



FACHHOCHSCHULE  
ERFURT UNIVERSITY  
OF APPLIED SCIENCES  
Angewandte  
Informatik

# Programmierung mobiler Endgeräte

## Android Grundlagen

Dr. Steffen Kern

Erfurt University of Applied Sciences

- Plattform für mobile Endgeräte wie Smartphones und Tablets
- Aktuelle Version: 4.4 (Kit Kat)



G1



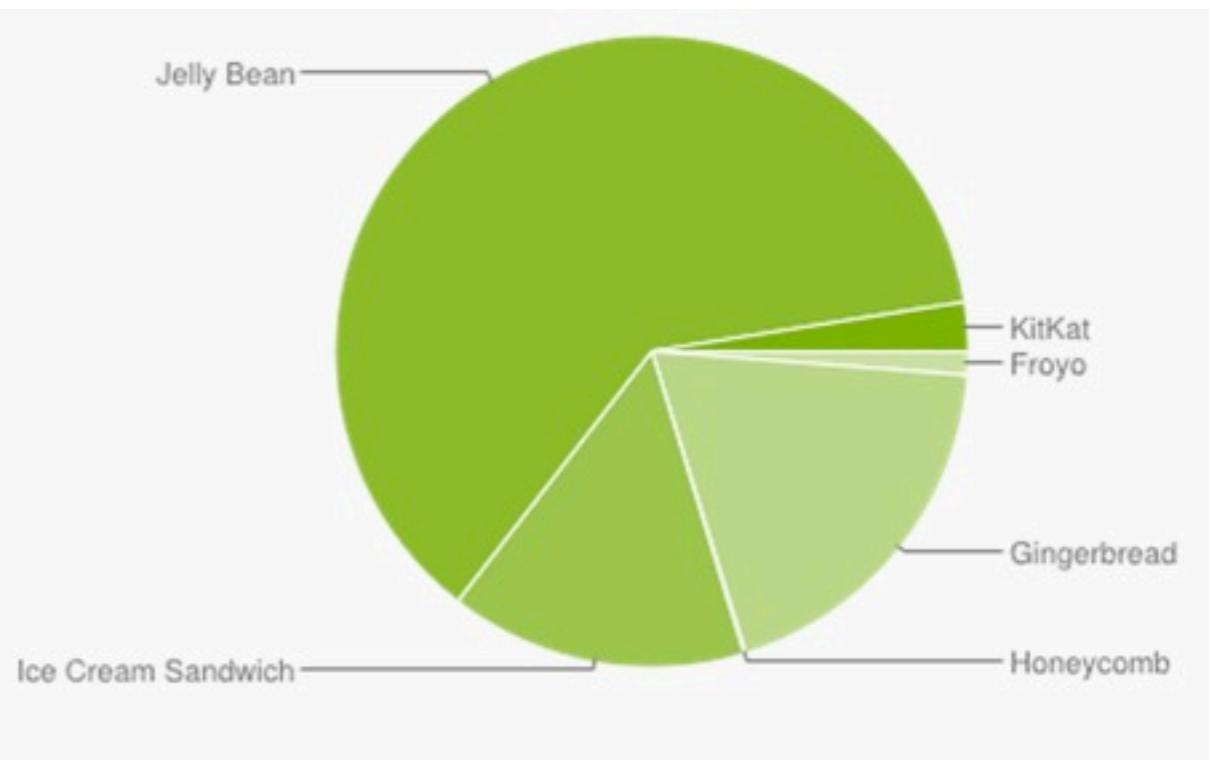
Motorola Xoom



HTC one

## Marktverteilung der Android-Versionen (via Google Play)

Version	Codename	API	Distribution
2.2	Froyo	8	1.2%
2.3.3 - 2.3.7	Gingerbread	10	19.0%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	15.2%
4.1.x	Jelly Bean	16	35.3%
4.2.x	Jelly Bean	17	17.1%
4.3		18	9.6%
4.4	KitKat	19	2.5%

