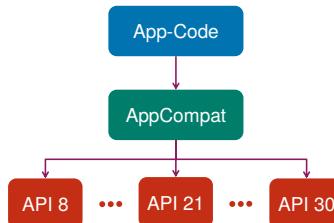


Rückwärtskompatibilität

AppCompat aus AndroidX

- AppCompat ist eine der Libraries in Jetpack
- Sie macht **neue Features auf tieferen API Levels** verfügbar, ohne Version-Checks
- Grundidee: Verwendung von Elementen aus dem **androidx**-Namespace anstelle der normalen Android-Komponenten
 - AppCompatActivity statt Activity
 - AppCompatButton statt Button
 - AppCompatDialog statt Dialog
 - etc.



41 | MGE | 02 – Android – Grundkonzepte und Rückwärtskompatibilität

30.09.2022



Mit "layout"-Parametern teilen die Kindern ihren Eltern mit, wie sie angeordnet werden möchten (Wunsch).
=> Darum heisst es auch "layout_margin" (Eltern-Eigenschaft) aber nur "padding" (Kind-Eigenschaft).
=> Wunsch: GetMeasuredWidth(), Realität: GetWidth(),

Layouts – Teil 1

layout_width / layout_height

- Für alle Layouts definiert sind
 - **layout_width**: Gewünschte Breite des Childs
 - **layout_height**: Gewünscht Höhe des Childs
- Zulässige Werte sind
 - **match_parent**: so gross wie möglich ("so gross wie mein Eltern-Layout")
 - **wrap_content**: so klein wie möglich ("so gross, dass mein Inhalt darin Platz findet")
 - Eine Zahl inkl. gültiger **Dimension** (eher unüblich, wenn verwendet, dann mit **dp**)

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello Android!"
        android:textSize="20sp"/>
</LinearLayout>
```

Höhe und Breite müssen auf jedem Element zwingend definiert sein!

12 | MGE | 03 - Android - GUI Programmierung

07.10.2022



Verschachtelung hat negativen Einfluss auf die Performance.

=> Flache, breite Hierarchien werden daher bevorzugt.

Layouts – Teil 1

Relative Layout => Kein Pendant in .NET MAUI

- Kinder werden relativ zueinander angeordnet
- Identifizierung der referenzierten Kinder über Resource IDs
- Insgesamt sind 23 (!) **Layout-Parameter** verfügbar
 - **android:layout_alignParentTop="true"**
 - **android:layout_toStartOf="@id/..."**
 - **android:layout_alignStart="@id/..."**
 - ...
- Sehr mächtig – dient oft als effizienter Ersatz für verschachtelte Linear Layouts



22 | MGE | 03 - Android - GUI Programmierung

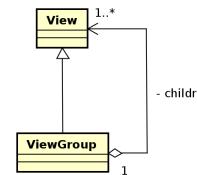
07.10.2022



Einstieg

View und ViewGroup

- Die Basisklasse aller GUI-Elemente ist **View**
 - Belegt einen rechteckigen Bereich
 - kümmert sich um Darstellung und Event-Verarbeitung
- Die Ableitung **ViewGroup** enthält **View**-Objekte
 - Modellierung eines hierarchischen **View-Baumes**
 - SEP2: [Composite Design Pattern](#)
- **ViewGroup**-Klassen ...
 - ordnen ihre Kinder nach einem Muster an
 - werden auch **Layouts** oder **Container** genannt
 - sind strukturierend, selber aber unsichtbar



8 | MGE | 03 - Android - GUI Programmierung

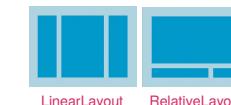
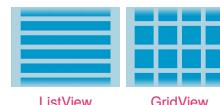
07.10.2022

Layouts – Teil 1

Hauptkategorien

Einfache Layouts (Teil 1)

- Endliche Menge von Inhalten
- UI-Gestaltung komplett im XML möglich
- AndroidX: [ConstraintLayout](#)

Android Developer 2020: Layout Overview
https://bit.ly/3IzQfU

07.10.2022



11 | MGE | 03 - Android - GUI Programmierung



Layouts – Teil 1

Frame Layout ≈ AbsoluteLayout (NET MAUI)

- Kinder werden **übereinander** angeordnet
 - Beispiel 1: Live-Kamerabild mit Auslöse-Button und Hilfslinien
 - Beispiel 2: Kartenansicht mit Zoom-Controls
- Anpassung der "Höhe" über dem Bild
 - Standardmäßig gilt die Reihenfolge im XML (später = höher)
 - Manuelle Anpassung mit **android:translationZ** möglich



20 | MGE | 03 - Android - GUI Programmierung

07.10.2022



Layouts – Teil 1

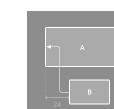
Constraint Layout

- Das modernste und flexibelste Layout in Android
 - Teil von Android Jetpack / AndroidX
 - Speziell für grosse, komplexe Layouts **mit flacher Hierarchie** entworfen
 - Diese Flexibilität bringt aber auch eine steile Lernkurve mit sich
- Grundidee ist ähnlich zum Relative Layout
 - Definieren von **Beziehungen zwischen Views**
 - Pro View muss mindestens eine horizontale und eine vertikale **Einschränkung** definiert werden
- Guter Support in Android Studio
 - Rein visuelle Editierung in der IDE bevorzugt (!)
 - Automatische Umwandlung von existierenden Layouts in Constraint Layouts möglich

=> **layout_width="0dp"** bedeutet bei diesem Layout "Match Constraint".

25

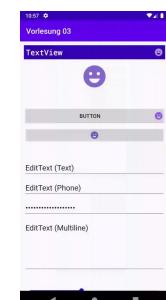
| MGE | 03 - Android - GUI Programmierung



07.10.2022



```
<ScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <!--Gehört ein Kind hier-->
</ScrollView>
```



07.10.2022



28 | MGE | 03 - Android - GUI Programmierung

Widgets Bezeichnung "Widgets" ist unscharf. Besser ist "Controls".
=> Widget: Sammelbegriff für "visuelle Elemente"

EditText

- **EditText** dient als Eingabefeld für Texte und Zahlen
- **android:inputType** beeinflusst Verhalten und aussehen
 - Format (`text, phone, date, textPassword, ...`)
 - Korrekturoptionen (`textAutoComplete, textAutoCorrect, ...`)
 - Darstellung (`textCapWords, textCapSentences, ...`)
 - Mehrzeiligkeit (`textMultiLine`)
 - ... [alle Optionen](#) ...
- Abhängig vom Typ wird passendes Keyboard angezeigt
- **inputType**-Werte sind mit Pipes kombinierbar, z.B.: `android:inputType="textCapCharacters|textAutoCorrect"`



Widgets Weitere Controls: Checkboxes, Radio Buttons, Seek Bars, FABs, etc.

EditText – Validierung von Eingaben

- **EditText** bietet eine einfache Möglichkeit, Inputvalidierung durchzuführen
- Mit `setError()` wird eine Nachricht gesetzt, bei jeder Änderung wird diese zurückgesetzt
- Fehler-Icon ist anpassbar (bei Bedarf)
- Dazu muss eine Implementierung von `TextWatcher` als Listener registriert werden: `myEditText.addTextChangedListener(new TextWatcher() { ... })`
- Das Interface umfasst 3 Methoden
 - `beforeTextChanged`: wird aufgerufen, bevor der Text geändert wird (wir sehen noch den alten Text)
 - `onTextChanged`: wird aufgerufen, sobald der Text geändert hat (meistens interessiert uns nur diese Methode)
 - `afterTextChanged`: wird aufgerufen, nachdem der Text geändert wurde (hier haben wir noch die Chance den Text anzubauen → Achtung vor Endlosschleife!)



