Lucerne University of
Applied Sciences and Arts
HOCHSCHULE
LUZERN
Informatik

Mobile Programming

Android 1 - Grundlagen

Kaspar von Gunten

Inhalt

- Grundlagen einer Android-Applikation
  - Komponenten
  - Android Manifest
  - Activities und Kommunikation/Navigation mittels Intents
  - Lebenszyklus & Zustände von Apps bzw. Activities
- Das Android Betriebssystem
  - Systemaufbau, Android-Stack & Sicherheitskonzept
- Entwicklungsumgebung
  - SDK: Android Studio inkl. Emulator und HW-Geräte

Wiederholung: Fahrplan Android

- SW1-6, siehe auch Folien «Organisatorisches & Infos»
  - **Android 1**: Grundlagen (Komponenten, Manifest, Ressourcen, Activities, Entwicklungsumgebung)
  - **Android 2**: Benutzerschnittstellen (Ressourcen, Views, Layouts, Adapter, Event-Handling, Dialoge, Notifications)
  - **Android 3**: Persistenz (Preferences, Dateisystem, Datenbank/Room, Content Providers)
  - **Android 4**: Backend-Kommunikation, Nebenläufigkeit und Hintergrundprozesse (API-Calls, AsyncTasks, WorkManager)
  - **Android 5**: Services, Broadcast Receiver
  - **Android 6**: Intents, Widgets, Play Store & der ganze Rest

  - Später ev. noch ein paar Flashtalks zu Themen wie *Testing*, *Build*, *Flutter, Kotlin, ML Kit, etc.*

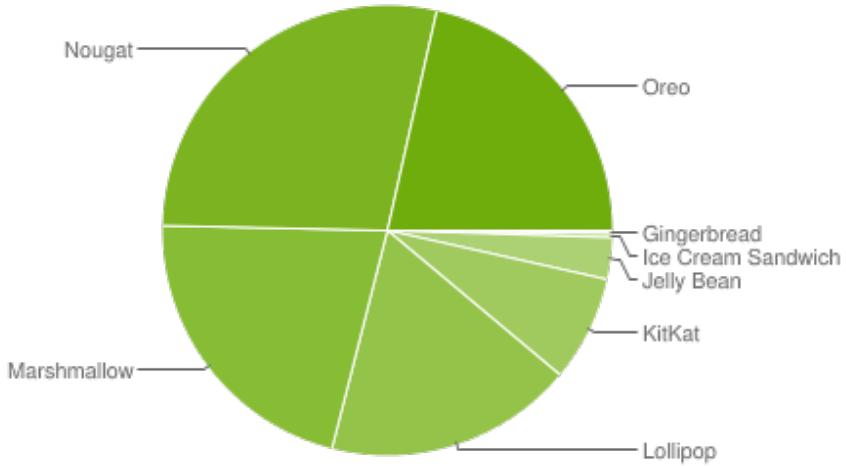
Warum Android?

- Meistgenutzte mobile Plattform der Welt
  - Seit 2017 nur noch iOS ein Konkurrent: 13% vs 87%
- Kompletter Stack: OS, Middleware, Applikationen
- >3.5 Mio Apps bei Google Play (Dez. 2017)[1]
- Offene SW-Plattform für mobile Entwicklung
  - Apache 2 Lizenz
- OHA-Projekt (Open Handset Alliance)
  - OHA != Google, aber von Google geführt
  - 84 Firmen (Februar 2018)[2]

1: http://en.wikipedia.org/wiki/Google_Play
2: http://www.openhandsetalliance.com/oha_faq.html

Android Versionen: Verbreitung

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.2%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.3%
4.1.x	Jelly Bean	16	1.1%
4.2.x		17	1.5%
4.3		18	0.4%
4.4	KitKat	19	7.6%
5.0	Lollipop	21	3.5%
5.1		22	14.4%
6.0	Marshmallow	23	21.3%
7.0	Nougat	24	18.1%
7.1		25	10.1%
8.0	Oreo	26	14.0%
8.1		27	7.5%



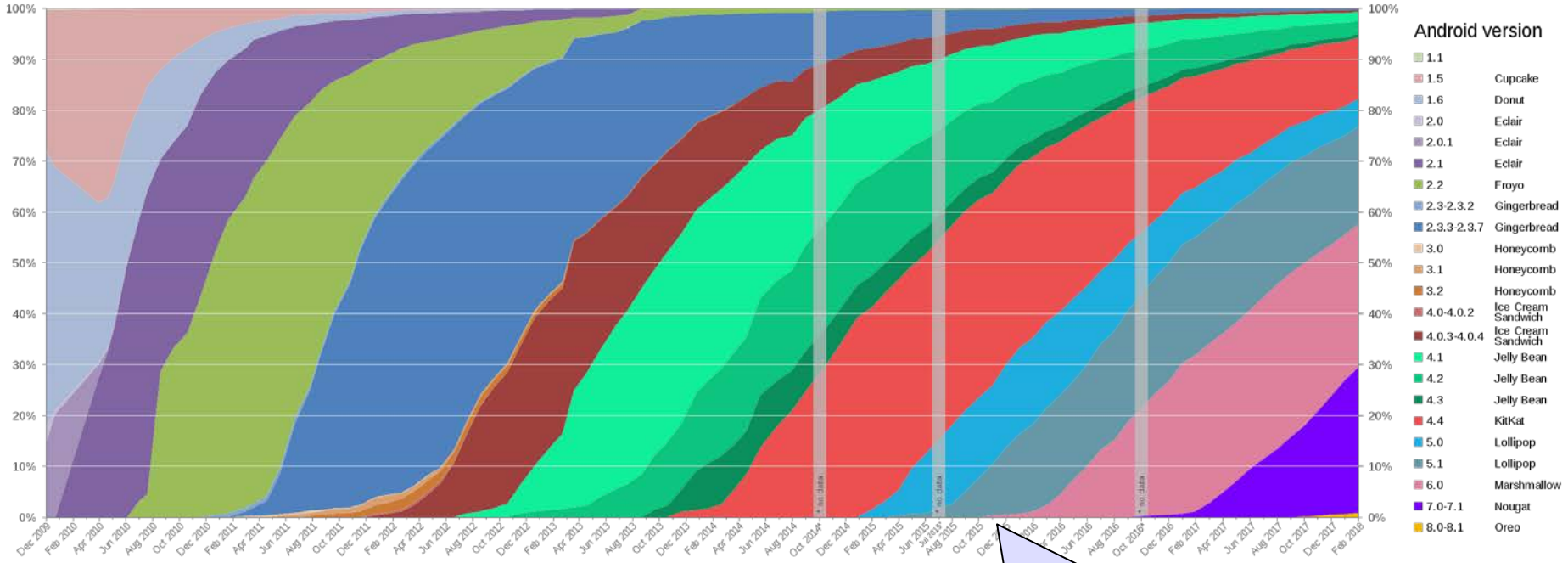
Alte, nicht länger unterstützte Versionen

- Wer hat Android < 7.0?
  - Problem: Hersteller liefern oft keine/späte OS-Aktualisierung, d.h. schleppende Aktualisierung ☹
- Details zu allen Android-Versionen

9.0: noch gar nicht erfasst (Rel. Aug 2018)

Bilder von http://developer.android.com/about/dashboards/
Siehe auch: https://de.wikipedia.org/wiki/Liste_von_Android-Versionen

Android-Versionen: Verbreitungs-Historie



Daten von 2018,
Keine neuere Grafik
verfügbar Feb 2019

Quelle: http://en.wikipedia.org/wiki/Android_version_history (Februar 2018)

Wir entwickeln gegen API 28 „Pie“

Android Versionen:
A, B, C, …, O, P

- Apps sollen grundsätzlich gegen das aktuellste API entwickelt werden
  - D.h. aktuell API Level 28 Android 9 „Pie“
  - Verwenden wir so in diesem Modul
  - Build Skript: `targetSdkVersion 28`

Je nach Projekt: Welche Geräte, Kunden, usw. werden unterstützt?

- Mindestanforderung ans SDK kann im Build-Skript gesetzt werden: `minSdkVersion`
  - Sollte tiefer sein als `targetSdkVersion`, da wie gesehen aktuell kaum Geräte Android 9.x haben...