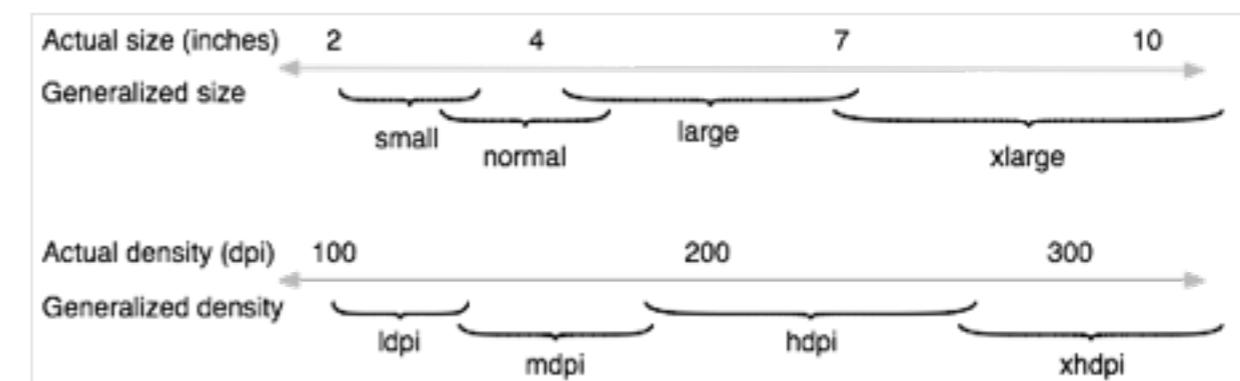


*Data collected during a 7-day period ending on March 3, 2014.  
Any versions with less than 0.1% distribution are not shown.*

Quelle: <http://developer.android.com/resources/dashboard/platform-versions.html>

## Bildschirmgrößen und -auflösungen

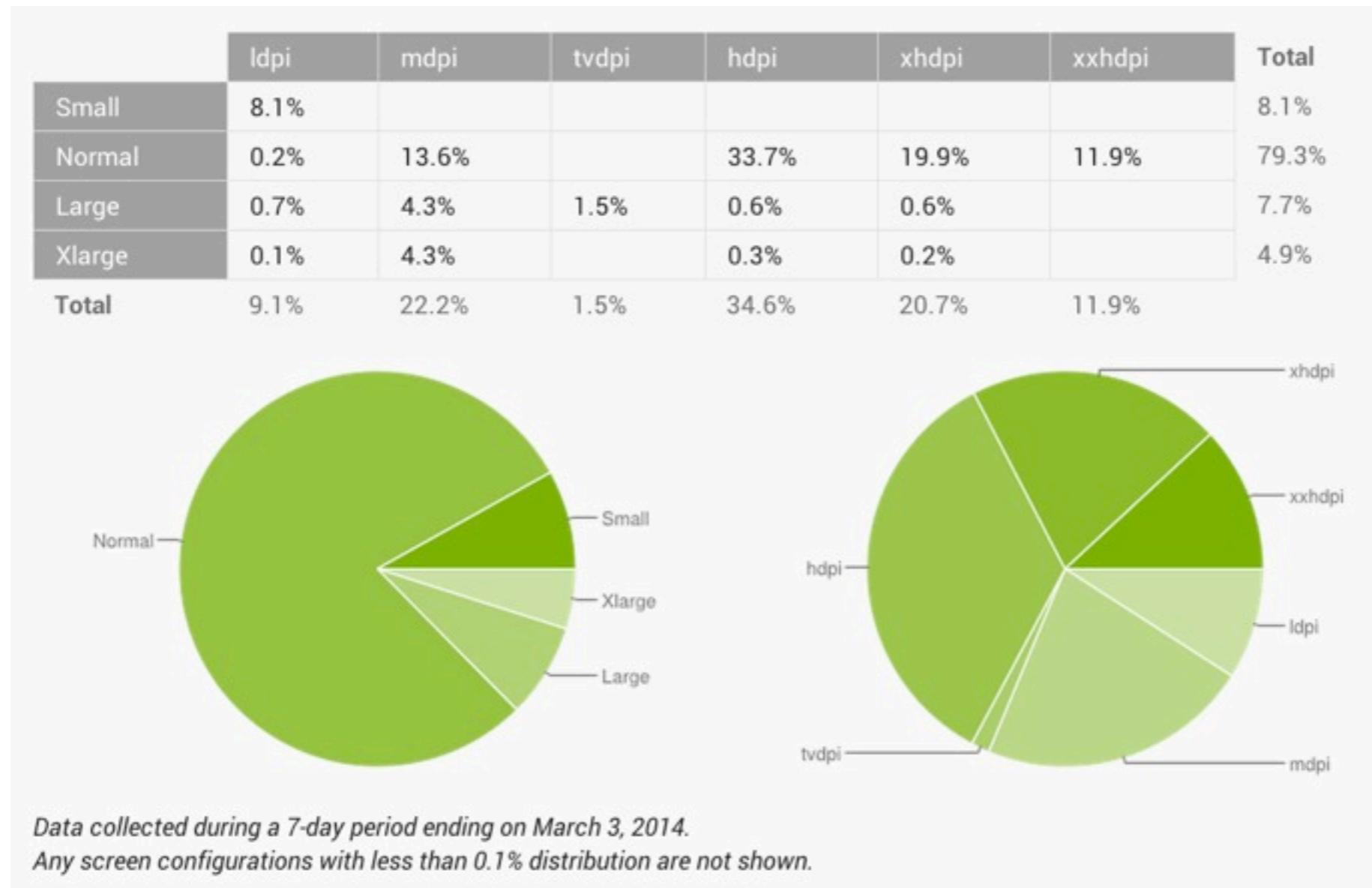
	Low density (120), <i>ldpi</i>	Medium density (160), <i>mdpi</i>	High density (240), <i>hdpi</i>	Extra high density (320), <i>xhdpi</i>
<i>Small screen</i>	QVGA (240x320)		480x640	
<i>Normal screen</i>	WQVGA400 (240x400) WQVGA432 (240x432)	HVGA (320x480)	WVGA800 (480x800) WVGA854 (480x854) 600x1024	640x960
<i>Large screen</i>	WVGA800** (480x800) WVGA854** (480x854)	WVGA800* (480x800) WVGA854* (480x854) 600x1024		
<i>Extra Large screen</i>	1024x600	WXGA (1280x800) <sup>†</sup> 1024x768 1280x768	1536x1152 1920x1152 1920x1200	2048x1536 2560x1536 2560x1600