```
EditText passwordInput = findViewById(R.id.edit_password);
passwordInput.addTextChangedListener(new TextWatcher() {
    // Methoden gekürzt
    @Override
    public void beforeTextChanged(CharSequence s, int start, int count, int after) {
        if (start < 8) {
            passwordInput.setError("Passwort zu kurz.");
        }
    }
});
```

07.10.2022 OST

33 MGE | 03 - Android - GUI Programmierung

07.10.2022

Beispiele von Controls / Widgets:



Darstellung von Collections

- Die Darstellung von Collections ist eine häufige Anforderung in Mobile Apps
- Ein Adapter vermittelt zwischen Darstellung und Datenquelle (SEP2: [Adapter-Pattern](#))
- Dank diesem Mechanismus bleibt unser Datenmodell frei von UI-Logik



07.10.2022

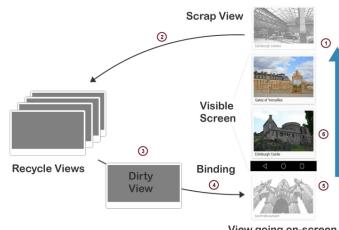
OST

View-Recycling: Objekte zu erzeugen ist aufwändig (Performance).
Darum wollen wir bestehende Objekte wiederverwenden / recyceln.

Layouts – Teil 2

RecyclerView

- Die **RecyclerView** ist eine moderne Alternative zu **ListView** und **GridView**
 - Integriertes View-Recycling
 - Erzwungene Verwendung von **View Holdern**
 - Weniger Overhead im eigenen Code
 - Layout-Flexibilität durch **LayoutManager**
- Die View ist, ihr ahnt es, Teil von **AndroidX**
 - Verwendung wird von Google empfohlen



Layouts – Teil 2

RecyclerView – Beispiel

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/recycler_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    >/>
```

main_activity.xml


```
setContentView(R.layout.activity_recyclerview);
RecyclerView recyclerView = findViewById(R.id.recycler_view);
recyclerView.setLayoutManager(new LinearLayoutManager(this));
recyclerView.setAdapter(usersAdapter);
```

MainActivity.java

View wird neu erstellt

```
public class UsersAdapter extends RecyclerView.Adapter<ViewHolder> {
    private ArrayList<User> users;

    @Override
    public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        Context context = parent.getContext();
        LayoutInflater inflater = LayoutInflater.from(context);
        View view = inflater.inflate(
            android.R.layout.simple_list_item_2,
            parent,
            false
        );
        return new ViewHolder(
            view,
            view.findViewById(android.R.id.text1),
            view.findViewById(android.R.id.text2)
        );
    }

    @Override
    public void onBindViewHolder(ViewHolder holder, int position) {
        User user = this.users.get(position);
        holder.text1.setText(user.getName());
        holder.text2.setText(user.age + " Jahre");
    }

    @Override
    public int getItemCount() {
        return this.users.size();
    }
}
```

UsersAdapter.java

07.10.2022 OST

48 MGE | 03 - Android - GUI Programmierung

07.10.2022

OST

Anmerkung: Leere Listen sind unschön.
=> Besser: Platzhalterbild und Info, wie Elemente in die Liste kommen.

54 MGE | 03 - Android - GUI Programmierung

07.10.2022

Einfache Rückmeldung ohne Interaktion
Toast.makeText(...).show()

Widgets

UI-Elemente ohne XML

- Android enthält UI-Widgets, welche direkt **aus dem Code heraus** erzeugt werden
- Die **Anpassbarkeit** dieser Widgets ist oft eingeschränkt (Farben, Texte, etc.)
- Beispiele
 - [Toasts](#)
 - [Snackbars](#)
 - [Dialoge](#)
 - [Notifications](#)



Android Developer 2020: Toasts
<https://bit.ly/3HUsU0p>

Android Developer 2020: Snackbars
<https://bit.ly/3gPQxw>

37 MGE | 03 - Android - GUI Programmierung

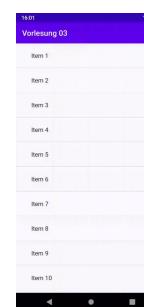
07.10.2022 OST

Dialoge mit erzwungener Benutzereingabe
new AlertDialog(...).show()

Layouts – Teil 2

View Holder-Pattern

- Es gibt noch zwei weitere, teure Operationen in unserem Beispiel
- Effizienter wäre es, diese Objekt-Referenzen pro erzeugte View zu speichern
- Genau dies ist die Idee des **View Holder-Pattern**



07.10.2022 OST

```
Context context = getContext();
LayoutInflater inflater = LayoutInflater.from(context);
view = inflater.inflate(
    android.R.layout.simple_list_item_2,
    parent,
    false
);
```

```
private class ViewHolder {
    TextView text1;
    TextView text2;
}

@Override
public View getView(int pos, View view, ViewGroup parent) {
    ViewHolder viewHolder;
    if (view == null) {
        view = R.layout.simple_list_item_2;
        viewHolder = new ViewHolder();
        viewHolder.text1 = view.findViewById(android.R.id.text1);
        viewHolder.text2 = view.findViewById(android.R.id.text2);
    } else {
        viewHolder = (ViewHolder)view.getTag();
    }
    User user = getItem(pos);
    viewHolder.text1.setText(user.name);
    viewHolder.text2.setText(user.age + " Jahre");
    return view;
}
```

UsersAdapter.java

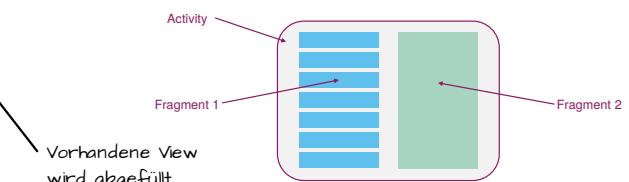
07.10.2022 OST

Wir verwenden Fragments aus AndroidX (Die Normalen sind "deprecated")
Erlaubt zB. das Darstellen von mehreren Views bei mehr Platz (Tablet)
Vorteile von Fragments: Wiederverwendbarkeit, Wartbarkeit, Flexibilität

Strukturierung

Fragments

- Activities können nicht kombiniert werden – dafür gibt es **Fragments**
- Ein **Fragment** ist ein modularer Teil einer Activity mit eigenem Lebenszyklus
- Werden in Activity-Layout eingebunden, auch mehrfach



7 MGE | 04 - Android - Strukturierung, Styling und Material Design

14.10.2022

OST

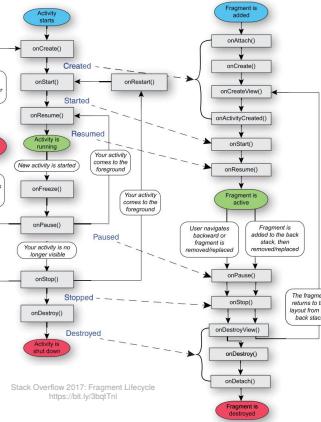
Strukturierung

Fragment Lifecycle

- Ähnlich wie Activity, aber umfangreicher
- Zusätzliche Callbacks gegenüber Activity
 - onAttach:** Fragment an Activity angehängt
 - onCreateView:** UI des Fragments erstellen
 - onActivityCreated:** Activity wurde erzeugt
 - onDestroyView:** Gegenstück zu onCreateView
 - onDetach:** Gegenstück zu onAttach