☞ Siehe später bei den Themen Android Manifest, Build Skript, Android Studio, Android Jetpack

Online Doku von Google

- Android Developer Pages
  - Startseite mit Links auf Design, Develop & Distribute
    - http://developer.android.com/
  - Getting Started
    - http://developer.android.com/training/
  - Intoduction to Android
    - http://developer.android.com/guide/
  - **Package Index (APIs)**
    - http://developer.android.com/reference/packages.html

☞ Viel und generell gute Doku von Google ☺

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

Grundlagen einer Android-Applikation: Komponenten & das Manifest

Android-Apps & Komponenten

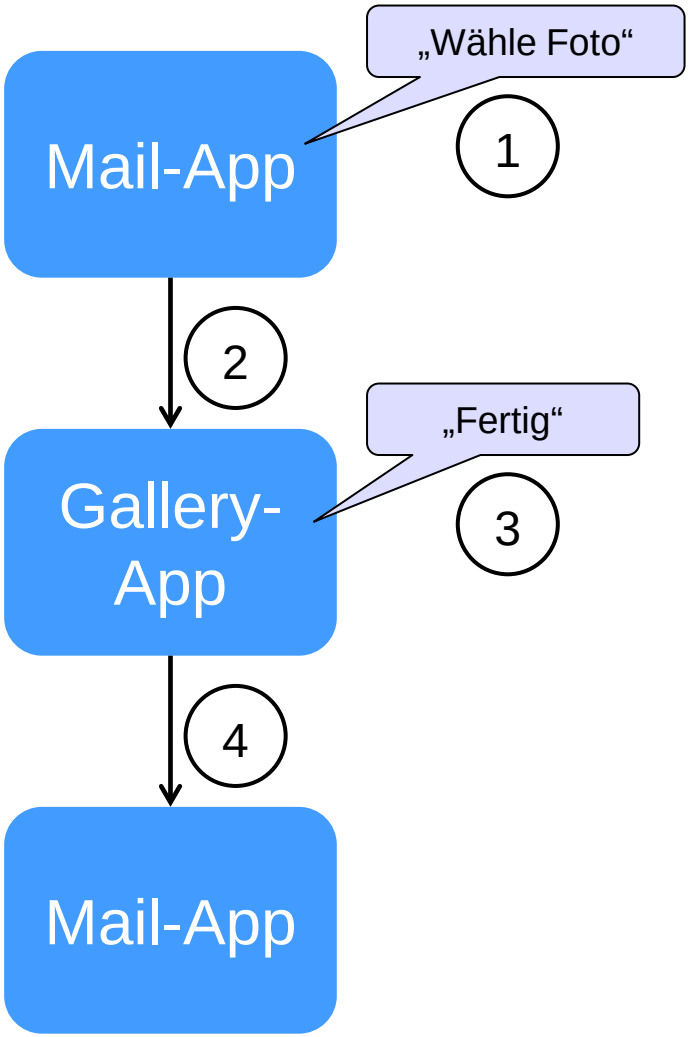
- Android Applikationen sind zusammengesetzt aus **lose gekoppelten Komponenten**
  - Erleichtert Wiederverwendung und Austausch von Komp.
- **Die Android Runtime verwaltet** Applikationen, d.h. die einzelnen Komponenten einer Applikation
  - Mit dem Intent-Mechanismus kann eine Komponente eine andere Komponente aufrufen
  - Komponenten müssen beim System registriert werden (teilweise mit Rechten = Privileges)
  - System verwaltet den Lebenszyklus von Komponenten: Gestartet, Pausiert, Aktiv, Gestoppt, etc.

Wiederverwendbare Komponenten

- Eine Applikation ist aufgebaut aus Komponenten
- App verwendet dabei entweder
  - eigene Komponenten... (sinnvollerweise mind. eine)
  - ...oder Komponenten von anderen, existierenden Applikationen

☞ Siehe Beispiel nächste Folie...

Beispiel: Mail-App will Foto...



1. Mail-App stellt Anfrage für Foto-Auswahl
2. Das System übergibt der passenden Gallery-App
3. Gallery-App übernimmt und meldet wenn fertig
4. Das System übergibt zurück an die Mail-App mit entsprechender Auswahl / Rückgabewert

Android Komponenten: 4 Typen

Name	Beschreibung
Activity	UI-Komponente, entspricht typischerweise einem Bildschirm
Service	Komponente ohne UI, Dienst läuft typsicherweise im Hintergrund
Broadcast Receiver	„Event-Handler“, welche auf App-interne oder systemweite Broadcast-Nachrichten reagieren
Content Provider	Komponente, welche Daten-Austausch zwischen verschiedenen Applikationen ermöglicht

Siehe http://developer.android.com/guide/components/fundamentals.html#Components

Activity: „1 Bildschirm“

- Eine **Activity** entspricht einem Bildschirm (Screen)
  - Stellt UI-Widgets (Label, Knöpfe, ...) dar
  - Reagiert auf Benutzer-Eingabe & -Ereignisse
- Eine App besteht meist aus mehreren Bildschirmen, die auf einem «Stack» liegen
  - Zu jedem Bildschirm gehört eine Activity*
  - Wechsel zu einem nächsten Bildschirm = neue Activity
  - Aufgerufene Activity kann Resultat zur vorangegangen Activity zurückliefern
- **Basisklasse**: `android.app.Activity`

* Ausnahme = Fragments, hier noch kein Thema

Service: „Hintergrund-Dienst ohne GUI“

- Ein **Service** läuft typischerweise im Hintergrund und für für unbeschränkte Zeit. Beispiele: Musik-Spieler, Internet-Download, GPS-Tracker, ...
- Service hat keine graphische Benutzerschnittstelle (UI)
  - UI für einen Service (wenn es eines gibt) wird immer von einer Activity dargestellt!
  - z.B.: *MusicPlayerActivity* für *MusicPlayerService*
- **Basisklasse**: `android.app.Service`

☞ Mehr zu Services später im Modul

Broadcast Receiver: „Handler für (System-)Nachrichten“

- Ein **Broadcast Receiver** ist eine Komponente, welche Broadcast-Nachrichten empfängt und darauf reagiert
  - z.B. durch Darstellung von Notifications oder Weiterleitung an bestimmte Activity
- Viele Broadcasts stammen vom System
  - z.B. Ankündigung neue Zeitzone, Batterie fast leer, ...
  - App kann aber auch *interne* Broadcasts verschicken
- **Basisklasse**: `android.content.BroadcastReceiver`

☞ Mehr zu Broadcast Receivern später im Modul

Content Provider: „Datenaustausch zwischen Apps“

- **Content Providers** sind die einzige *direkte** Möglichkeit zum Datenaustausch zwischen Android Applikationen
  - D.h. Austausch von Daten via API Aufruf
  - z.B. Adressbuch, Photo-Galerie, SMS, Dokumente, …
- Bieten Standard-API für Suchen, Löschen, Aktualisieren und Einfügen von Daten
- **Basisklasse**: `android.content.ContentProvider`

☞ Mehr zu Content Providers später im Modul

* Natürlich gibt es auch noch andere Möglichkeiten, z.B. können Daten auch via shared Diskmemory (SD Karte) ausgetauscht werden

Registrierung von Komponenten

- Alle Komponenten einer Applikation müssen dem System bekannt gegeben werden
- Zu diesem Zweck hat jede Android-Applikation eine Datei **AndroidManifest.xml**
- Dieses Manifest enthält u.a.:
  - Informationen über Komponenten der Applikation
  - Statische Rechte (Privileges)
  - Liste mit „uses“ Erlaubnissen (Permissions)
  - Ggf. Einschränkungen für Aufruf (Intent-Filter)

Das Android Manifest

- Beschreibt grundsätzlich die statischen Eigenschaften einer Applikation, z.B:
  - Basis Java-Package-Name
  - Benötigte Rechte (Internet, Kontakte, usw.)
  - Deklaration von Komponenten (siehe vorherige Folie)
    - **A**ctivities, **S**ervices, **C**ontent **P**roviders, **B**roadcast **R**eceivers
    - Name (+ Basis-Package = Java Klasse)
    - Anforderungen für Aufruf (Intent-Filter) für **A, S, BR**
    - Format der gelieferten Daten für **CP**
- Diese Info wird App-Installation im System registriert
- Zusätzliche Informationen (Version, ID, etc.) befinden sich im Build-Skript (können build-abhängig sein)

Details von Manifest-Einträgen behandeln wir später im Modul!

Einfaches Bsp.-Manifest & Demo (Siehe Übung 1)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="ch.hslu.mobpro.hellohslu">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name"
            android:theme="@style/AppTheme.NoActionBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

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

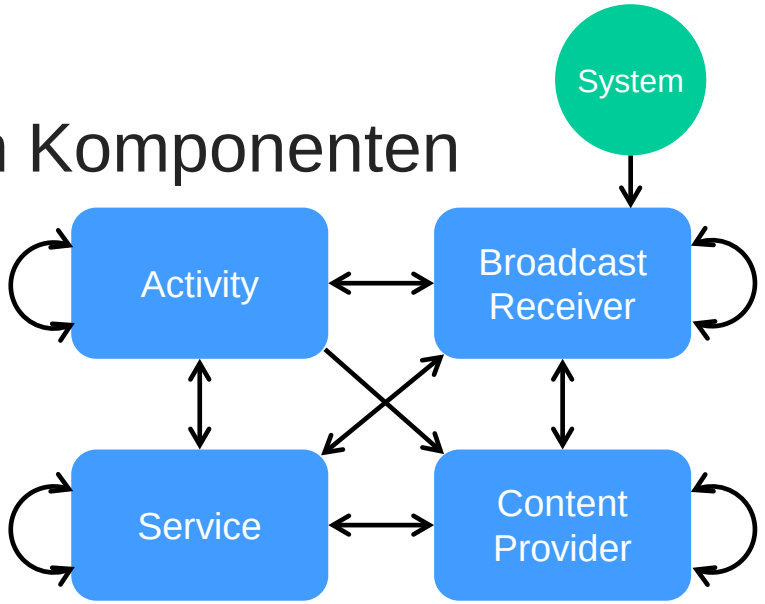
Activities & Aufruf mit Intents

Lose Kopplung zwischen Komponenten

- Prinzip der losen Kopplung
  - Komponenten (z.B. Activities) rufen über Intents (=Nachrichten) andere Komponenten auf
  - Offene Kommunikation: Sender weiss nicht, ob Empfänger existiert
  - Parameterübergabe als Strings (untypisiert)
  - Parameter werden vom Empfänger geprüft, geparst und interpretiert (oder ignoriert)
- Keine expliziten Abhängigkeiten = Robuste Systemarchitektur

Intents: Aufruf andere Komponenten

- Kommunikation zwischen Komponenten
  - Activity <-> Activity
  - Activity <-> Service
  - Service <-> Service
  - ...