Quellen:

[http://developer.android.com/guide/practices/screens\\_support.html](http://developer.android.com/guide/practices/screens_support.html)  
<http://developer.android.com/design/style/devices-displays.html>

## Verbreitung von Bildschirmtypen



Quelle: <http://developer.android.com/resources/dashboard/platform-versions.html>

## Eingabemöglichkeiten

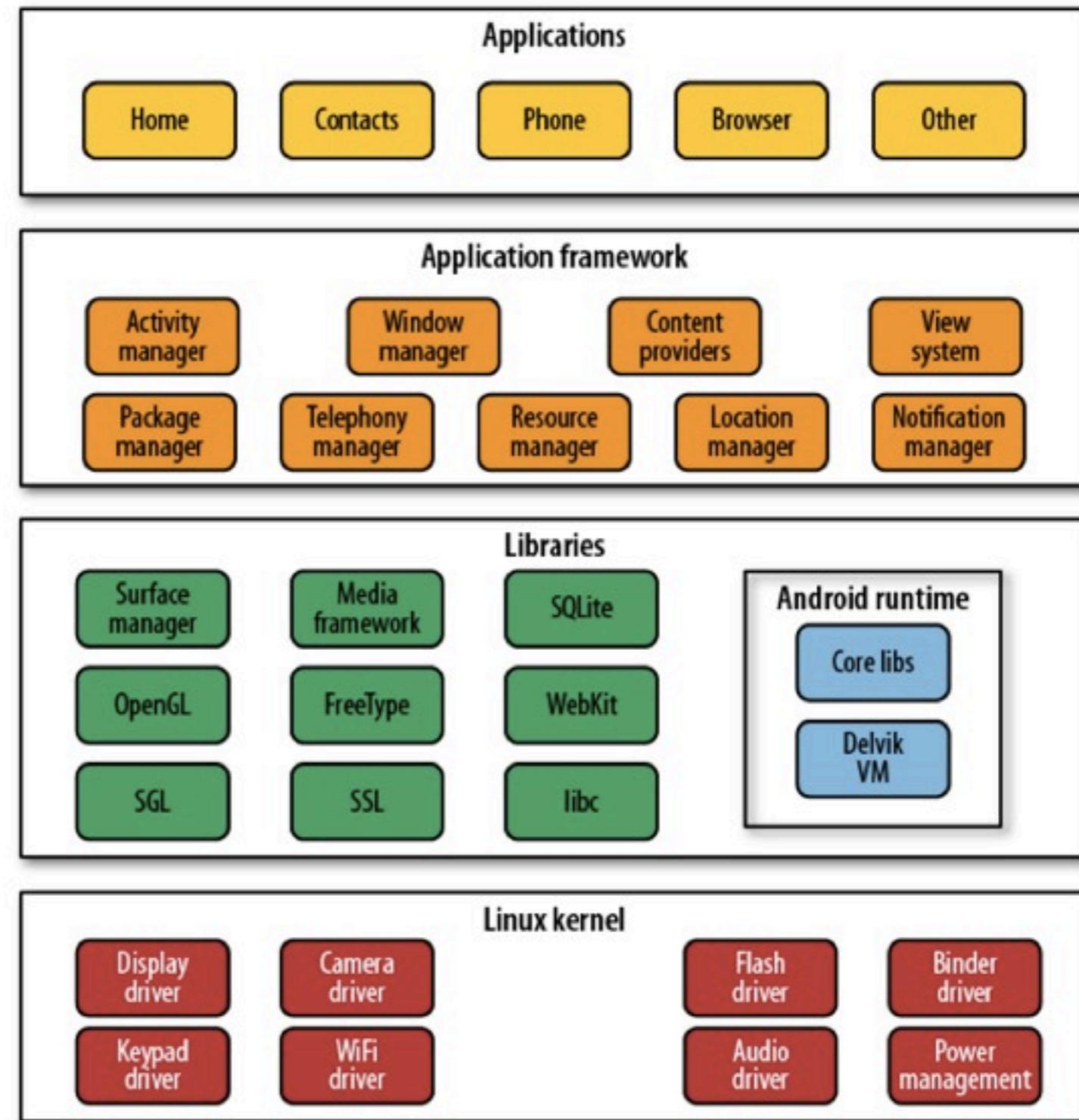
- Keyboard
- D-Pad oder Trackpad (inzwischen sehr selten)
- Multitouch
- Kamera
- Sprachsteuerung
- Lage- und Bewegungssensoren
- NFC (Near Field Communication)