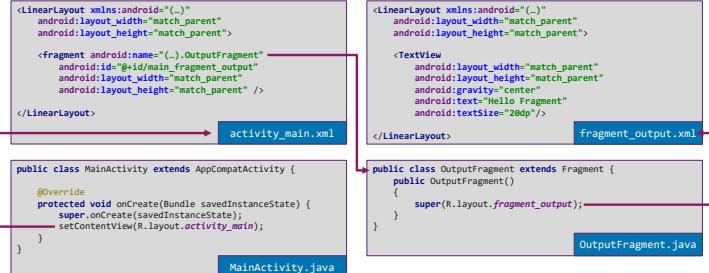
Parameter als **Bundle** übergeben, nie als Konstruktor-Parameter (Slide 14)

9 MGE | 04 - Android - Strukturierung, Styling und Material Design



Strukturierung

Beispiel – Statische Einbindung



10 MGE | 04 - Android - Strukturierung, Styling und Material Design

Strukturierung

Beispiel – Dynamische Einbindung



14.10.2022

Austausch von Fragments (mit Animation)

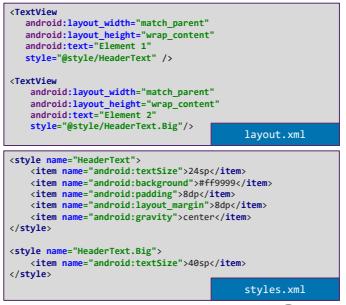
```
fragmentManager.beginTransaction()
    .setCustomAnimations(
        R.anim.slide_in, // Einblenden neues Fragment
        R.anim.fade_out, // Ausblenden altes Fragment
        R.anim.fade_in, // Einblenden altes Fragment (Pop)
        R.anim.slide_out) // Ausblenden neues Fragment (Pop)
    .replace(R.id.main_fragment_container, newFragment)
    .addToBackStack(null)
    .commit();
```

OST

Styling

Themes

- Themes sind spezielle Styles, die für eine ganze App oder einzelne Activities gelten
- Definition wie normale Styles in Resources **res/values/styles.xml**
- Überschreiben von vordefinierten **Schlüsseln**
 - Tipp: SDK-Quellcode nutzen ([Link](#))
- Unterschiedliche Themes als «Parent» (mehr dazu im Block «Material Design»)
- Variante 1: Manifest
 - Für die ganze App (**application**)
 - Für einzelne Activities (**activity**)



14.10.2022

OST

Gilt für alle TextViews



- Variante 2: Activity-Code
 - Innerhalb von **onCreate()**
 - Wichtig:** vor **setContentView()**



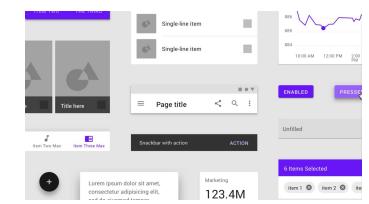
AndroidManifest.xml

MainActivity.java

Material Design

Material Design

- Design Language von Google
- Seit 2014 stetig weiterentwickelt
- Eine umfangreiche Sammlung von ...
 - Prinzipien
 - Richtlinien
 - Empfehlungen
 - Icons**
 - Beispielen**
 - Komponenten** (Libraries)
- Optimiert für Mobile und Web

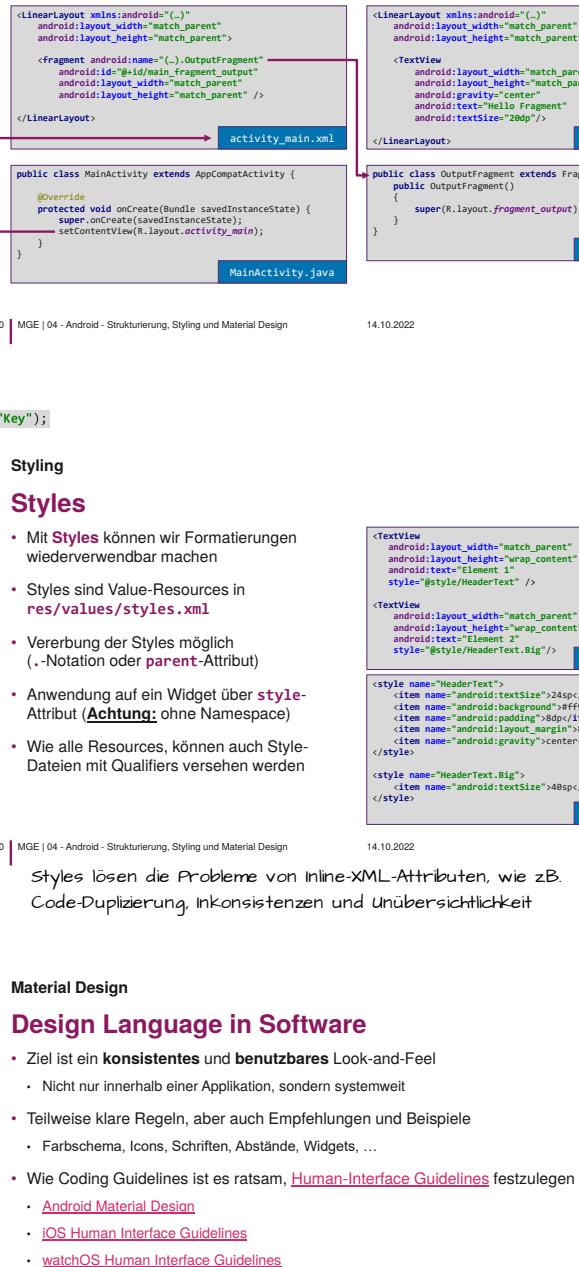


14.10.2022

OST

Strukturierung

Material Design



Design Language: Hilfestellung für den Designprozess.
Beschreibt, wie die Teile einer Applikation aussehen und sich verhalten sollen.

26 MGE | 04 - Android - Strukturierung, Styling und Material Design

14.10.2022

OST

27 MGE | 04 - Android - Strukturierung, Styling und Material Design

14.10.2022

OST

Höhe und Schatten der Elemente klar definiert.

Material Design

Grundprinzipien

• Material is the metaphor

- Inspiriert von der physischen Welt
- Oberflächen erinnern an Papier und Tinte
- Materialien reflektieren Licht & werfen Schatten

• Bold, graphic, intentional

- Basiert auf Prinzipien von Print-Medien
- Hierarchie, Raster, Schriften, Farben, ...

• Motion provides meaning

- Bewegung bedeutet Aktion
- Zurückhaltende, subtile Verwendung

Flach (1 dp), unendliche Auflösung

Guidelines

- Basierend auf den Prinzipien existieren Richtlinien für
 - Layout
 - Navigation (Types, Transitions, Search, ...)
 - Color
 - Typography
 - Sound
 - Iconography
 - Shape
 - Motion
 - Interaction (Gestures, Selection, States)
 - Communication (Empty States, Onboarding, Offline, ...)
 - Usability (Accessibility, Bidirectionality)

Material Design

Farben

- Es gibt eine **Primär-** und eine optionale **Sekundärfarbe** mit Abstufungen
 - Die Primärfarbe wird am häufigsten verwendet
 - Die Sekundärfarbe dient der Hervorhebung
- Tools zur Farbwahl
 - 2014 Material Design Color Palettes
 - Material Palette Generator
 - Color Tool



Material Design 2020: Color
<https://bit.ly/3bgD0de>

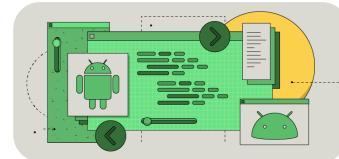
Elemente sind veränderbar, aber nicht "zerreissbar".