- Intents sind der „Leim“ zwischen den Android Systemkomponenten
  - Analogie: Postpaket mit Nachricht (=Daten)
    - *Genaue Adresse* oder *Empfängerbeschreibung*

Explizite Intents

Implizite Intents

Kontrollübergabe mittels Intents

- Android benutzt Intents, um Komponenten zu benachrichtigen oder um Kontrolle zu übergeben
- **Zwei Arten von Intents**:
  - **Explizite Intents** adressieren Komponente direkt
  - **Implizite Intents** beschreiben geeigneten Empfänger
- Implizite Intents kann man sich als Verb (und Objekt) vorstellen: Kurzbeschreibung, was getan werden soll
  - z.B.: *view image, take photo, call contact, play movie*, ...
- Das System übergibt dann der am besten passenden Komponente (Details dazu später)

Beispielaufruf **Expliziter Intent**

- Sender Activity

```
public void onClickSendBtn(final View btn){
    Intent intent = new Intent(this, Receiver.class);
    intent.putExtra("msg", "Hello World");
    startActivity(intent);
}
```

Expliziter Empfänger

- Receiver Activity

Achtung: Activity muss im Manifest deklariert werden, auch wenn Klasse angegeben! Sonst gilt sie nicht als «public». Siehe nächste Folie...

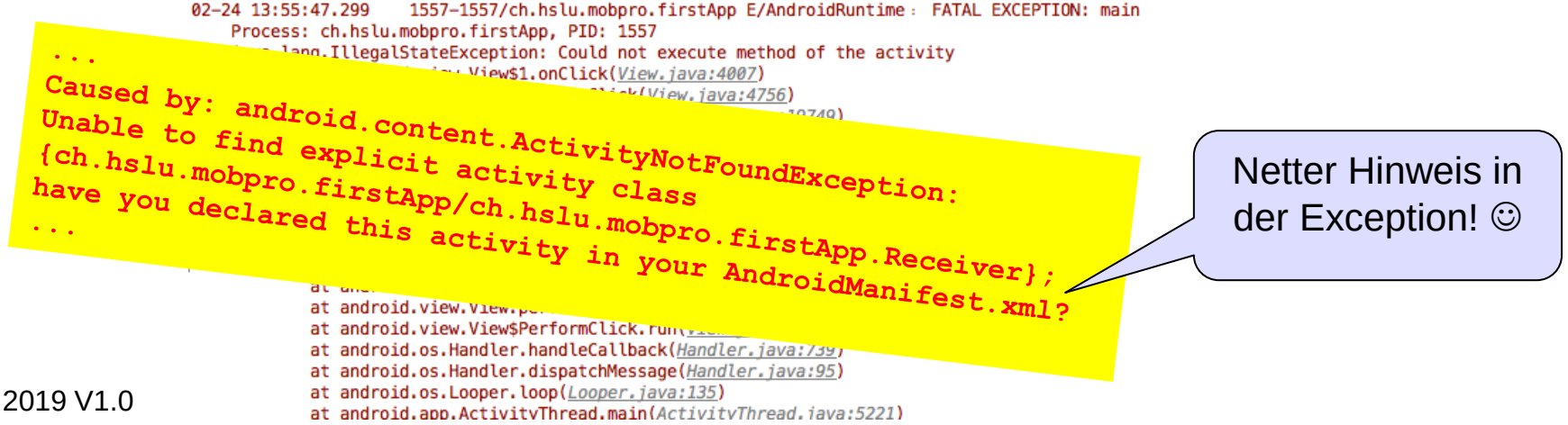
```
public void onCreate(Bundle savedInstanceState){
    // ...
    Intent intent = getIntent();
    String msg = intent.getExtras().getString("msg");
    displayMessage(msg);
}
```

Deklaration Activity im Manifest

- In AndroidManifest.xml (im Element „application“):

```
...
<activity android:name=".Sender" />
<activity android:name=".Receiver" />
...
```

  - Falls Activity NICHT deklariert, passiert beim Aufruf durch expliziten Intent zur Laufzeit folgendes:



Beispielaufruf **Impliziter Intent**

- Sender Activity

```
Intent browserCall = new Intent();
browserCall.setAction(Intent.ACTION_VIEW);
browserCall.setData(Uri.parse("http://www.hslu.ch"));
startActivity(browserCall);
```

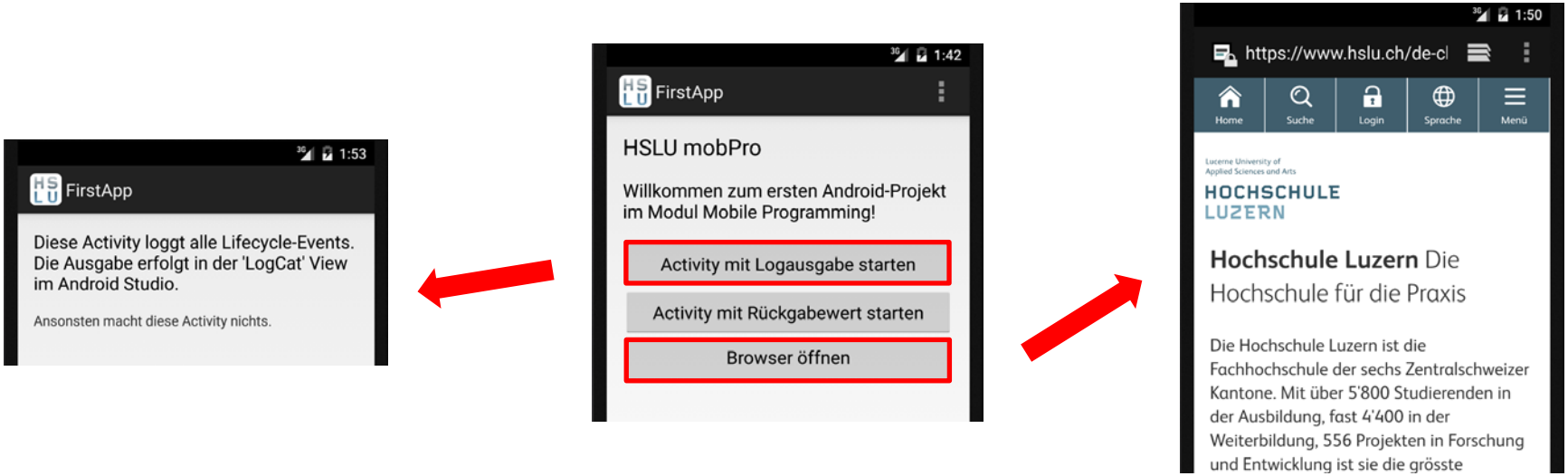
Kein expliziter Empfängertyp, nur gewünschte Aktion

Bedeutung = «Call Parameter» Gesucht ist eine Komponente, welche eine URL anzeigen kann

☞ Receiver-Activity und Manifest-Eintrag dazu schauen wir später an!

Demo: Expliziter und impliziter Intent

- Analog zu Übung 1
  - onClick-Methoden in der MainActivity
  - Expliziter Intent: Log-Seite (LifecycleLogActivity)
  - Impliziter Intent: Browser zeigt http://hslu.ch



Activity Back Stack

- Activities liegen aufeinander wie ein Stapel Karten
- Neue Activity zuoberst, i.d.R. nur diese sichtbar
- Bei «back» oder «finish» wird oberste Karte entfernt → Rückkehr zu zweitletzter Activity
- Mehrere Instanzen der gleichen Activity = mehrere Karten
- Verhalten kann konfiguriert werden

Ausnahmen möglich (Transparenz)

- Max 1 Instanz
- Mehrere Activities öffnen
- ...

(Sub-)Activities & Rückgabewerte

- Eine Activity kann die Rückgabewerte einer anderen Activity (= Sub-Activity) erhalten
  1. Aufruf der SubActivity mit
     `startActivityForResult(intent, requestId)`
  2. SubActivity setzt am Ende Resultat mit
     `setResult(resultCode, intent)`
  3. SubActivity beendet sich mit
     `finish()`
  4. Nach Beendung der Sub-Activity wird
     `onActivityResult(requestId, resultCode, intent)`
     im Aufrufer aufgerufen

Intent als Wrapper für Rückgabewert(e)

OK, CANCEL, ...

