunkt des onDraw()-Aufrufs

Laufzeitsystem entscheidet über Aufruf von `onDraw()`

View-Methode `invalidate()` bittet um diesen Aufruf

- **aber keine Garantie, dass er umgehend erfolgt**

⇒ **Ansatz ist nur für statische Grafiken und langsame Animationen geeignet**

`onDraw()` ist eine Call-back-Methode von Views: Sie wird vom Programmierer geschrieben und vom Laufzeitsystem aufgerufen. Das Laufzeitsystem entscheidet über den Zeitpunkt des Aufrufs, also darüber, wann der View neu gezeichnet wird.

Durch den Aufruf on `invalidate()` kann eine Applikation das Laufzeitsystem zwar darum „bitten", die Oberfläche neu zu zeichnen. Es ist aber nicht garantiert, dass dies dann unmittelbar geschieht.

Die zeitliche Kontrolle liegt hier also im Wesentlichen beim Laufzeitsystem. Schnelle Animationen, bei denen die Applikation die zeitliche Steuerung übernimmt, sind damit nicht möglich.

Animationen

Realisierung einer Animation als:

- **View Animation:**
  - **Frame Animation: Anzeige einer Folge von Bilddateien**
    - → ***Beispielprogramm Android: Frame Animation***
  - **Tween Animation: Transformationen eines Einzel-Views**
    - **Änderung von Größe, Position, Orientierung, Transparenz**
    - → ***Beispielprogramm Android: View Animation***

Bei einer Animation wird das Aussehen der GUI kontinuierlich verändert. Diese Folie und die folgenden zeigen verschiedene Möglichkeiten, solche Animationen zu programmieren.

Animationen (Forts.)

- **Property Animation**:
  - Transformationen der Eigenschaften von Objekten
    - keine Beschränkung auf Views
    - mehr Änderungsmöglichkeiten als bei View Animation
    - → *Beispielprogramm Android: Property Animation*
  - Lehrvideos, Details und Beispiel-App:
    http://www.nt.th-koeln.de/vogt/vma/videos.html
- **Layout Transitions**:
  - Animation des Hinzufügens / Löschens / Änderns von GUI-Elementen
  - Lehrvideos, Details und Beispiel-App:
    http://www.nt.th-koeln.de/vogt/vma/videos.html

Weitere Infos zu Animationen:
https://developer.android.com/guide/topics/resources/animation-resource

Mit einer *Property Animation* kann nicht nur die Anzeige von Views auf dem Display animiert werden, sondern es können im Prinzip beliebige Attributwerte (= Eigenschaften) beliebiger Objekte im Laufe der Zeit geändert werden.

Bei einer *Layout Transition* verschwinden / erscheinen GUI-Elemente (= Views) nicht schlagartig, sondern in einer sanften Bewegung.

Viele Details kann man den Lehrvideos und den zugehörigen Beispiel-Applikationen entnehmen.

- *Beispielapplikation „GrafAnimMMAndroid“:*
  - *Führen Sie die drei entsprechenden Activities aus, um sich einen Eindruck aus Benutzersicht zu verschaffen (Auswahlpunkte „Frame Animation“, „View Animation“ und „Property Animation“).*
  - *Arbeiten Sie die entsprechenden kommentierten Quellcode-Auszüge durch (Überschriften „Android: Frame Animation“, „Android: View Animation“ und „Android: Property Animation“).*
  - *Schauen Sie sich die Quellcodes in Android Studio an und experimentieren Sie ein wenig damit.*
- *Exkurs: Wenn Sie Zeit und Lust haben, so sehen Sie sich die entsprechenden Lehrvideos und die zugehörigen Programmcodes an.*

Animationen mit SurfaceView

Weitere Möglichkeit: Animation mit Threads selbst erstellen

Selbst programmierte Animationen durch Zusammenspiel von Objekten verschiedener Klassen und Interfaces:

- `SurfaceView`:
  View, in dem die Animation abläuft
- `SurfaceHolder`:
  zum Zugriff auf den SurfaceView
- `SurfaceHolder.Callback`:
  zur Verarbeitung von Rückmeldungen des SurfaceViews
- `Thread`:
  zur Steuerung der Animationsschritte

→ *Beispielprogramm Android: Animation auf SurfaceView*

Die Klasse `SurfaceView` ermöglicht es, animierte GUI-Elemente zu programmieren, deren Animation unter voller Kontrolle der Applikation abläuft.

Wie bei einem normalen View ist für ein `SurfaceView`-Objekt eine Methode `onDraw()` definiert, die festlegt, wie der View auf der GUI gezeichnet wird. `onDraw()` wird hier allerdings nicht vom Laufzeitsystem aufgerufen, sondern von einem Thread der Applikation selbst. Der Programmierer schreibt diesen Thread und kontrolliert damit die Animation. Üblicherweise läuft ein solcher Thread in einer Endlosschleife.