Material Design "Components" sind vordefinierte GUI-Elemente

Material Components Library

• Mittels Components Library können wir die Material Design-Widgets verwenden

- Bottom Navigation
- Buttons
- Cards
- Chips
- FABs
- Snackbars
- Tabs
- ... und mehr ...



Material Design 2020: Components Library
<https://bit.ly/3K9Vc>

Die Library enthält Alternativen zu den normalen SDK-Controls (z.B. Switch)

Berechtigungen

Manifest

• Alle benötigten Berechtigungen müssen im Manifest deklariert werden

- Knoten `<uses-permission>`

• Obergrenze für API-Level definierbar

• Hinweise auf benötigte Features für Filtering im Google Play Store

- Optionaler Knoten `<uses-feature>`

• **required="true"** Notwendige Hardware (Standard-Wert falls leer)

• **required="false"** Empfohlene Hardware

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
    <!-- Berechtigungen -->
    <uses-permission
        android:name="android.permission.CALL_PHONE"/>
    <uses-permission
        android:name="android.permission.CAMERA"/>
    <uses-permission
        android:name="android.permission.WRITE_EXTERNAL_STORAGE"
        android:maxSdkVersion="28" />
    <!-- Feature-Hinweise -->
    <uses-feature
        android:name="android.hardware.location" />
    <uses-feature
        android:name="android.hardware.camera"
        android:required="true" />
    <uses-feature
        android:name="android.hardware.bluetooth"
        android:required="false" />
</manifest>
```

Mechanismen im Überblick

• App-spezifische Dateien

- Beliebige Dateiformate
- Für app-internen Gebrauch
- Beispiel: Domänenobjekte als JSON-serialisiert

• Preferences

- Key-Value-Paare
- Für app-internen Gebrauch
- Beispiel: Einstellungen des Benutzers

• Datenbanken

- Strukturierte Daten
- Für app-internen Gebrauch
- Beispiel: Umfangreiche Domänenobjekte

• Medien

- Bilder, Dokumente, Musik und Videos
- Austausch mit anderen Apps ohne UI-Dialog (benötigt ggf. Berechtigungen)
- Beispiel: Von der App aufgenommene Fotos

• Dokumente

- Beliebige Dateiformate
- Austausch mit anderen Apps mit UI-Dialogen (benötigt keine Berechtigungen, da durch User gesteuert)
- Beispiel: Von der App erzeugte PDFs



```
private static final int CALLBACK_CODE = 1;
String permission = Manifest.permission.CALL_PHONE;
// Aktuellen Status prüfen (Android)
int status = ContextCompat.checkSelfPermission(
    this,
    permission);
if (status != PackageManager.PERMISSION_GRANTED) {
    if (shouldShowRequestPermissionRationale(permission)) {
        // Erklärung für Benutzer anzeigen (wozu nötig)
        // liefert true nach erstmaliger Verweigerung
    }
    // Berechtigung beim Benutzer anfordern
    requestPermissions(
        new String[]{permission},
        CALLBACK_CODE);
}
>MainActivity.java
```

MainActivity.java

```
@Override
public void onRequestPermissionsResult(
    int requestCode, String[] permissions, int[] results) {
    if (requestCode != CALLBACK_CODE)
        return;
    if (results.length == 8)
        return; // Anfrage abgebrochen
    if (results[0] == PackageManager.PERMISSION_GRANTED)
        // Berechtigung erteilt
    else
        // Berechtigung verwiegt
}
>MainActivity.java
```

Die Arrays ermöglichen die Anforderung mehrerer Berechtigungen gleichzeitig

MainActivity.java

Material Design

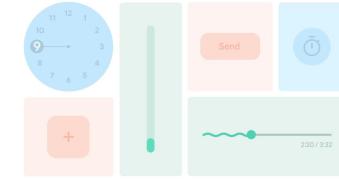
«Material You» in Android 12+

• Weiterentwicklung von Material Design

- Mehr Farben
- Mehr Animationen
- Mehr abgerundete Ecken (mal wieder ☺)
- Mehr Raum für Individualisierung (**You**)

• Einführung mit **Android 12**

- Zuerst für Pixel-Smartphones
- Später für weitere Hersteller



Material Design 2021: Unveiling Material You

<https://bit.ly/3yUfjsg>



Persistenz

Speicherarten

• Interner Speicher

- Stets verfügbar
- Geschützter Speicherbereich pro App
- Speicherplatz begrenzt
- Für app-interne Daten

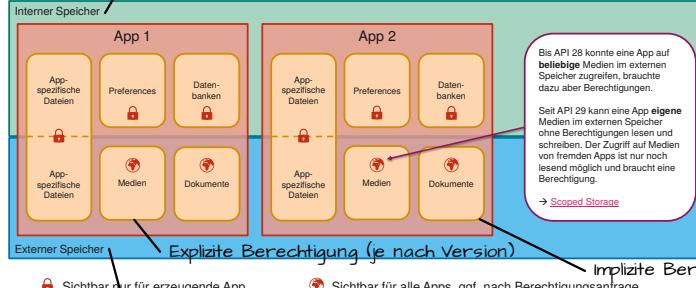
• Externen Speicher

- Nicht immer verfügbar
- Oft ein Wechseldatenträger
- Emulation durch Android möglich
- Speicherplatz begrenzt (aber meist größer)
- Primär für mit anderen Apps geteilte Daten

Persistenz

Werden bei der Deinstallation der App gelöscht.
Keine Berechtigungen nötig.

Mapping von Mechanismen zu Speicherarten



15 | MGE | 05 - Android - Berechtigungen, Persistenz und Hardwarezugriff

21.10.2022



Sind auch nach der Deinstallation weiter verfügbar.
Berechtigungen je nach Funktion nötig.

=> App-Dateien lassen sich debuggen und auch exportieren.

Hardwarezugriff

Sensor Framework – Typen

Sensor	Type	Common Uses
TYPE_ACCELEROMETER	Hardware	Motion detection (shake, tilt, etc.)
TYPE_AMBIENT_TEMPERATURE	Hardware	Monitoring air temperatures
TYPE_GRAVITY	Software or Hardware	Motion detection (shake, tilt, etc.)
TYPE_GYROSCOPE	Hardware	Rotation detection (spin, turn, etc.)
TYPE_LIGHT	Hardware	Controlling screen brightness
TYPE_LINEAR_ACCELERATION	Software or Hardware	Monitoring acceleration along a single axis
TYPE_MAGNETIC_FIELD	Hardware	Creating a compass
TYPE_ORIENTATION	Software	Determining device position
TYPE_PRESSURE	Hardware	Monitoring air pressure changes
TYPE_PROXIMITY	Hardware	Phone position during a call
TYPE_RELATIVE_HUMIDITY	Hardware	Monitoring dewpoint, absolute, and relative humidity
TYPE_ROTATION_VECTOR	Software or Hardware	Motion detection and rotation detection
TYPE_TEMPERATURE	Hardware	Monitoring temperatures

34 | MGE | 05 - Android - Berechtigungen, Persistenz und Hardwarezugriff

21.10.2022



Funktionen wie Kamera, Positionsbestimmung und Internet-Verbindungsinformationen verwenden jeweils eine eigene API (CameraX, LocationManager, ConnectivityManager)

=> Diese APIs können sich stark voneinander unterscheiden.

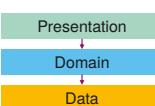
=> Das gilt auch für die unterschiedlichen Kanäle bei der Connectivity. (WiFi, Bluetooth etc.)

Architektur

Schichtenarchitektur

- Presentation**
 - Darstellung und Interaktion mit Benutzer
 - Typischerweise stark an UI-Toolkit gebunden

- Data**
 - Speicherung der Daten
 - Stellt Daten der Domain zur Verfügung
 - Auch Persistenz oder Datenhaltung genannt



Domain

- Businesslogik und Domänenklassen
- Keine UI-Funktionalität
- Wenig externe Abhängigkeiten
- Einfach zu testen

Hardwarezugriff

Grundlagen

- Smartphones besitzen diverse Akteure & Sensoren
 - Display
 - Lautsprecher
 - Vibration
 - Kamera
 - Beschleunigung
 - Licht
 - Lage
 - ...
- Die Qualität der verbauten Hardware variiert stark

31 | MGE | 05 - Android - Berechtigungen, Persistenz und Hardwarezugriff

Sensor Framework

- Framework für verschiedene Sensoren
 - Bewegung
 - Umgebung
 - Lage
- Alle Sensoren werden gleich verwendet
 - SensorManager** als Einstiegspunkt
 - Sensor** als Repräsentant für realen Sensor
 - SensorEvent** enthält Werte des Sensors
 - SensorEventListener** für Callbacks

21.10.2022