Demo: Activity mit Rückgabewert

- Analog zu Übung 1
  - Aufruf Frageseite (QuestionActivity) aus MainActivity
  - Frageseite setzt Resultat
  - MainActivity erhält Resultat nach Beendigung & Anzeige

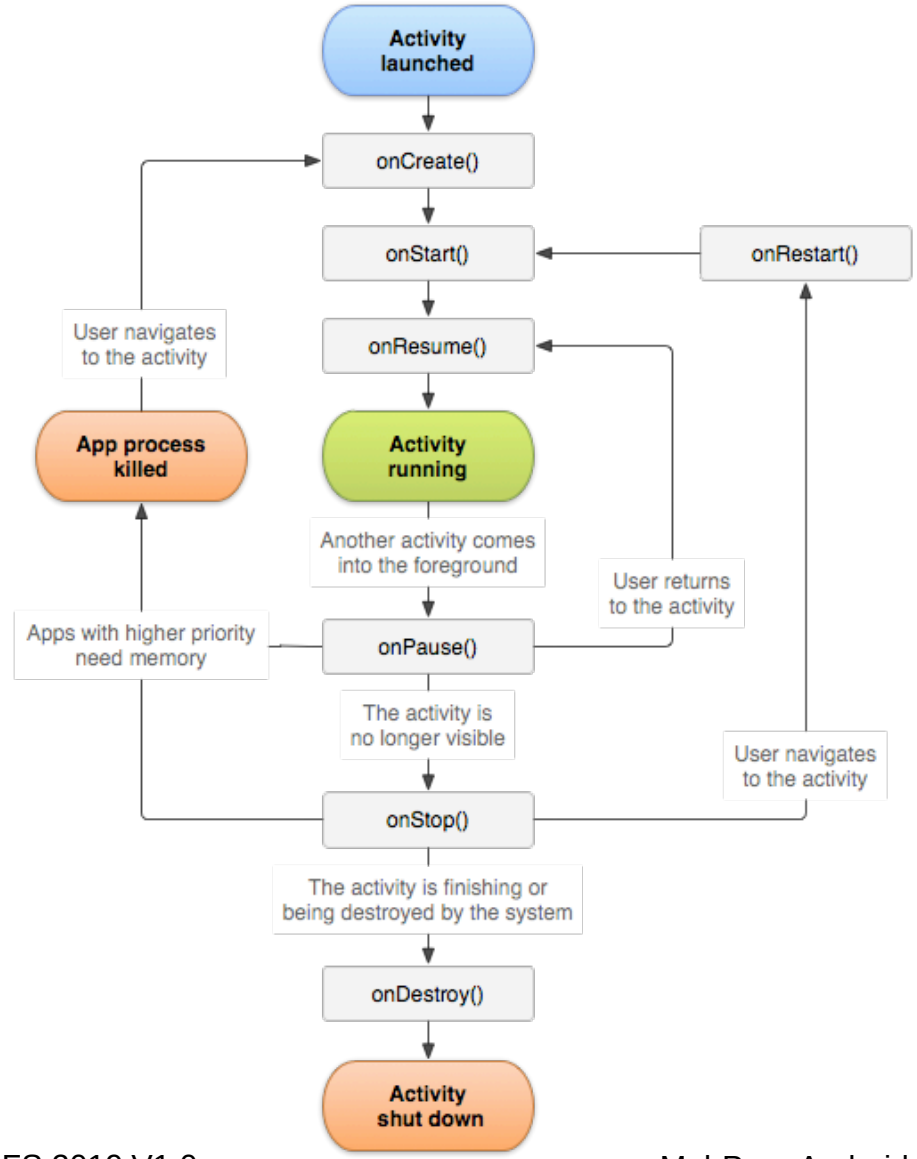
http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg

Lebenszyklus & Zustände von Applikationen bzw. Activities

Lebenszyklus einer Applikation

- Das System kann eine Applikation ohne Vorwarnung terminieren, wenn der Speicher knapp wird
  - Nur Activities im Hintergrund
  - Von User unbemerkt, bei Zurücknavigation wird Applikation wieder hergestellt (siehe nächste Folie)
- D.h. eine Applikation kann ihren Lebenszyklus nicht kontrollieren und muss daher u.a. in der Lage sein, ihren Zustand zu speichern und wieder zu laden

Lebenszyklus einer Activity

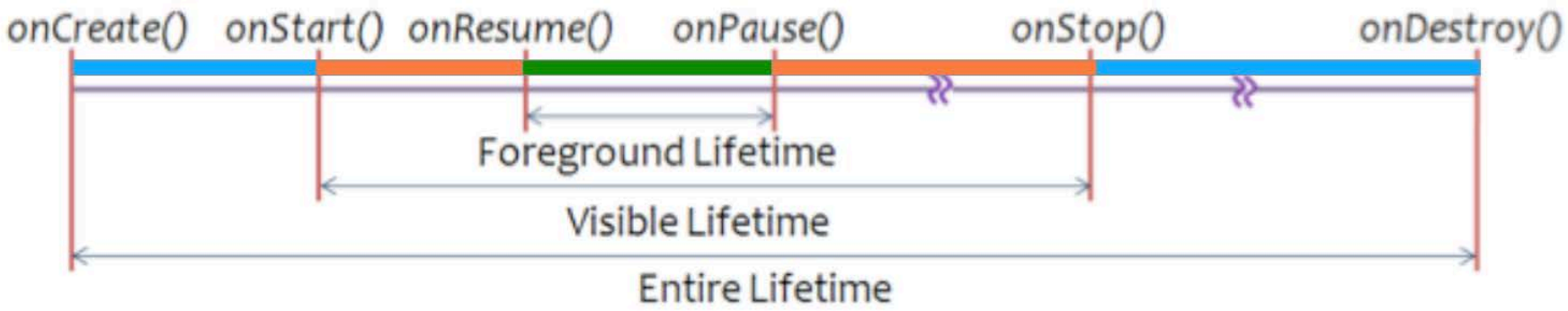


https://developer.android.com/guide/components/activities/activity-lifecycle.html

- Activities gehen im Laufe ihres Lebens durch unterschiedliche Zustände
- Zustandsübergänge rufen Callback-Methoden auf, die von uns überschrieben werden können
- Android-Spezialität: OS kann App killen, wenn Speicher benötigt wird

Activity-Zustände

Zustand	Beschreibung
Running	Die Activity ist im Vordergrund auf dem Bildschirm (zuoberst auf dem Activity-Stack für die aktuelle Aufgabe)
Paused	Die Activity hat den Fokus verloren, ist aber immer noch sichtbar für den Benutzer
Stopped	Die Activity ist komplett verdeckt von einer andern Activity. Der Zustand der Activity bleibt jedoch erhalten



Charakterisierung einer Activity

- GUI Controller
  - Repräsentiert eine Applikationsseite (Bildschirmseite)
  - Definiert Seitenlayout und GUI-Komponenten
  - Kann aus Fragment(en) (=„Sub-Activities“) aufgebaut sein
  - Reagiert auf Benutzereingaben
  - Beinhaltet Applikationslogik für dargestellte Seite
- Muss im Manifest deklariert werden