- *Starten Sie die Beispielapplikation „GrafAnimMMAndroid“ möglichst auf einem realen Gerät und gehen Sie zum Auswahlpunkt „Animation auf SurfaceView“. Tippen Sie auf den Bildschirm, um das Icon anzuhalten, machen Sie dann eine Wischbewegung usw.*

- *Die Applikation sollte auch auf einem Emulator laufen können. Allerdings gibt es dort manchmal Probleme mit der grafischen Darstellung des SurfaceViews, wenn der Grafik-Beschleuniger des Emulators nicht korrekt funktioniert. Verwenden Sie dann eine anderen Emulator (z.B. von Genymotion) oder ein reales Gerät.*

Animationen mit SurfaceView

`SurfaceView`:

- `View`, in dem eine Animation abläuft
  - gesteuert durch einen dedizierten Thread
  - also: Timing unter Kontrolle des Threads!
- definiert die Methode `onDraw()`
  - enthält Zeichenoperationen auf einem Canvas
  - legt also fest, was angezeigt werden soll
    - die Animationsschritte
  - wird durch den Thread aufgerufen
- besitzt einen SurfaceHolder
  - ermöglicht den Zugriff auf den SurfaceView
  - wird durch `SurfaceView`-Methode `getHolder()` geliefert

Animiert wird also ein Objekt der Klasse `SurfaceView`, dessen `onDraw()` wiederholt von einem speziell dafür programmierten Thread aufgerufen wird und dann die einzelnen Animationsschritte zeichnet.

Timing unter Kontrolle des Threads:
Die Zeitpunkte, zu denen die Animationsschritte gezeichnet werden, werden durch den Thread im Programm bestimmt, nicht durch das Laufzeitsystem.

Der animierende Thread greift über einen *Surface Holder* auf den Surface View zu.

Die Erläuterungen auf dieser und den folgenden Folgen lassen sich am besten anhand der Beispielapplikation nachvollziehen.

Animationen mit SurfaceView

`SurfaceHolder.Callback`:

- wird bei einem SurfaceView registriert
- wird benachrichtigt
  bei Änderungen des Zustands des SurfaceViews
  - durch Aufruf einer seiner Callback-Methoden
    - `surfaceCreated()`
    - `surfaceChanged()`
    - `surfaceDestroyed()`
- insbesondere: mit dem Zeichnen beginnen,
  wenn der SurfaceView bereit ist
  - d.h. in `surfaceCreated()`:
    Starten des Animations-Threads

Randbemerkung zur Programmierung: Ein laufender Thread kann den Akku merklich belasten. Man sollte ihn also erst starten, wenn er tatsächlich benötigt wird (in der Methode `surfaceCreated()`), und ihn unbedingt wieder stoppen, wenn er nicht mehr benötigt wird – insbesondere, wenn der SurfaceView nicht mehr angezeigt wird (in der Methode `surfaceDestroyed()`).