```
okHttpClient.newCall(request).execute();
retrofit.create(Students.class).getStudents().execute();
```

Hardwarezugriff

REST-Kommunikation

- Für "einfache" REST-Calls existieren (zu) viele Varianten
 - Berechtigung **INTERNET** zwingend für alle
- HttpsURLConnection** ist Teil der Android SDK
 - Verfügbar seit API 1
 - Verwendung eher kompliziert ([Beispiel](#))
- OkHttp** ist eine oft verwendete, effiziente Alternative
 - Verfügbar ab API 21
 - 3rd Party Library
- Retrofit** erlaubt die Definition von Endpunkten & Datenobjekten
 - Verwendet **OkHttp** zur Kommunikation
 - Unterschiedliche Konverter für Serialisierung (z.B. [Gson](#))
 - 3rd Party Library

37 | MGE | 05 - Android - Berechtigungen, Persistenz und Hardwarezugriff



21.10.2022

Hardwarezugriff

Sensor Framework – Beispiel

- Verzögerung beeinflusst Energieverbrauch
 - SENSOR_DELAY_FASTEST** (0ms)
 - SENSOR_DELAY_GAME** (20ms)
 - SENSOR_DELAY_UI** (60ms)
 - SENSOR_DELAY_NORMAL** (200ms)
- Änderungen der Genauigkeit
 - SENSOR_STATUS_ACCURACY_HIGH**
 - SENSOR_STATUS_ACCURACY_MEDIUM**
 - SENSOR_STATUS_ACCURACY_LOW**
 - SENSOR_STATUS_ACCURACY_UNRELIABLE**
- Inhalten des **values**-Array sind abhängig vom verwendeten Sensor

```
// Bet Sensor auf Änderungen registrieren
String service = Context.SENSOR_SERVICE;
int type = Sensor.TYPE_LIGHT;
int delay = SensorManager.SENSOR_DELAY_NORMAL;

SensorManager mgr = (SensorManager) getSystemService(service);
Sensor sensor = mgr.getDefaultSensor(type);
mgr.registerListener(this, sensor, delay);

// Implementierung von SensorEventListener
@Override
public void onSensorChanged(SensorEvent sensorEvent) {
    float lux = sensorEvent.values[0];
    Log.d("null", lux + " lux");
}

@Override
public void onAccuracyChanged(Sensor sensor, int i) {
}
```

33 | MGE | 05 - Android - Berechtigungen, Persistenz und Hardwarezugriff

21.10.2022

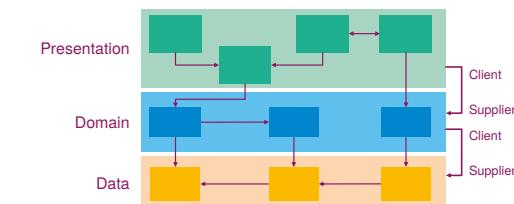


Ziel von solchen Architekturen: Verbesserung der Verständlichkeit und Wartbarkeit von Software-Systemen.

Architektur

Schichtenarchitektur

- Häufig angewendete Strukturierungshilfe für unsere Software
- Schichten gruppieren **zusammengehörige Konzepte**
- Keine Zyklen**: Abhängigkeit nur auf darunterliegende Schicht(en)



7 | MGE | 06 - Android - Architektur und fortgeschritten Themen

28.10.2022



Varianten davon:

- Zusätzliche Schichten
- Vertikale Zerlegung (siehe [NET MAUI](#))
- Presentation Patterns (MVC, MVVM, MVP, Observer Pattern)

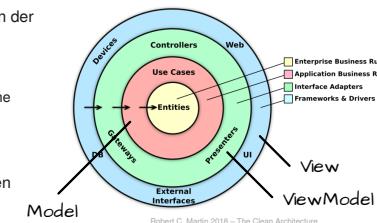
Architektur

Observer Pattern

- Wenn Schichtenarchitekturen keine Zyklen erlauben:
Wie kann **Domain** dann die **Presentation** über Änderungen am internen Zustand informieren?



Beobachtet
(View)



«The observer pattern is a software design pattern in which an object, called the **subject**, maintains a list of its dependents, called **observers**, and notifies them automatically of any state changes, usually by calling one of their methods.»

Der Observer kennt das Subject gut.

Umgekehrt kennt das Subject nur das Interface des Observers.



8 | MGE | 06 - Architektur und fortgeschritten Themen

28.10.2022



10 | MGE | 06 - Android - Architektur und fortgeschritten Themen

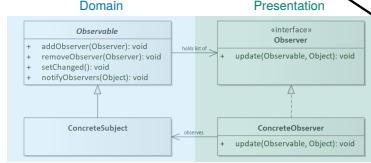
28.10.2022



Architektur

Observer Pattern

- Subject ist eine Ableitung der abstrakten Klasse **Observable**
- Observer implementieren das Interface **Observer**
- **notify()** in zwei Schritten: `setChanged()` und `notifyObservers()`
- **update()** erhält als Parameter das Subject und den Parameter von `notifyObservers()`



Subjekt sagt, dass es sich verändert hat.

Observer reagieren auf die Veränderung.

14 MGE | 06 - Android - Architektur und fortgeschrittene Themen

28.10.2022

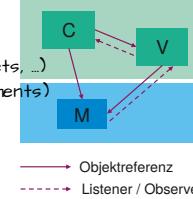


Probleme: Bei grösseren Modellen geht die Übersicht schnell verloren.
=> Wer observiert wen? Wurden alle Observer korrekt abgemeldet? etc.

Architektur Android basiert lose auf MVC

Model-View-Controller (MVC)

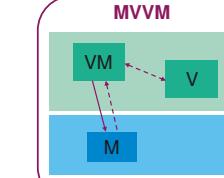
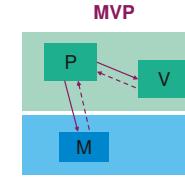
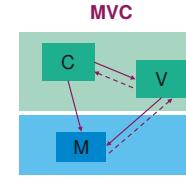
- Ein Pattern für die Organisation der Präsentation
- **Model** beinhaltet die Daten (**Non-UI-Code**)
- **View** liest die Daten des Models und zeigt diese an (**Views, Widgets, ...**)
- **Controller** koordiniert zwischen Model und View (**Activity, Fragments**)
- Entstanden in den ersten Smalltalk GUIs, heute auch Grundlage einiger Web-Frameworks
- Viele Varianten, je nach Umsetzung leichte Unterschiede
- Verwandte Patterns
 - **MVP: Model – View – Presenter (Eigenbau)**
 - **MVVM: Model – View – View Model (AndroidX / Android Jetpack)**



→ Objektreferenz
→ Listener / Observer / Bindings

Architektur

MVC vs. MVP vs. MVVM



Sehr relevant für Android & .NET MAUI

17 MGE | 06 - Android - Architektur und fortgeschrittene Themen

28.10.2022



17 MGE | 06 - Android - Architektur und fortgeschrittene Themen

28.10.2022



19 MGE | 06 - Android - Architektur und fortgeschrittene Themen

28.10.2022



Fortgeschrittene Themen

Application

- Der Eltern-Knoten unserer Komponenten im Manifest heisst **application**
- Zu diesem Knoten wird beim Start eine Instanz der **Application**-Klasse erzeugt
- Eigene Ableitungen der Klasse sind möglich
 - Einmalige Initialisierungen
 - Erzeugung von Singleton-Objekten
 - Zugriff auf globale Objekte
- **Lifecycle-Methoden** werden von Android zu bestimmten Zeitpunkten aufgerufen

AndroidManifest.xml:

```

<manifest xmlns:android="..."
  package="ch.ost.rj.mge.v06.myapplication"

  <application android:name=".MyApplication">
    </application>
  </manifest>
  
```

MyApplication.java:

```

public class MyApplication extends Application {
    @Override
    public void onCreate() {
        super.onCreate();
    }

    // ... mehr Callbacks auf der nächsten Slide...
}
  
```

28.10.2022



Fortgeschrittene Themen

Application Context

- **Context**-Objekte haben eine eingeschränkte Lebensdauer
 - Angeforderte Ressourcen werden mit dem zugehörigen Context freigegeben
 - Beispiel: Activity-Context via `this`
- Das **Application Context**-Objekt ist während der ganzen App-Lebensdauer gültig
 - Abholen via `Context.getApplicationContext()`
 - Eingeschränkt verwendbar für UI-Aktionen
 - Vorsicht vor **Memory Leaks**

	Application	Activity	Service	ContentProvider	BroadcastReceiver
Show a Dialog	NO	YES	NO	NO	NO
Start an Activity	NO ¹	YES	NO ¹	NO ¹	NO ¹
Layout Inflation	NO ¹	YES	YES	NO ¹	NO ¹
Start a Service	YES	YES	YES	YES	YES
Bind to a Service	YES	YES	YES	YES	NO
Send a Broadcast	YES	YES	YES	YES	YES
Register BroadcastReceiver	YES	YES	YES	YES	NO ¹
Load Resource Values	YES	YES	YES	YES	YES

Je nach "Context" sind andere Funktionen möglich / erlaubt (siehe Tabelle).

25 MGE | 06 - Android - Architektur und fortgeschrittene Themen

28.10.2022

Fortgeschrittene Themen

Broadcasts

- Austausch von Meldungen zwischen Apps
 - Variante des **Publish-Subscribe-Patterns**
- Zwei Arten von Broadcasts
 - **Lokal**: innerhalb einer App
 - **Global**: innerhalb des ganzen Systems
- Android sendet selbst globale Broadcasts
 - System wurde gestartet
 - Netzwerkverbindung verloren
 - SMS empfangen
 - ... und mehr ...

27 MGE | 06 - Android - Architektur und fortgeschrittene Themen

Fortgeschrittene Themen

Broadcasts – Best Practices

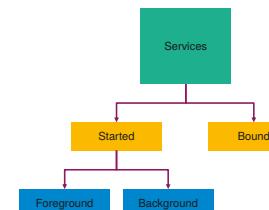
- Gemeinsame Einführung
- Lokale Broadcasts bevorzugen
- Keine sensiblen Daten in Broadcasts übermitteln
- App ID in eigene Broadcast-Actions integrieren
- Selbststudium
 - Dynamische Registrierung bevorzugen
 - Schnelle Rückkehr aus `onReceive()`
 - Ausführung auf Main-Thread
 - Besser: Ausführung auf Background-Thread
 - Falls App nicht aktiv: keine Activity starten
 - Besser: Notification anzeigen

28.10.2022



29 MGE | 06 - Android - Architektur und fortgeschrittene Themen

Ausserhalb der App



Threads entkoppeln Aufgaben vom UI-Thread, Services von einer Activity



28.10.2022

Fortgeschritten Themen

Started Services

- Für einmalige Aktionen
- Laufen potentiell endlos weiter
 - Beendigung durch Service selbst: `stopSelf()`
 - Beendigung durch eine Applikation: `stopService()`
 - Beendigung durch Android
- Spezialvarianten
 - `IntentService` für Ausführung in Background-Thread und automatischem Stop
 - `JobIntentService` als modernere Alternative für `IntentService`
 - `Foreground Services` mit Notifications als UI

Broadcasts:

```
// Registrierung
BroadcastReceiver receiver = new My BroadcastReceiver();
String action = ConnectivityManager.CONNECTIVITY_ACTION;
IntentFilter filter = new IntentFilter(action);
registerReceiver(receiver, filter);

// Abmeldung
unregisterReceiver(receiver);
```

onReceive()Funktion

```
// Impliziter Broadcast (für dyn. Registrierung)
Intent intent = new Intent(this, MyStartedService.class);
// Start
startService(intent);
// Stop
stopService(intent);
```

Android Developer 2020 - Services Lifecycle
<https://bit.ly/2PpDvu>

The service is stopped by itself or a client

Service running

The service is stopped by its self or a client

onDestroy()

Service shut down

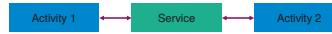
Unbound service

Android Developer 2020 - Services Lifecycle
<https://bit.ly/2PpDvu>

Fortgeschritten Themen

Bound Services

- Für Aufgaben über längere Zeitdauer
- Client-Server ähnliche Kommunikation
 - Innerhalb eines Apps: via Interface
 - App-übergreifend: via `Messenger`-Klasse
- Austausch von Daten fester Bestandteil
- Mehrere Clients gleichzeitig möglich
- Nach Verbindungsende zu letztem Client wird der Service automatisch gestoppt



Services:

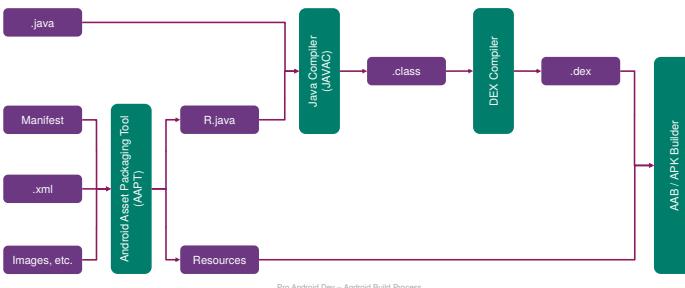
```
public class MyStartedService extends Service {
    @Override
    public void onCreate() {
        super.onCreate();
    }