Beispiel Activity

```
public class Demo extends Activity {
    // Called when the activity is first created
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

Definiert Layout und UI:
Siehe nächste Vorlesung

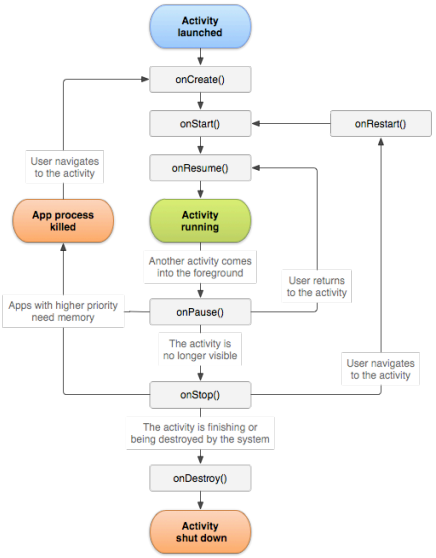
Zustandsänderung: Hook Methoden

- System benachrichtigt Activities durch Aufruf einer der folgenden Methoden der Klasse `Activity`:
  - `void onCreate(Bundle savedInstanceState)`
  - `void onStart()` oder `void onRestart()`
  - `void onResume()`
  - `void onPause()`
  - `void onStop()`
  - `void onDestroy()`

z.B. Animation stoppen

z.B. Ressourcen freigeben

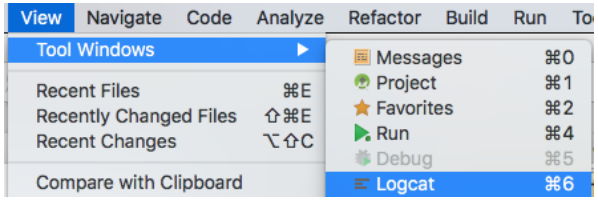
- Durch Überschreiben entsprechender Methoden können wir uns in den Lebenszyklus einklinken
- Unbedingt immer `super()` aufrufen, sonst Exception

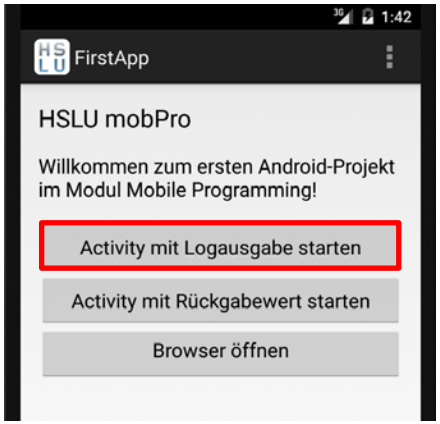


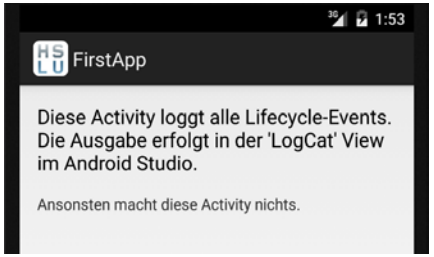
http://developer.android.com/reference/android/app/Activity.html

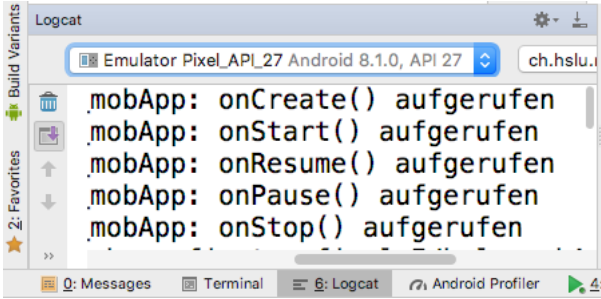
Demo: Logging der Zustandsübergänge

- Ausgangspunkt: Beispielprojekt FirstApp
  - Gerüst im Ilias, siehe Übung 1
  - Aufgabe: *LifecycleLogActivity* mit Logausgabe für alle Lebenszyklus-Methoden-Aufrufe ausstatten
  - Zweck: Alle Lifecycle-Zustände im Log dokumentieren
    - View > Tool Windows > Logcat:



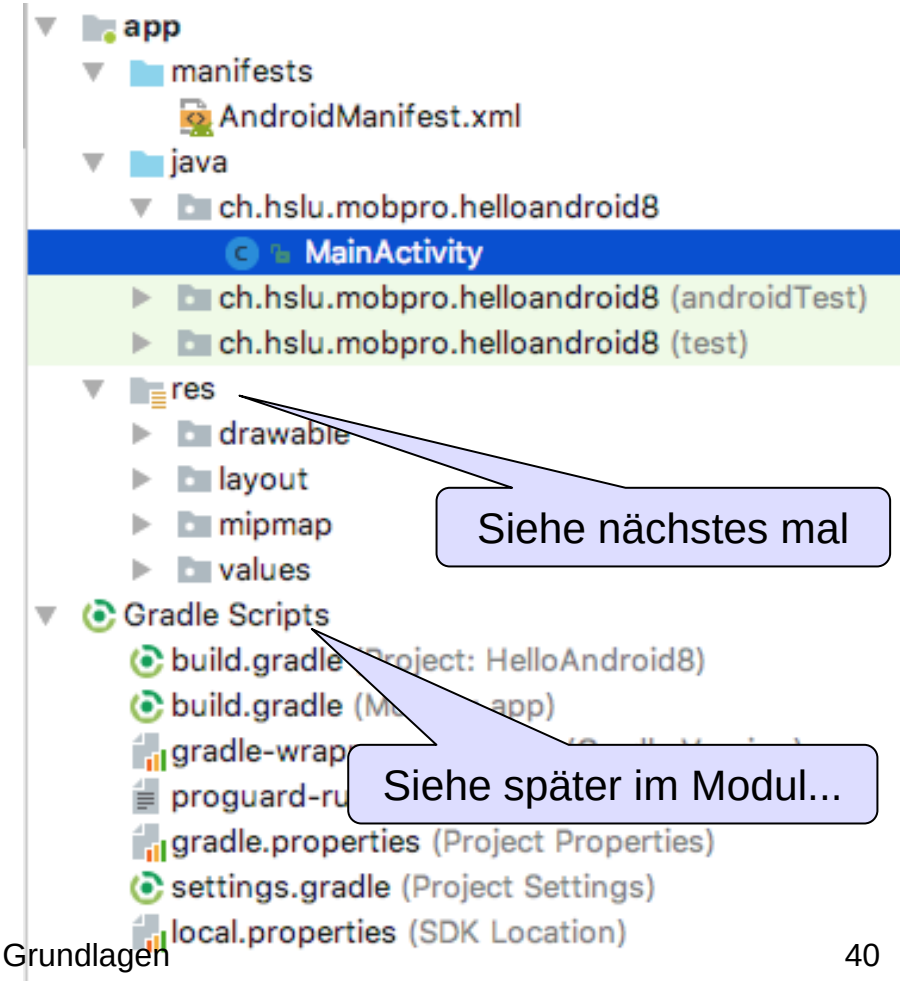






Demo: Struktur eines Android-Projekts

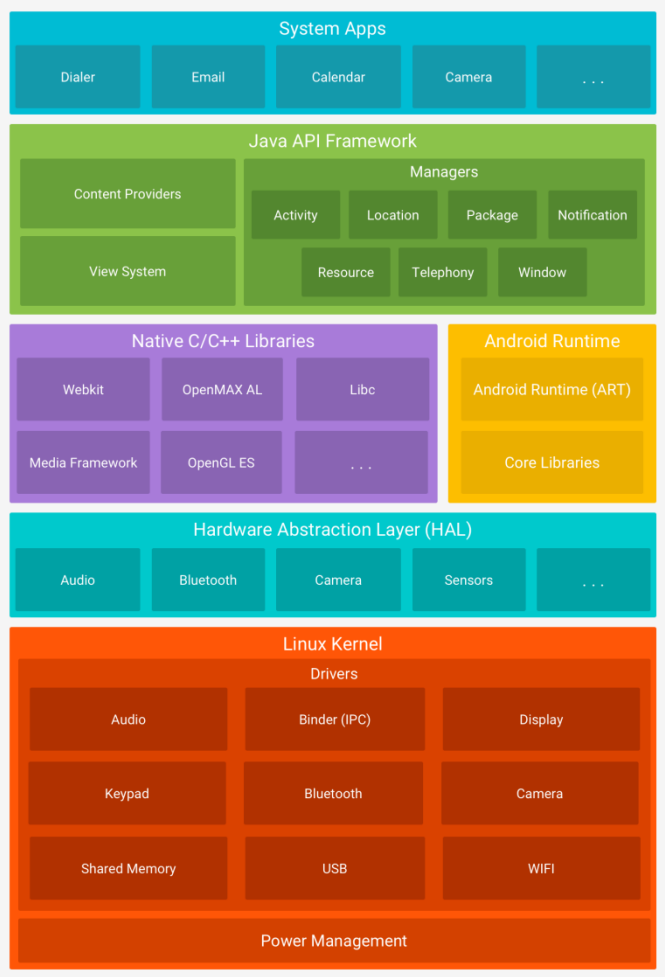
- Manifest
- Java-Code: z.B. Activities
- Ressourcen (res)
  - Bilder (drawable)
  - Layouts (layout)
  - Menus (menu)
  - Werte (value)
- Gradle Skripte
  - Angaben zum Build

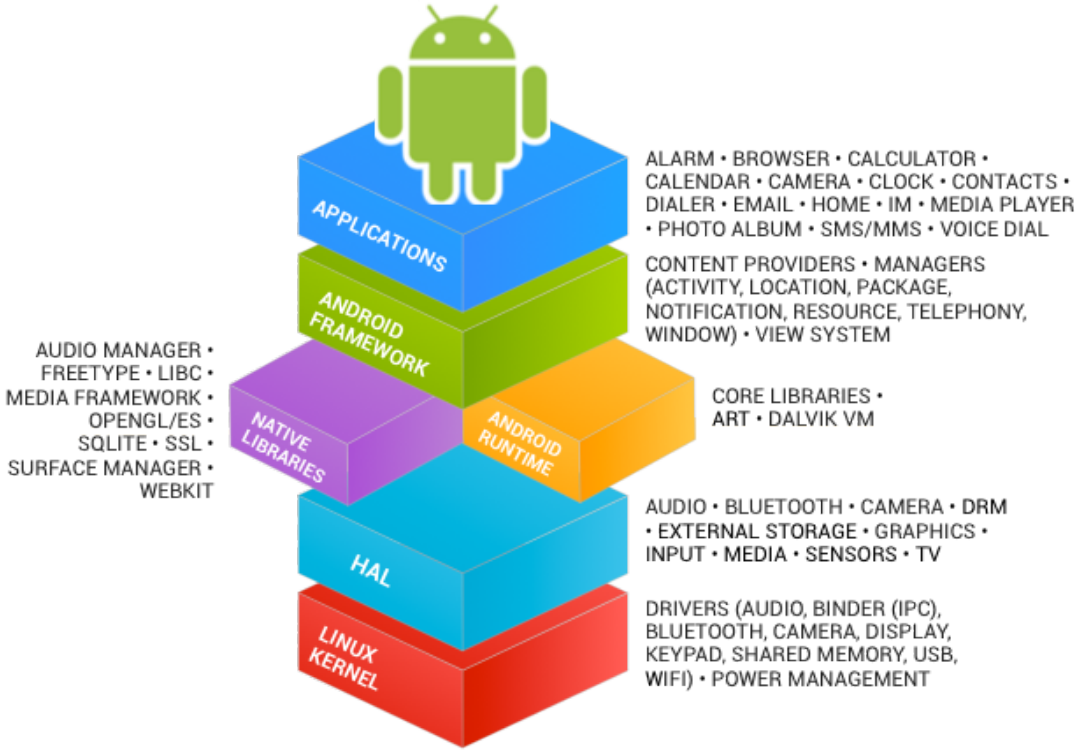


http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg

Android: ein Blick hinter die Kulissen

Der Android Stack





https://developer.android.com/guide/platform/
https://source.android.com/setup/

Plattform-Details

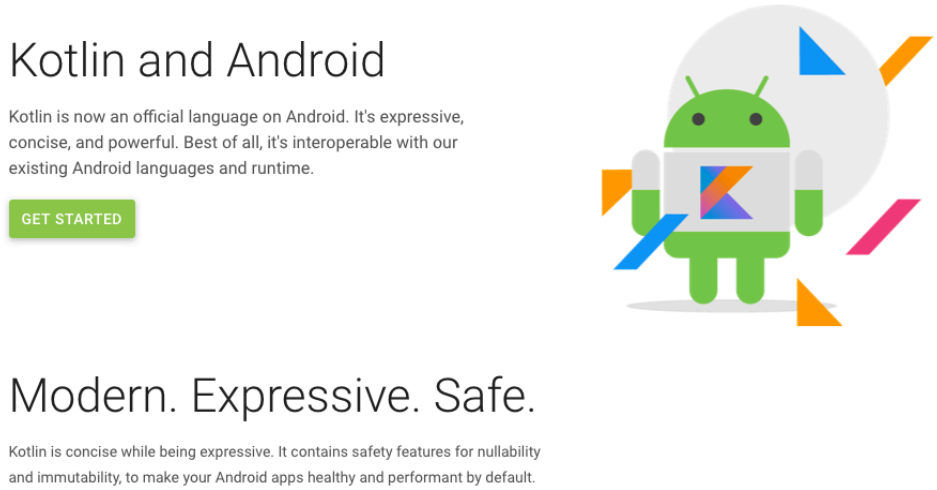
- Linux-Kernel: OS, FS, Security, Drivers, …
- HAL: Camera-, Sensor-, … Abstraktion
- ART (Android Runtime)
  - Jede App in eigenem Prozess
  - Optimiert für mehrere JVM auf low-memory Geräten
  - Eigenes Bytecode-Format (Crosscompiling)
  - JIT und AOT Support
- Native C/C++ Libraries: Zugriff via Android NDK
- Android Framework: Android Java API
- Applications: System Apps und **eigene Apps**

Android & Java 8 Features (Lambda & Co.)

- Android Studio 3+ unterstützt Java 7 und tw. Java 8
- Java 8 Features müssen explizit enabled werden
  - https://developer.android.com/studio/write/java8-support
- Damit werden unter Android auch Streams, Lambdas, Methodenreferenzen und statische Interfacemethoden nutzbar (benötigt API-Level 24+)
- Ist nicht Pflicht für‘s Modul aber empfohlen, um «moderner» und funktionaler implementieren zu können