Animationen mit SurfaceView

Animations-Thread

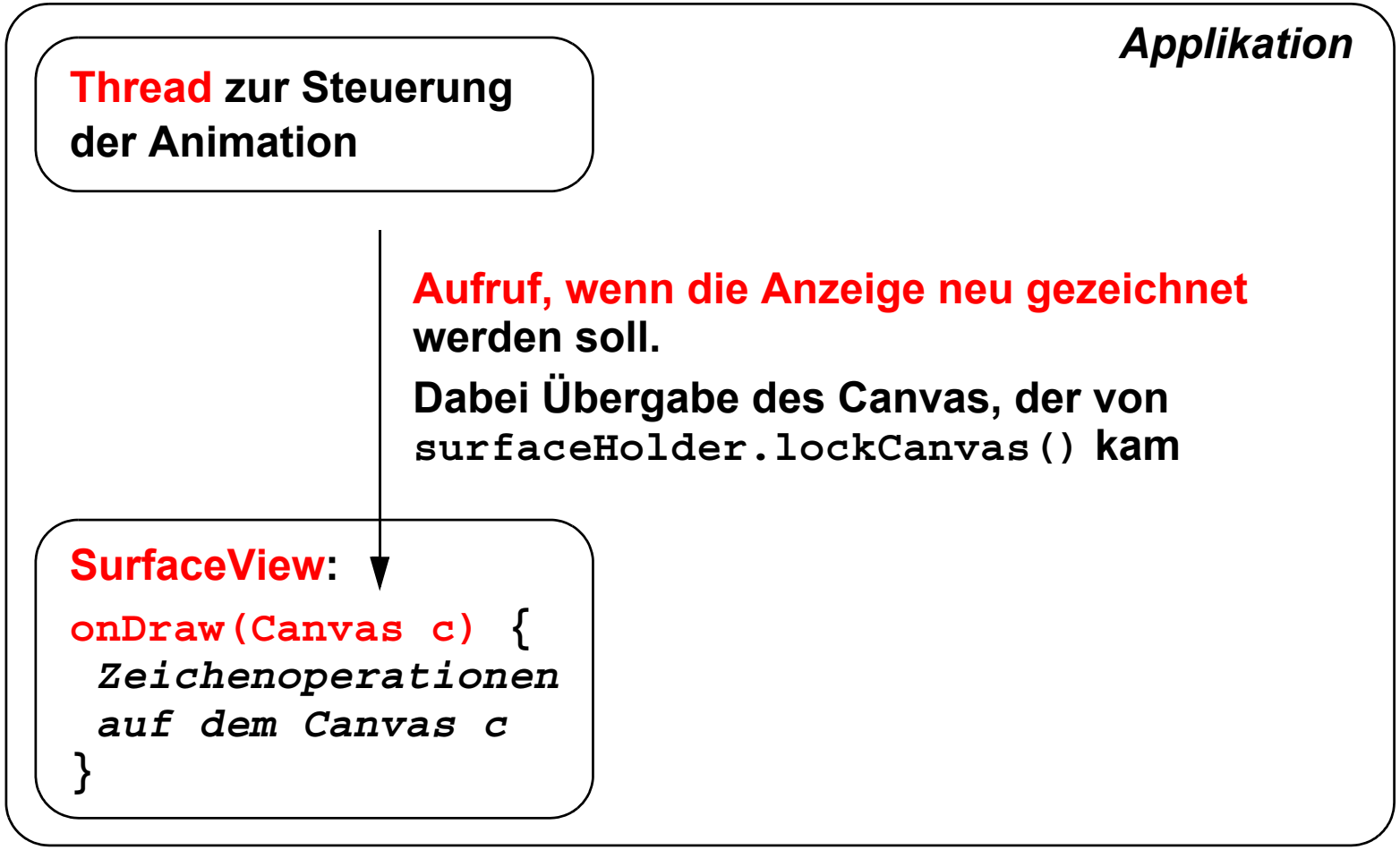
- „rendert“ die Oberfläche
- wird gestartet,
  wenn der SurfaceView zum Zeichnen bereit ist
- führt eine Endlosschleife mit den folgenden Schritten aus:
  - Belegung eines Canvas,
    über den auf dem SurfaceView gezeichnet werden kann
    - `Canvas c = surfaceHolder.lockCanvas()`
  - Aufruf der Zeichenfunktion des SurfaceViews
    - `onDraw(c)`
  - Freigabe des Canvas
    - `Canvas c = surfaceHolder.unlockCanvasAndPost()`
    - damit Anzeige des neuen SurfaceView-Inhalts
      auf dem Display

Animations-Thread wird gestartet, wenn der SurfaceView zum Zeichnen bereit ist
= wenn das Laufzeitsystem seine Methode `surfaceCreated()` aufruft. In den Körper von `surfaceCreated()` programmiert man also einen Aufruf der `start()`-Methode des Animations-Threads (siehe vorige Folie und Beispiel-Applikation).

Die Endlosschleife dient dazu, durch `onDraw()`-Aufrufe die Animation Bild für Bild zu zeichnen. Mit „neuer SurfaceView-Inhalt“ ist also das jeweils nächste Bild der Animation gemeint.

Man beachte den Unterschied zu einem normalen View: Dort ruft das Laufzeitsystem `onDraw()` auf, hier tut es der Thread und kann damit die Zeitpunkte des Zeichnens kontrollieren.

Also: Grundprinzip der Grafikausgabe mit SurfaceView



- vergleiche dazu Folie 108

Die Grafik zeigt, dass die `onDraw()`-Methode eines `SurfaceView`-Objekts von einem Thread aufgerufen wird, der zur Applikation gehört. Der zeitliche Ablauf, in dem die grafische Darstellung aufgefrischt wird, liegt damit unter der Kontrolle der Applikation selbst und damit des Programmierers.

- *Vollziehen Sie anhand der Beispielapplikation „GrafAnimMMAndroid“ die Erklärungen der vorangehenden Folien nach:*
  - *kommentierte Quellcode-Auszüge:*
    Überschrift „Android: Animation auf SurfaceView“.
  - *Android Studio:*
    Activity „Animation“, View „AnimationExampleView“.