    @Override
    public int onStartCommand(Intent i, int flags, int id) {
        new Thread(() -> {
            // Aktion im Hintergrund
        }).start();

        return START_NOT_STICKY;
    }
}
```

Build & Deployment

Build – Erzeugung AAB / APK



Pro Android Dev - Android Build Process
<https://bit.ly/2QqfPer>

47 MGE | 06 - Android - Architektur und fortgeschritten Themen

28.10.2022



=> "dex"-Code läuft wie Java-Code auf einer VM (Android Runtime)

Architecture Components – Teil 1

View Binding

- Vereinfacht den Zugriff auf View-Elemente
 - Kein `findViewById()` mehr
 - Typ- und Null-Sicherheit
- Erzeugung von `Binding`-Klassen beim Build
 - Aktivierung über Gradle
 - Deaktivierung für einzelne Layouts möglich
- Namensgebung der Klassen
 - Layout-Name als `Camel Case + Binding`
 - `activity_main.xml` → `ActivityMainBinding`

(Standard in .NET MAUI)

```
android {
    buildFeatures {
        viewBinding true
    }
} build.gradle

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/button_hello"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Hello World!" />
</LinearLayout> activity_main.xml

public class MainActivity extends AppCompatActivity {
    private ActivityMainBinding binding;

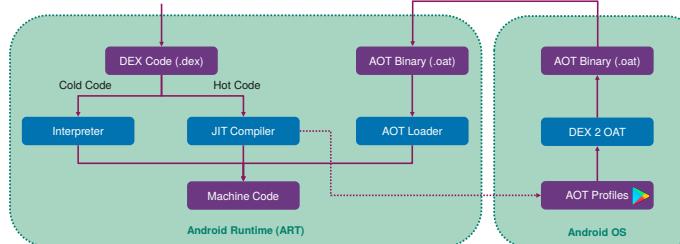
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        LayoutInflator inflater = getLayoutInflater();
        binding = ActivityMainBinding.inflate(inflater);
        setContentView(binding.getRoot());
        binding.buttonHello.setOnClickListener(v -> {});
    }
} MainActivity.java
```

04.11.2022



12 MGE | 07 - Android - Android Jetpack

Build – ART 2.0 (Android >= 5.0)



Pro Android Dev - Android Build Process
<https://bit.ly/2QqfPer>

50

MGE | 06 - Android - Architektur und fortgeschritten Themen

28.10.2022



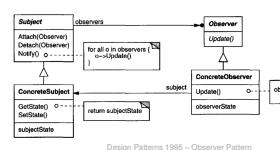
Verbesserung der Runtime: JIT-Code wird auf einem Gerät ausgeführt und dann als AOT-Code im Play Store gespeichert.

- Vorteil 1: Kein JIT mehr nötig nach erstmaliger Ausführung (Performance)
- Vorteil 2: Kein AOT bei Installation & Upgrade des Geräts nötig.

Architecture Components – Teil 1

Data Binding

- Erlaubt im XML den Zugriff auf Objekte
 - Layout als Observer der Daten
 - Einfache Logik direkt im XML
- Basiert ebenfalls auf `Binding`-Klassen
 - Aktivierung über Gradle
 - Generierung beim Build
- Unterschied zu View Binding
 - `View Binding`: vereinfacht Zugriff auf View
 - `Data Binding`: vereinfacht Zugriff auf Daten



```
android {
    buildFeatures {
        dataBinding true
    }
} build.gradle
```

Verwendung der "Expression Language" (= Markup Language .NET MAUI)

13

MGE | 07 - Android - Android Jetpack

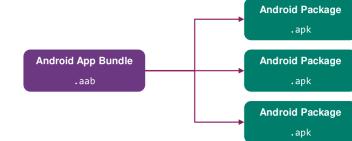
04.11.2022

Build & Deployment

Deployment – APK (Android Package)

- Apps werden aus `APK-Dateien` installiert
 - ZIP-Archiv mit eigener Dateiendung
 - Kann über beliebige Kanäle verteilt werden
 - Enthält alle zur Ausführung nötigen Daten
- APKs aus dem Play Store sind signiert
 - Verifizierung des Herausgebers
 - Privaten Schlüssel sicher aufbewahren
 - Andere APKs gelten als "unbekannte Apps"
- `APK Analyzer` in Android Studio

Verschiedene APKs, z.B.
für 64 und 32Bit.



Es gilt: Pro APK einen eigenen Linux-User und Linux-Prozess.

Limit im Play Store: 100MB => APKs aufteilen oder Bilder, etc. in OBB-Archive.

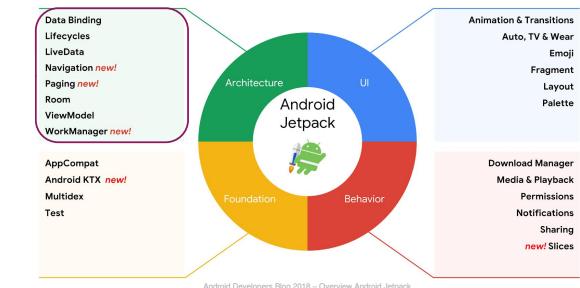
AAB: Container mit App-Daten. APKs werden nach Bedarf erstellt.

=> Konsequenz von AABs: Google im Besitz der APK-Signaturschlüsseln.

=> Seit 2021 sind AABs zwingend im Play Store (Limit 150MB)

Einführung

Komponenten im Überblick (2018)



Android Developers Blog 2018 - Overview Android Jetpack
<https://bit.ly/3B1P1>

7 MGE | 07 - Android - Android Jetpack

04.11.2022

=> Aktuell existieren etwa 90 Jetpack-Libraries!

=> zB. AppCompatActivity, ConstraintLayout, RecyclerView, etc.
=> Eine Komponente ist nicht zwingend eine eigene Library

Architecture Components – Teil 1

Data Binding – Beispiel

```
public class User {
    public String firstName;
    public String lastName;

    public User(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
} User.java

public class MainActivity extends AppCompatActivity {
    private ActivityMainBinding binding;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        LayoutInflater inflater = getLayoutInflater();
        binding = ActivityMainBinding.inflate(inflater);
        setContentView(binding.getRoot());
        binding.buttonHello.setOnClickListener(v -> {
            User user = new User("Thomas", "Kilian");
            binding.setUser(user);
        });
    }
} MainActivity.java