```
String[] names = {"Kaspar", "Ruedi", "X"};
String result= Arrays.stream(names)
        .filter(s -> s.length() > 3)
        .map(String::toUpperCase)
        .collect(Collectors.joining( delimiter: "," ));
```

Android & Kotlin



- Kotlin
  - V1.0 released in 2011
    - Aktuell V1.3
  - JVM-Sprache von JetBrains
  - Kompiliert zu Java-Bytecode oder JavaScript
  - **Seit Android Studio 3.0 offizielle 2. Android-Sprache**

☞ Dürfen Sie im Modul und für Projekt gerne einsetzen ☺

Beachte: Modul (Code auf Folien, Übungen, MEP usw.) setzt auf Java

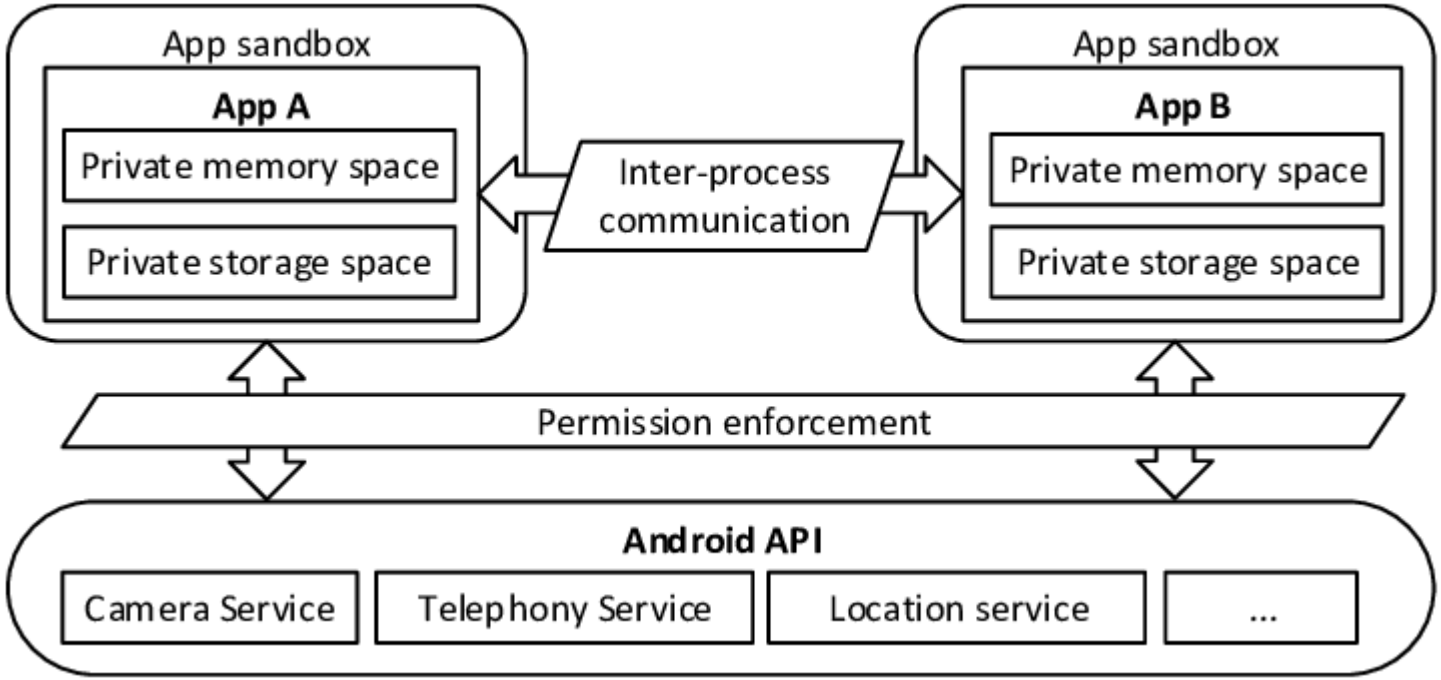
Das Android Security Konzept

- Sandkasten-Konzept (Sandbox)
  - Jede laufende Android-Anwendung hat eigenen Prozess, Benutzer, ART-Instanz, Heap und Dateisystembereich
  - Berechtigungssystem von Linux: Betrifft sowohl Speicherzugriff wie auch Dateisystem (Benutzer-basiert)
  - Signieren von Anwendungen
    - Erschwert Code-Manipulationen erschweren (Viren)
    - Erlaubt Teilen einer Sandbox bei gleicher sharedUser-Id
- Berechtigungen
  - Im Android-Manifest definiert

Kontrollierte Öffnung der Sandbox-Restriktionen

https://source.android.com/security/app-sandbox

Security-Modell: Jede App hat eigenen Linux-User



https://www.researchgate.net/figure/Two-sandboxed-Android-apps-and-their-interaction-with-one-another-and-with-the-Android_fig5_307984756

TODO: Update

...icklungs-umgebung: Android Studio

http://en.wikipedia.org/wiki/File:Somethingdifferen

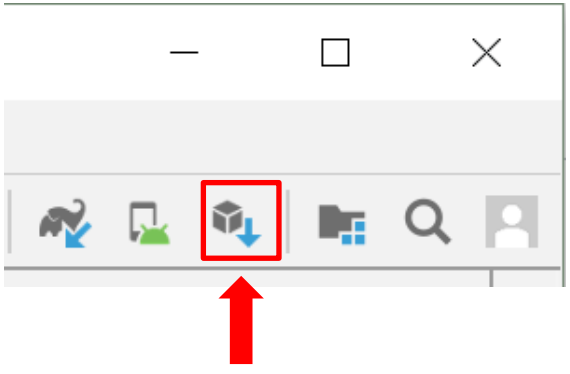
Android Studio

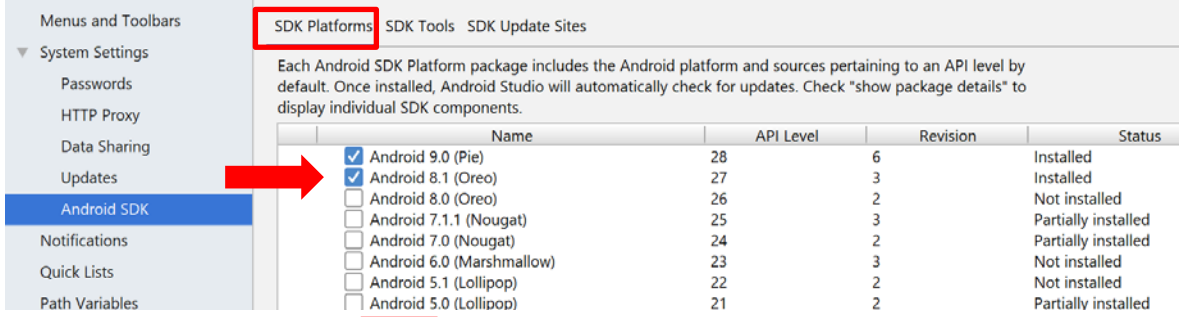
- Komplette Entwicklungsumgebung für Android-Apps
  - Editoren, Compiler, Refactoring, Analyse, Emulator, Debugger, Build (Gradle), VCS (Git), etc.
- Basiert auf IntelliJ IDEA
  - Aktuelle Version: 3.3.1
  - https://developer.android.com/studio/

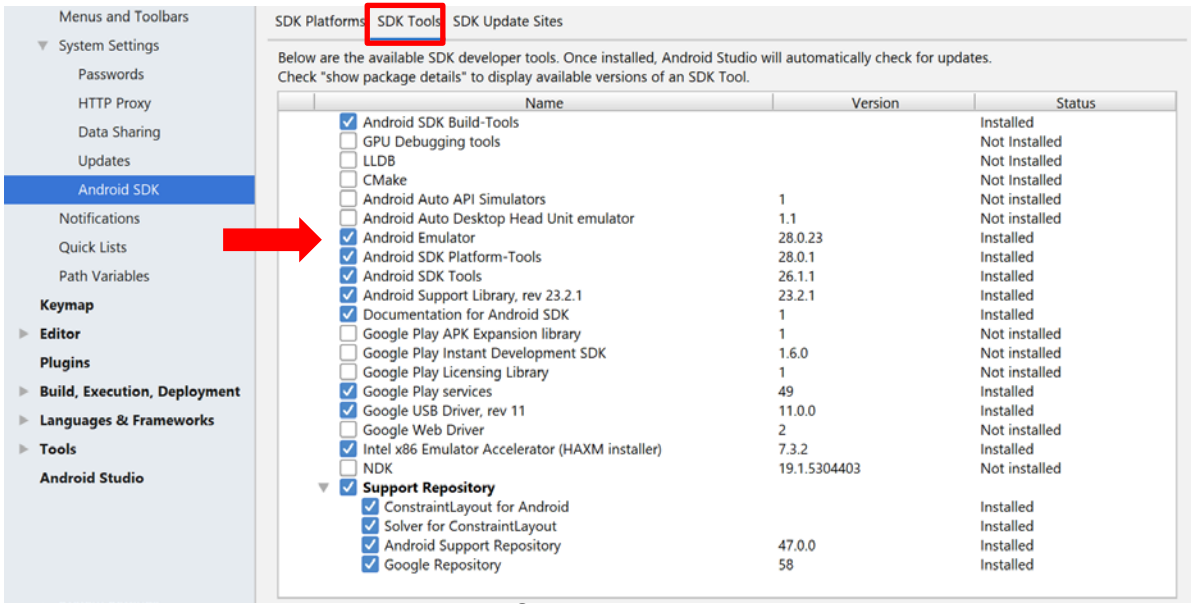
Demo: Android Plattform(en) installieren

Mit **Android SDK Manager** die SDK Plattform für Android 9 (API-Level 28) herunterladen & installieren

+ SDK Tools (Emulator)

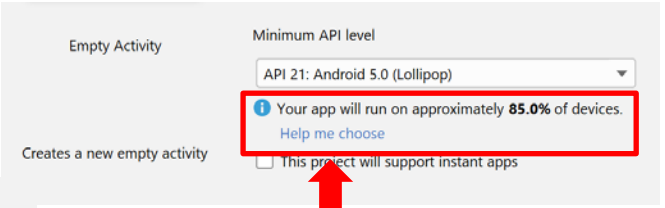






Welche Android-Version wählen?

- New-Project Wizard hilft bei Entscheid



ANDROID PLATFORM VERSION	API LEVEL	CUMULATIVE DISTRIBUTION
4.0 Ice Cream Sandwich	15	
4.1 Jelly Bean	16	99.6%
4.2 Jelly Bean	17	98.1%
4.3 Jelly Bean	18	95.9%
4.4 KitKat	19	95.3%
5.0 Lollipop	21	85.0%
5.1 Lollipop	22	80.2%
6.0 Marshmallow	23	62.6%
7.0 Nougat	24	37.1%
7.1 Nougat	25	14.2%
8.0 Oreo	26	6.0%
8.1 Oreo	27	1.1%

Lollipop

User Interface
Material design support
Concurrent documents and activities in the recents screen
WebView updates
Screen capturing and sharing

Notifications
Lock screen notifications
Notifications metadata

Graphics
Support for OpenGL ES 3.1
Android Extension Pack

Media
Camera API for advanced camera capabilities
Audio playback
Media playback control
Media browsing

Storage
Directory selection

Wireless & Connectivity
Multiple network connections
Bluetooth Low Energy
NFC enhancements

Battery - Project Volta
Scheduling jobs
Developer tools for battery usage

Android in the Workplace and in Education
Managed provisioning
Device owner
Screen pinning

Printing Framework
Render PDF as bitmap

System
App usage statistics

Testing & Accessibility
Testing and accessibility improvements

IME
Easier switching between input languages

Manifest Declarations
Declarable required features
User permissions

https://developer.android.com/about/versions/android-5.0.html

OK Cancel

hm, 9.x fehlt hier noch komplett...

Android Build System: Gradle

- Default Build System: Gradle (Details später im Modul!)
- Bsp. Auszug aus app/build.gradle:

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 27
    defaultConfig {
        applicationId "ch.hslu.mobpro.helloandroid8"
        minSdkVersion 21
        targetSdkVersion 27
        versionCode 2
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.Andr
    }
```