Open GL ES

- Animationen auch mit OpenGL ES
  - „Open Graphics Library for Embedded Systems”
  - umfassendes Paket für 2D- und 3D-Animation
  - leistungsstark, da hardwarenah
  - aber komplexe Programmierung
- Details unter:
  - *http://www.learnopengles.com/android-lesson-one-getting-started/*
    - sehr ausführliches Tutorium
    - umfassendes Beispiel mit Quellcode: *https://github.com/learnopengles/Learn-OpenGLES-Tutorials*
  - *https://developer.android.com/guide/topics/graphics/opengl*
  - *https://developer.android.com/training/graphics/opengl/*

> ***Laden Sie, wenn Sie Zeit und Lust haben, das Beispielprogramm https://github.com/learnopengles/Learn-OpenGLES-Tutorials herunter und gehen Sie es durch.***

Android Multimedia Framework

Aufzeichnung und Wiedergabe von

- **Standbildern**
- **Audio**
- **Video**

Grundlegende Klassen:

- `android.media.MediaPlayer`, `android.widget.MediaController` **zur Steuerung der Wiedergabe von Medien**
- `android.media.MediaRecorder`: **zur Steuerung der Aufzeichnung von Medien**
- `android.hardware.Camera`: **zum Zugriff auf die Kamera des Geräts**
- `android.widget.VideoView`: **zur Anzeige von Videos**

→ ***Beispielprogramm Android: Multimedia***

`android.hardware.Camera` **ist seit API-Level 21 „deprecated" und wurde durch das Paket** `android.hardware.camera2` **ersetzt.**

- *Beispielapplikation „GrafAnimMMAndroid“, Activity „Multimedia“:*
  - *Führen Sie die Activity aus, um sich einen Eindruck aus Benutzersicht zu verschaffen (Auswahlpunkt „Multimedia“).*
  - *Arbeiten Sie die entsprechenden kommentierten Quellcode-Auszüge durch (Überschrift „Android: Multimedia“).*
  - *Schauen Sie sich die Quellcodes in Android Studio an und experimentieren Sie ein wenig damit.*

Webseiten mit weiteren Informationen

„Audio & Video“:
https://developer.android.com/guide/topics/media/

„Drawables overview“:
https://developer.android.com/guide/topics/graphics/drawables

„Images and graphics“:
https://developer.android.com/guide/topics/graphics/

„Animation Resources“:
https://developer.android.com/
guide/topics/resources/animation-resource

4. Android: Grafische Benutzeroberflächen

Unterkapitel 4.7 weist kurz auf weitere Themen und Techniken hin, die bei der GUI-Programmierung eine Rolle spielen. Manche von ihnen werden in späteren Kapiteln erneut aufgegriffen.

Weitere GUI-Elemente

`WebView`: `View` zur Anzeige von Web-Seiten

- Klasse `android.webkit.WebView`
- Details:
  - Kapitel 7
  - *https://developer.android.com/guide/webapps/webview*

`MapView`: `View` zur Anzeige von Landkarten

- Paket `com.google.android.gms.maps` mit Klasse `MapFragment`
- Details:
  - Kapitel 8
  - *https://developers.google.com/maps/documentation/android-sdk/intro*

In einem *WebView* kann eine Web-Seite ähnlich wie in einem Browser grafisch angezeigt werden.

Bildschirm und Tastatur

Zugriff auf Display-Eigenschaften über den Window Manager

- `getWindowManager().getDefaultDisplay()`
  liefert aktuelles Display
  - `getWindowManager()` ist eine `Activity`-Methode
- Klasse `Display` mit Methoden wie:
  - `getSize()` (, `getWidth()`, `getHeight()`)
  - `getRotation()` (, `getOrientation()`)

liefert aktuelles Display: als Objekt der Klasse `Display`

Mit diesen `Display`-Methoden kann man Eigenschaften des aktuellen Displays abfragen.

Zugriff auf das Soft Keyboard über den Input Method Manager

- `getSystemService(Context.INPUT_METHOD_SERVICE)`
  liefert aktuellen Input Method Manager
  - `getSystemService()` ist eine `Context`-Methode
- Klasse `InputMethodManager` mit Methoden wie:
  - `showSoftInput()`, `hideSoftInputFromWindow()`

Mit diesen Methoden kann man das Soft Keyboard erscheinen und wieder verschwinden lassen.

Definition eigener View-Klassen

Viele Views sind vordefiniert

- `TextView`, `EditText`, `Button`, ...

Programmierer können zudem eigene Views definieren

- **können sie dann in XML-Layouts aufnehmen,**
- **innerhalb dieser Layouts an der GUI anzeigen**
- **und auf ihre Attribute zugreifen**

→ Benutzung wie vordefinierte Views

Man kann mit selbstdefinierten Views auf dieselbe Weise arbeiten wie mit den vordefinierten Views:

- **Einbettung in ein XML-definiertes Layout; dabei Verwendung von Attributen, die für diese View-Klasse definiert wurde.**
- **Anzeige innerhalb eines Layouts auf dem Display.**
- **Reaktion auf GUI-Ereignisse bzgl. des Views, z.B. in der Callback-Methode `onTouchEvent()` bei einer Berührung.**

Definition eigener View-Klassen

Vorgehensweise für eigenen View `MyView`:

- **XML-Attribute für `MyView` festlegen**
  - in `/res/values/attrs.xml`
  - z.B.

```
<declare-styleable name="MyViewAttrs">
  <attr name="text" format="string"
  <attr name="color" format="integer" ...>
```

Diese Folie und die folgende skizzieren die Schritte, mit denen man selbstgestaltete Views definiert.