<layout xmlns:android="...">
    <data>
        <variable name="user" type="ch.ost.rj.mge.v07.User"/>
    </data>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        <Textview
            android:id="@+id/first"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@{user.firstName}" />
        <Textview
            android:id="@+id/last"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@{user.lastName}" />
    </LinearLayout>
</layout>
```

14 MGE | 07 - Android - Android Jetpack

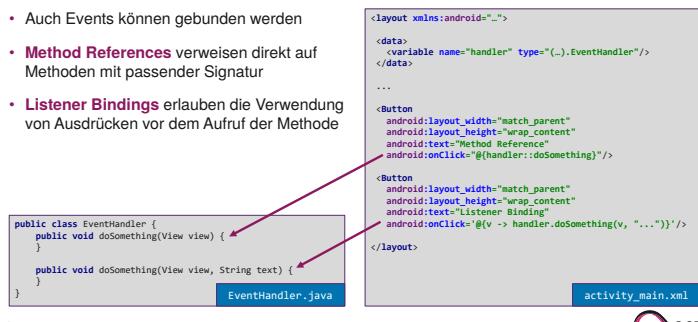
04.11.2022

OST

Architecture Components – Teil 1

Data Binding – Event Handling

- Auch Events können gebunden werden
- Method References** verweisen direkt auf Methoden mit passender Signatur
- Listener Bindings** erlauben die Verwendung von Ausdrücken vor dem Aufruf der Methode



16 MGE | 07 - Android - Android Jetpack

04.11.2022

OST

Data-Binding Vorteile: Schlanke Activities, bessere Testbarkeit, ermöglicht MVVM-Implementierung

Nachteile: Model wird verunreinigt (ohne MVVM), erhöhter Zeitbedarf bei Kompilierung, schwereres Debugging

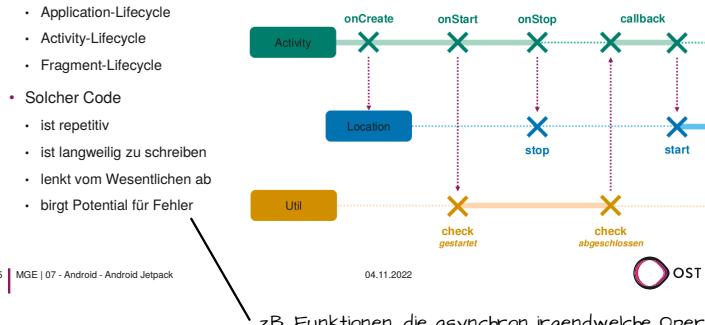
Problem 1: Unsichtbare Observers, also Updates am GUL obwohl es nicht sichtbar ist. (Lösung: Lifecycle-Aware Components)

Problem 2: Bei Rotation des Geräts wird die View und somit die Daten neu erzeugt. (Lösung: ViewModel von Androidx)

Architecture Components – Teil 2

Lifecycle Methoden – Once again...

- Häufig beschäftigt sich unser Code damit, auf **Zustandsänderungen** zu reagieren



25 MGE | 07 - Android - Android Jetpack

04.11.2022

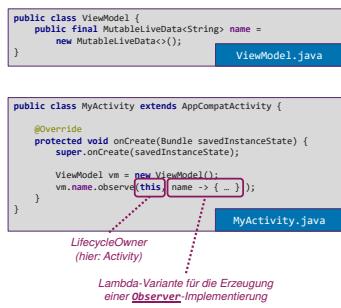
OST

z.B. Funktionen, die asynchron irgendwelche Operationen ausführen.

Architecture Components – Teil 2

LiveData

- LiveData** ist ein lifecycle-aware Observable
- Alternative zu **Observable Fields** und **Observable Classes**
- Als Datenquelle für **Data Bindings** verwendbar
- Observer werden ...
 - benachrichtigt, sofern diese aktiv sind (**STARTED, RESUMED**)
 - entfernt, wenn diese gestoppt wurden (**DESTROYED**)



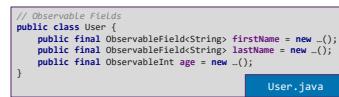
04.11.2022

OST

Architecture Components – Teil 1

Data Binding – Änderungen an Daten

- Ein in Data Bindings verwendetes Objekt wird **nicht** automatisch observierbar
- Für die automatische Aktualisierung der View muss die Datquelle angepasst werden
 - Observable Fields** für einzelne Werte
 - Observable Classes** für ganze Klassen
- Normale Java-Observable sind inkompatibel



17 MGE | 07 - Android - Android Jetpack

04.11.2022

Variante des Observer-Patterns

```
// Observable Class
public class User extends BaseObservable {
    private String firstName;
    private String lastName;
    private int age;

    @Bindable
    public String getFirstName() { return this.firstName; }

    @Bindable
    public String getLastname() { return this.lastName; }

    @Bindable
    public int getAge() { return this.age; }

    public void setLastName(String lastName) {
        this.lastName = lastName;
        notifyPropertyChanged(BR.lastName);
    }

    public void setAge(int age) {
        this.age = age;
        notifyPropertyChanged(BR.age);
    }
}
```

User.java

Architecture Components – Teil 1

Data Binding – Two-Way Bindings

- Verschiedene XML-Attribute unterstützen diese Variante von Haus aus
- Notation: `=` vor der Binding Expression
 - `android:value="@={user.age}"`
 - Eselstrüke: Zwei Linien für Zwei Wege
- Eigene Two-Way-Bindings sind möglich
 - [Siehe Anleitung](#)
- Vorsicht vor Endlos-Schleifen

One-Way: Model → View, View → Model

Two-Way: Model <→ View

19 MGE | 07 - Android - Android Jetpack

04.11.2022

OST

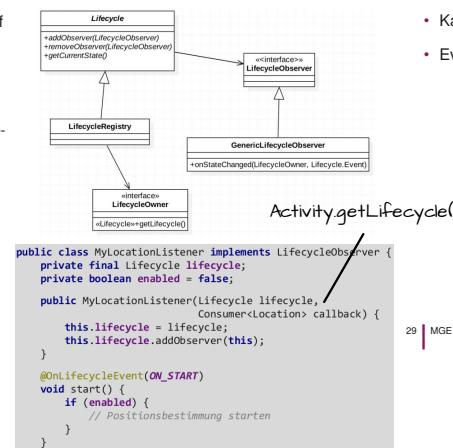
Android Developer 2020: Data Binding - Two-Way Bindings

<https://bit.ly/3Fk5yA>

Architecture Components – Teil 2

Klasse Lifecycle

- Kapselung des Zustands (Enumeration `Lifecycle.State`)
- Events an registrierte Observer (Enumeration `Lifecycle.Event`)



Activity.getLifecycle(),

29 MGE | 07 - Android - Android Jetpack

04.11.2022

Android Developer 2020: Lifecycle Class

<https://bit.ly/32EDKTP>

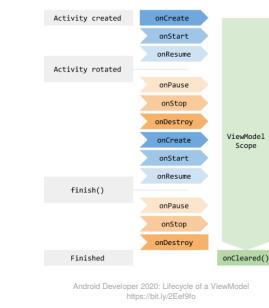
observiert Zustandsänderungen von Lifecycle Owner



Architecture Components – Teil 3

ViewModel

- Die abstrakte Klasse `ViewModel` löst unser letztes Problem: die **Rotation des Gerätes**
- `ViewModel`-Objekte hängen von `Lifecycle`-Objekten ab
 - Mehrfache Erzeugung des `ViewModel` mit demselben `Lifecycle`-Objekt liefert Singleton
 - Erst bei Zerstörung des `Lifecycle`-Objektes wird auch das `ViewModel`-Objekt freigegeben
- Ein App-Lebenszyklus geknüpfte View Models sind möglich (via `Application`-Objekt)
- Basisklasse `AndroidViewModel`



35 MGE | 07 - Android - Android Jetpack

04.11.2022

OST

Architecture Components – Teil 2

MVVM im Eigenbau



Setzen des LifecycleOwner für die Data Bindings im XML

32 MGE | 07 - Android - Android Jetpack

04.11.2022

OST