SDK-Versions: compile, target & min

```
compileSdkVersion 27
defaultConfig {
    applicationId "ch.hs
    minSdkVersion 21
    targetSdkVersion 27
```

- **targetSdkVersion**: Für diese Version ist App gebaut, diese API-Version kann voll ausgeschöpft werden

- **minSdkVersion**: Die tiefste Version, auf welcher diese App lauffähig ist
  - Bedingt Einschränkung bei verwendeten APIs oder Verwendung von Support-Library (Backports)
  - App wir nicht angezeigt im Store für Geräte mit < Version

- **compileSdkVersion**: Diese Version wird verwendet, um die App (sprich die APK-Datei) zu erstellen
  - Entspricht typischerweise der targetSdkVersion

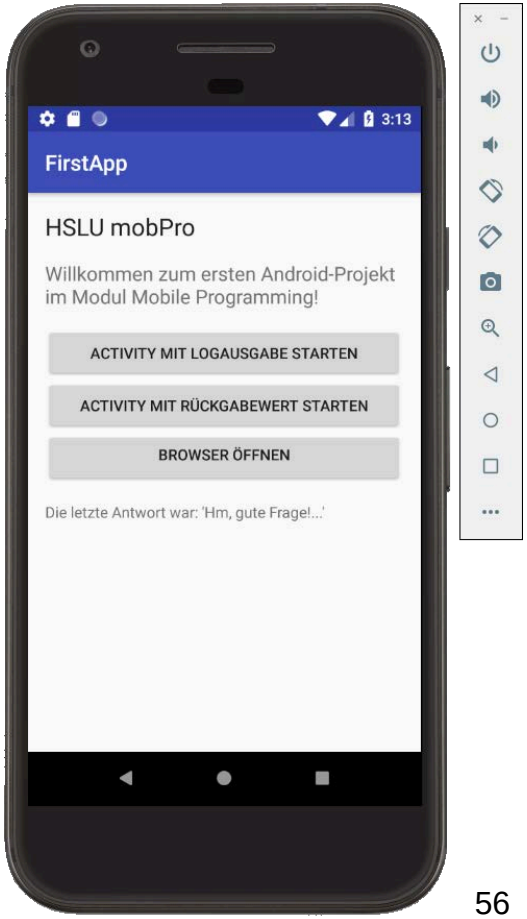
Android-Emulator

- Emuliert Android auf einem virtuellen Gerät
  - App läuft mit 1:1 demselben App-Paket wie auf Gerät
- AVD = Android Virtual Device
  - Verschiedene Eigenschaften
- Empfehlung: HAXM installieren

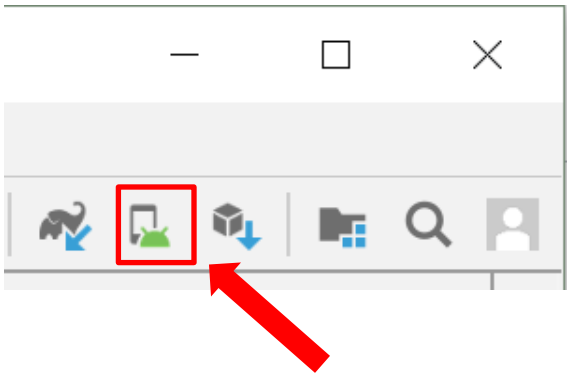
Google Web Driver	2	Not installed
Intel x86 Emulator Accelerator (HAXM installer)	7.3.2	Installed
NDK	19.1.5304403	Not installed