Mit XML-Attributen, die auf diese Weise definiert werden, geht man so um wie mit den vordefinierten Attributen der vordefinierten Views. Man weist ihnen in einer XML-Layout-Definition Werte zu und greift aus Java-Code mit `get()`-Methoden darauf zu. Insbesondere nutzt die `onDraw()`-Methode, die den View auf das Display zeichnet, diese Werte.

Die Beispiel-Definition von `MyViewAttrs` spezifiziert die Attribute (anzuzeigender Text, Farbe) für die `View`-Unterklasse `MyView`, die auf der folgenden Folie definiert wird.

Definition eigener View-Klassen (Forts.)

- `View`-Unterklasse `MyView` schreiben
  - mit Konstruktor

    ```
    MyView(Context context, AttributeSet attrs)
    ```

    - Aufruf bei Anzeige eines XML-Layouts mit `MyView`-Element
    - lädt Werte aus den XML-Attributen des Layouts
  - mit Methode `onDraw(Canvas canvas)`
    - definiert, wie `MyView` auf dem Display gezeichnet wird
    - greift dazu auf die geladenen Werte zu
  - mit Methode `onMeasure(...)`
    - berechnet, wie groß der View auf dem Display erscheinen soll
  - ggf. mit weiteren Methoden
    - z.B. `onTouchEvent()`

Weitere Details:
https://developer.android.com/training/custom-views/create-view

Das Laufzeitsystem ruft den Konstruktor auf, wenn es ein XML-definiertes Layout, das einen solchen View enthält, auf dem Display anzeigen will. Es erzeugt damit (wie bei vordefinierten Views auch) ein entsprechendes Java-Objekt und übergibt über `attrs` die Attributwerte aus dem XML-Layout.

Das Laufzeitsystem ruft `onDraw()` auf, um das View-Objekt auf dem Display zu zeichnen. Dabei wird mit `onMeasure()` die Größe der Darstellung berechnet. `onMeasure()` trifft Fallunterscheidungen danach, ob im Layout eine feste Höhe und Breite angegeben ist oder die Ausdehnung durch `match_parent` (früher: `fill_parent`) oder `wrap_content` spezifiziert ist.

> *Beispielapplikation „SelfDefinedView“*

> *Führen Sie die Applikation aus und betrachten Sie ihre Ausgabe. Drücken Sie auf den View, der unten im Display dargestellt wird.*

> *Schauen Sie sich in Android Studio die Java-Quellcodes an. Suchen Sie nach den Codestücken, die festlegen, wie Views der hier definierten Typen „ZweiZeilenText“ und „ThreeWaySwitch“ auf das Display gezeichnet werden. Suchen Sie zudem nach der Methode, die auf das Drücken auf den ThreeWaySwitch reagiert.*

> *Betrachten Sie dann, wie die Attribute der beiden neuen View-Klassen in attrs.xml definiert werden und wie die Layoutdefinition in activity_main.xml darauf Bezug nimmt.*

> *Stellen Sie die Layout-Datei activity_main.xml im Design-Modus dar. Clicken Sie unter „Palette“ in der Rubrik „Custom“ auf „CustomView“. Wählen Sie in dem Fenster, das sich öffnet, „ThreeWaySwitch“ und klicken Sie „OK“. Sie*

können nun einen neuen View dieses Typs mit der Maus in der GUI platzieren. Tun Sie das und stellen Sie in der Textansicht fest, dass der XML-Code nun ein entsprechendes neues Element enthält.

Nebenläufigkeit bei der GUI-Steuerung

Im einfachsten Fall:
Nur *ein* „Main Thread“, der alles ausführt

- **Event Loop:**
  Schleife zur Entgegennahme von GUI-Ereignissen
  - **Clicks auf Buttons, Gesten, ...**
- **Listener:**
  Reaktion auf jeweils eines dieser Ereignisse

Sicherstellen: Main Thread darf nicht blockieren

- **sonst: Einfrieren der Oberfläche**
- **„ANR“ = „Application Not Responding“**

Ansatz dazu: Eigene Threads für die Listener

- **hierzu z.B.: Class `AsyncTask`**
- **detaillierte Betrachtung in Kap. 6.6**

Main Thread = UI Thread

Die *Event Loop* ist eine Endlosschleife, in der der Main Thread / UI Thread ein Ereignis nach dem anderen entgegennimmt und darauf reagiert.

Eine Gefahr besteht darin, dass der Main Thread für die Ausführung eines Listeners zu lange braucht oder dort sogar blockiert. Er kann dann nicht auf weitere GUI-Ereignisse reagieren, so dass die GUI einfriert. Lang laufende oder sogar blockierende Listener lagert man daher besser in gesonderte Threads aus. Kapitel 6.6 beschäftigt sich näher mit diesem Thema.

4. Android: Grafische Benutzeroberflächen

Unterkapitel 4.8. zeigt, wie man die grafische Oberfläche einer Applikation an die unterschiedlichen Bildschirmgrößen von Smartphones und Tablets anpassen kann.

Motivation

Problem: Stark unterschiedliche Bildschirmgrößen

- Smartphones klein, Tablets groß
- „Ein Layout für alle“ nicht angemessen

Lösung: „Fragments“

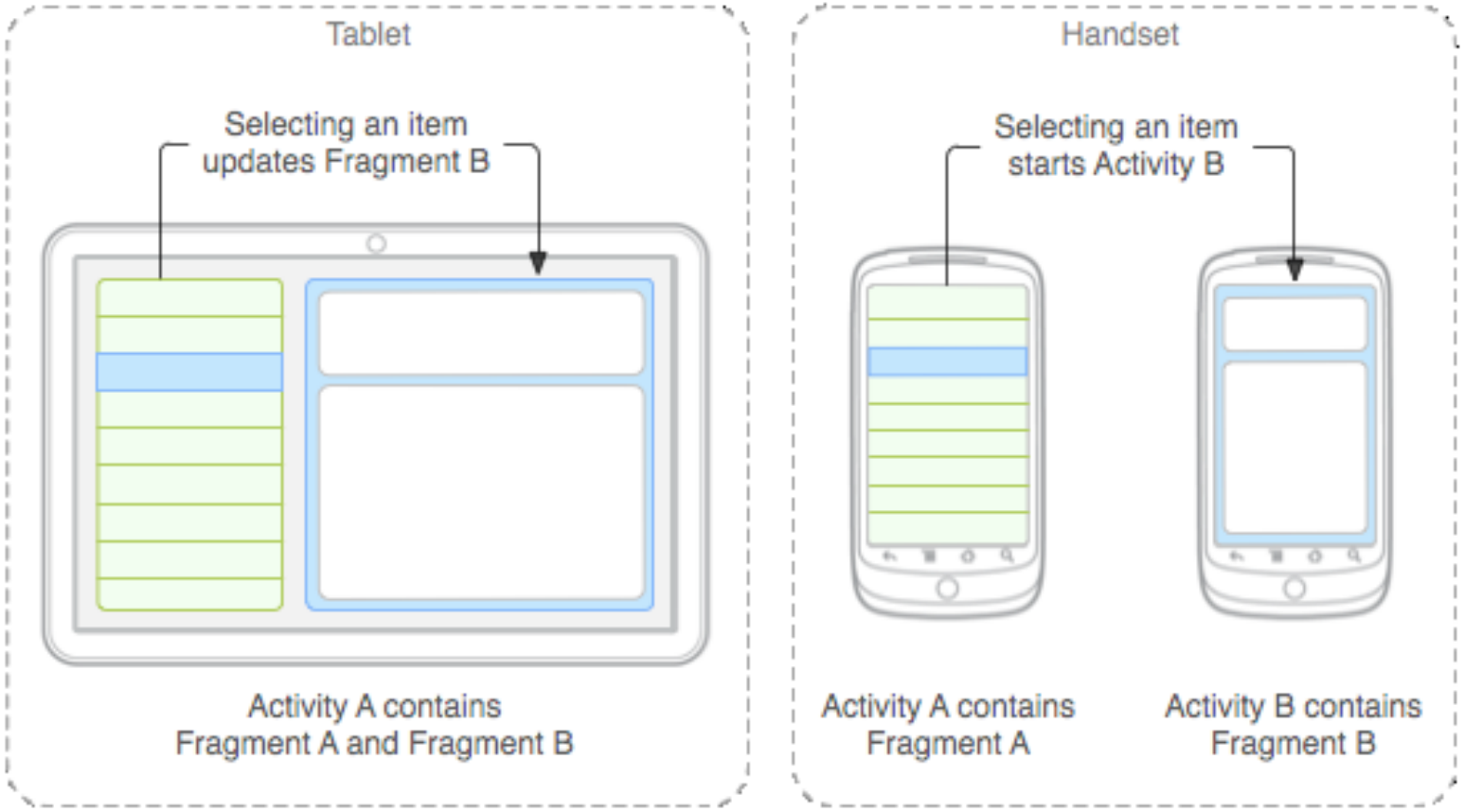
- Seit API-Level 11 (= Android 3.0)
- Neue Ebene zwischen Activities und Views
  - Activity enthält ein oder mehrere Fragments
    - Anzahl und Position je nach Bildschirmgröße unterschiedlich
  - Fragments enthalten wiederum Views
    - zeigen also die eigentlichen Inhalte

Fragments sind Teil-GUIs, die mehrere Views enthalten können. Je nach Größe des Displays kann eine Activity ein oder mehrere Fragments gleichzeitig anzeigen. Dies wird automatisch vom Laufzeitsystem entschieden.