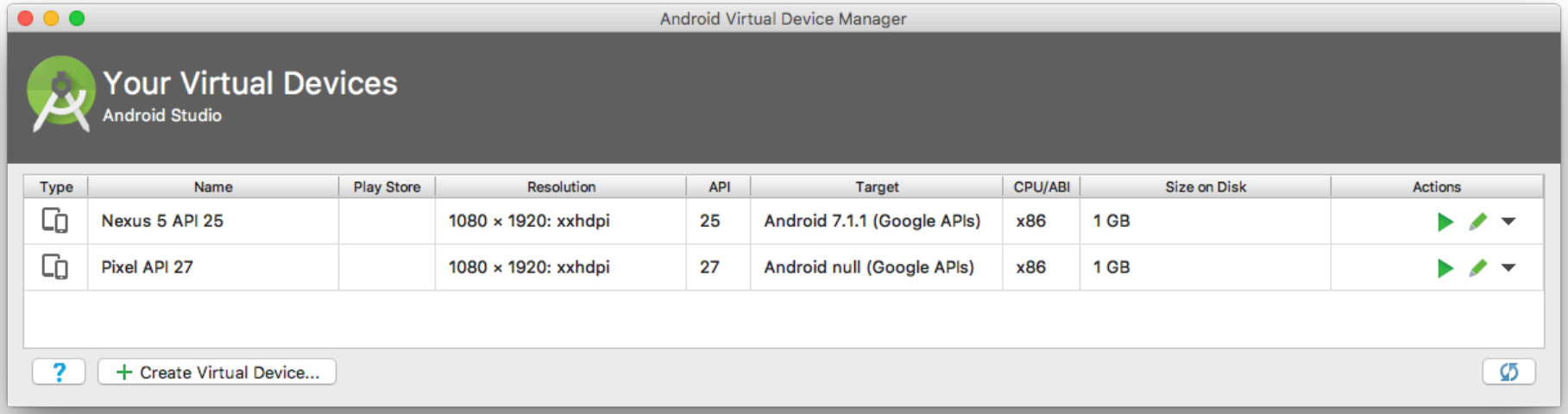
  - Achtung bei Windows + Docker: Gleichzeitige Verwendung nicht möglich
- Für Entwicklung möglichst HW-Gerät verwenden, ist am schnellsten

Emulator für Testing von anderen HW-Konfigurationen (z.B. Auflösung)



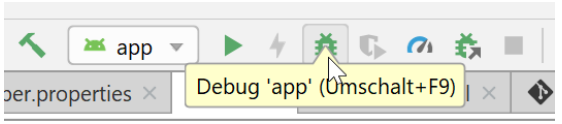
Demo: Android Virtual Device Manager

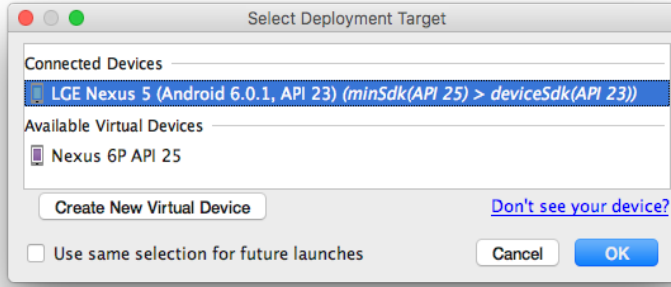




AVD = Android Virtual Device („Emulator“)

- Wenn mind. ein AVD definiert oder ein HW-Gerät via USB verbunden, dann kann eine App gestartet werden
  1. Projekt öffnen
  2. Toolbar > Debug/Run App (Ctrl-D, -R)
  3. Falls mehrere AVDs definiert sind
     - Auswahl AVD oder...
     - HW-Geräte

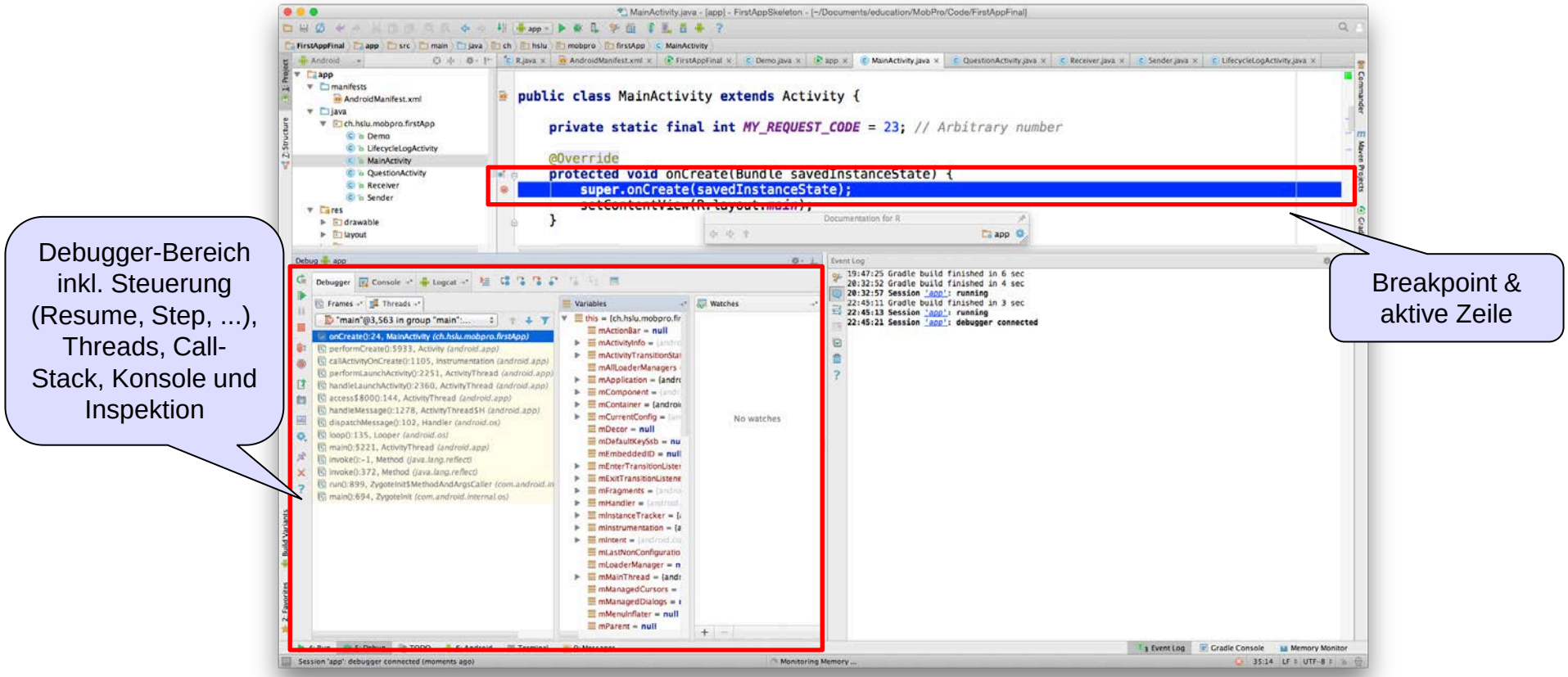




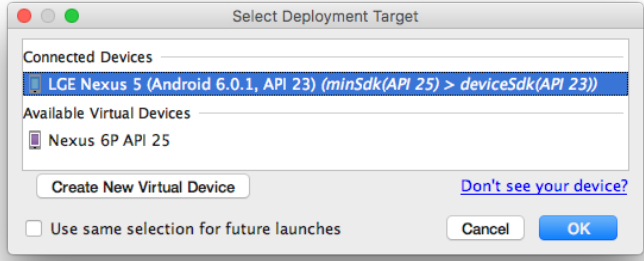
- Hinweis: braucht viel CPU
- Tipp: Emulator nie schliessen
  - Cold Start Emulator kann dauern! (mit HAXM schneller)

Demo: Debug-Modus (Demo)

- Breakpoints setzen, running Code inspizieren, usw.
- Projekt/App starten mit *Toolbar > Debug (Ctrl-D)*

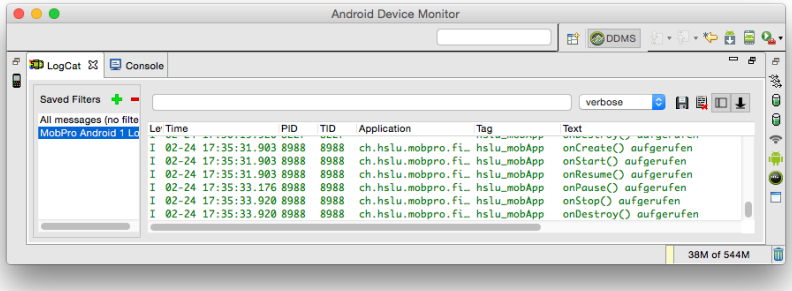


Demo: Projekt erstellen



1. Neues Projekt erstellen: Hello Android!
   - App starten auf AVD & HW-Gerät
2. Projekt importieren aus Dateisystem:
   - File -> Open...
   - Gerüstprojekt „FirstAppSkeleton“ aus dem Ilias
3. Debugger
   - Breakpoint setzen
   - Laufenden Code inspizieren, ...

Zur Übung 1



- Android Studio einrichten inkl. mind 1 AVD
- *HelloWorld* auf Emulator und Gerät
  - Erste eigene App! ☺
- *FirstAppSkeleton* ausbauen
  - Ilias: *MobPro_Android-Uebung-1*
  - Activities mit Intents aufrufen
  - Alle Zustandswechsel einer Activity loggen und provozieren