Durch die Einführung von Fragments werden Applikationen anpassungsfähiger. Dafür wird die Programmierung komplexer, da hier eine zusätzliche Ebene zwischen die Gesamt-GUI einerseits und die einzelnen Views andererseits eingeschoben wird.

Als Android-Programmierer wird man dazu gedrängt, Fragments zu verwendet, indem z.B. Klassen, die noch nicht auf Fragments basierten, als „deprecated” gekennzeichnet wurden. Wie die vorherigen Abschnitte zeigen, kann man dennoch ohne Fragments auskommen.

Beispiel



Quelle: http://developer.android.com/guide/components/fragments.html

→ ***Beispielprogramm Android: Fragments***

Die Grafik zeigt die GUI einer Applikation, in der zwei Fragments definiert sind. Fragment A enthält eine Auswahlliste, mit der die Anzeige in Fragment B gesteuert wird:

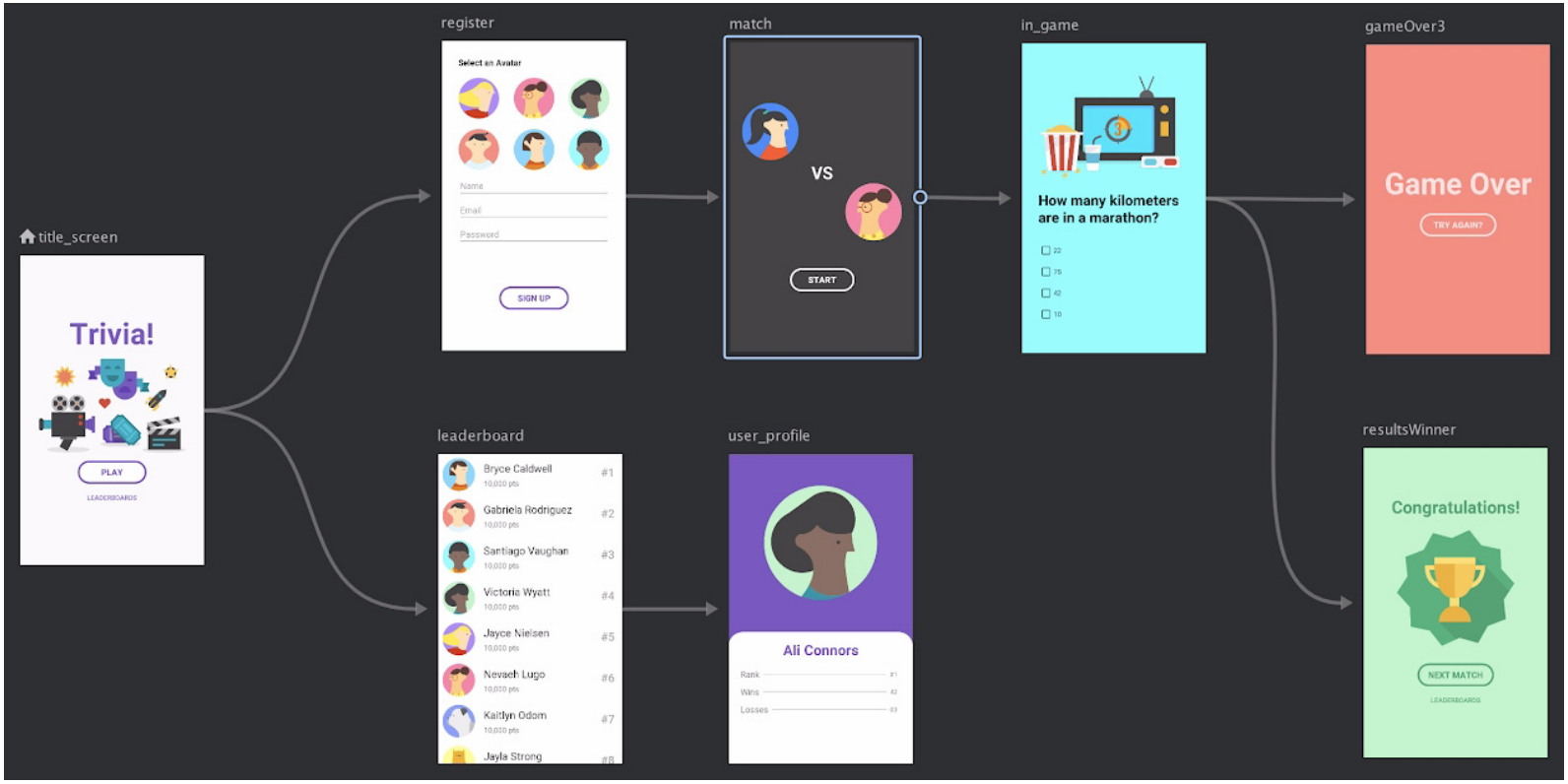
- **Auf einem Tablet mit einem großen Bildschirm werden beide Fragments nebeneinander angezeigt. Eine Auswahl im linken Fragment aktualisiert die Anzeige im rechten Fragment. Beide Fragments gehören hier zur selben Activity.**
- **Auf einem Smartphone mit einem kleinen Bildschirm wird nur Fragment A mit der Liste angezeigt. Eine Auswahl schaltet dann zur Anzeige von Fragment B um. Hier gibt es zwei getrennte Activities, die jeweils eines dieser Fragments anzeigen.**

- *Beispielapplikation „FragmentsAndroid“*
  - *Führen Sie die Applikation auf einem Smartphone und einem Tablet (oder entsprechenden Emulatoren) aus und bemerken Sie die Unterschiede in der Darstellung.*
  - *Schauen Sie sich in Android Studio die Dateien des Projekts sowie die entsprechenden kommentierten Quellcodeauszüge an. Stellen Sie fest, wie realisiert wurde, dass die Darstellung auf Smartphone und Tablet unterschiedlich aussehen.*

4. Android: Grafische Benutzeroberflächen

Unterkapitel 4.9. reißt kurz an, welche Unterstützung Android bei der Programmierung von Applikationen bietet, bei denen der Benutzer zwischen verschiedenen Oberflächen hin und her navigiert.

Navigieren durch Oberflächen



Quelle: https://android-developers.googleblog.com/2018/05/android-studio-3-2-canary.html

Typische Struktur einer Applikation:
Oberflächen, durch die der Anwender navigiert

→ benötigt: Unterstützung bei der Programmierung

Der Screenshot illustriert eine typische Applikationsstruktur: Die Applikation besteht aus einem System von Oberflächen, zwischen denen der Anwender durch seine Aktionen „navigiert“, also durch Drücken von Buttons, Auswahl von Menupunkt usw. hin- und herwechselt.

Ein Programmierwerkzeug (hier: Android Studio) sollte hierfür eine Programmierunterstützung bieten, die einem das Schreiben von „Boilerplate Code” (= immer wiederkehrenden länglichen Codestücken) erspart.

Navigation: Unterstützung durch Android Jetpack

Navigation Editor: zur Programmierung in Android Studio

- Visueller Editor
- Erstellung des „Navigationsgraphen" einer Applikation
  - siehe vorige Folie
- Grundidee:
  - Verknüpfung des Graphen mit der Main Activity
  - Navigation durch Umschalten zwischen Fragments

Navigation Architecture Component: dahinterliegender Code

- Paket `androidx.navigation.*`
- Spezifikation der Navigationsstruktur
  - in XML
- Navigation bei der Ausführung
  - durch Java-/Kotlin-Methoden

Android Jetpack ist eine Sammlung von Bibliotheken, Entwicklerwerkzeugen und Architekturempfehlungen, die mit Android Studio 3.2 eingeführt wurden. Ein Bestandteil von Jetpack sind der *Navigation Editor* und die *Navigation Architecture Component*. Mit dem Editor können Navigationsstrukturen visuell programmiert werden, wie in der vorigen Folie illustriert. Die so spezifizierten Strukturen werden in XML-Dateien gespeichert. Bei der Ausführung navigieren Methoden der Navigation Architecture Component durch die Oberflächen der Applikation.

Navigation: Details

Anleitung zur Vorgehensweise:

- *https://developer.android.com/topic/libraries/architecture/navigation*

Dokumentation des Navigation-Pakets:

- *https://developer.android.com/reference/androidx/navigation/package-summary*

Videos:

- *https://www.youtube.com/watch?v=8GCXtCjtg40*
- *https://www.youtube.com/watch?v=GOpeBbfyb6s*
  - instruktive Schritt-für-Schritt-Erläuterung

→ *Beispielprogramm Android: Navigation*

Die Folien dieses Kurses können den Navigation Editor und die dahinterliegenden Konzepte nicht im Einzelnen darstellen, da dies den Umfang sprengen würde. Detaillierte Informationen findet man unter den angegebenen Links.

Insbesondere zu empfehlen ist das zweite Video. Es zeigt „auf den Punkt", wie man mit Android Studio die Navigation durch die Oberflächen einer Applikation programmiert.

- *Beispielapplikation „AppNavigationAndroid“*
  - *Schauen Sie sich in Android Studio die Dateien des Projekts an – insbesondere „navigation/nav_graph.xml” in der Text- und in der Design-Darstellung. Lesen Sie zudem die Kommentare in MainActivity.java oben.*
  - *Anhand des Videos https://www.youtube.com/watch?v=GOpeBbfyb6s können Sie nachvollziehen, wie die Navigation mit Buttons programmiert wurd.*