## Sensoren

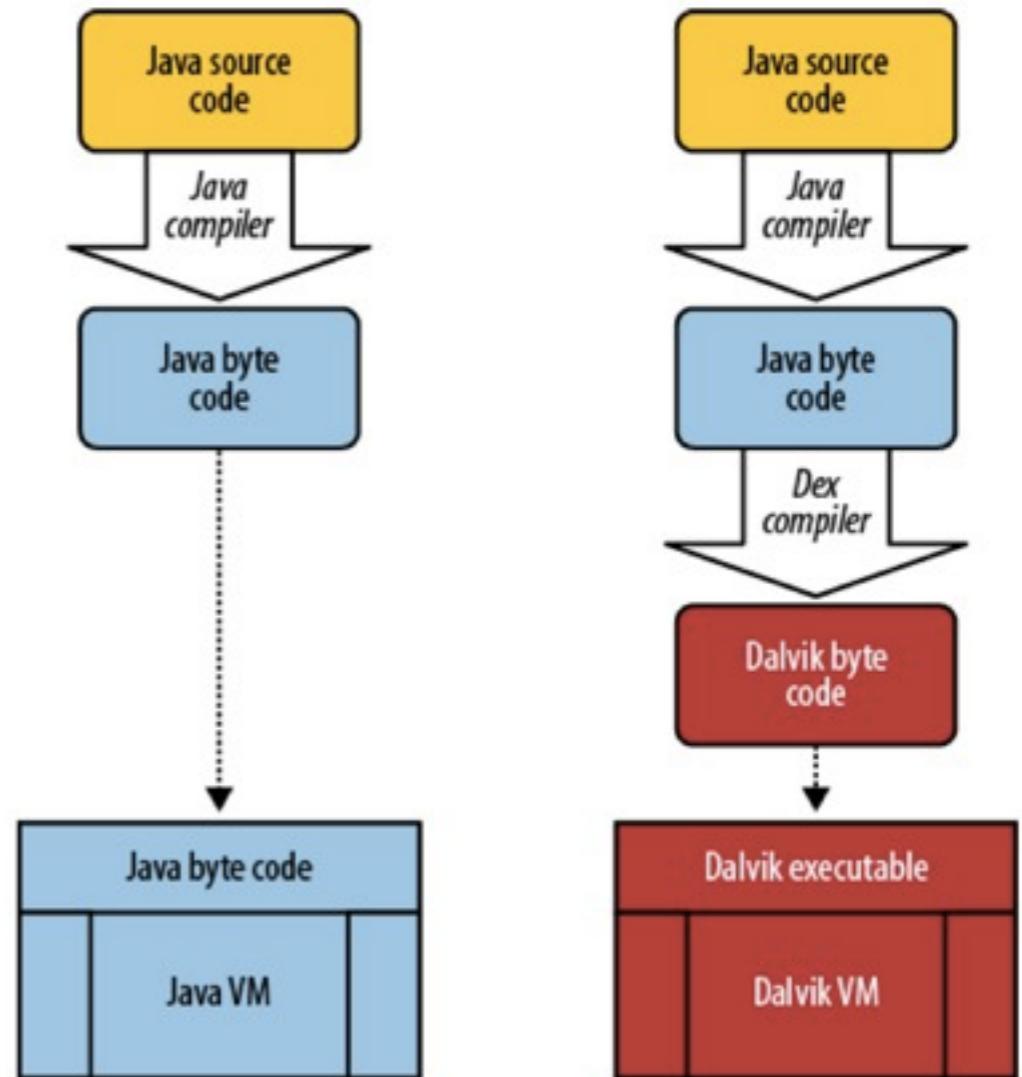
- Ortung via GPS, WiFi und Mobilfunkzelle
- Kompass
- Lage, Bewegung, Beschleunigung
- Temperatur
- Entfernung
- Umgebungslicht

- Multi-Process
  - Ein Prozess pro Applikation
  - Applikationen und Hintergrunddienste werden parallel ausgeführt
- App Widgets auf dem Homescreen
- Touch, Gesten, Multitouch
- Hard- und Softwarekeyboards
  - Unterstützung für beide Typen nötig
  - Wichtig vor allem in Bezug auf das Screen-Design

## Systemaufbau

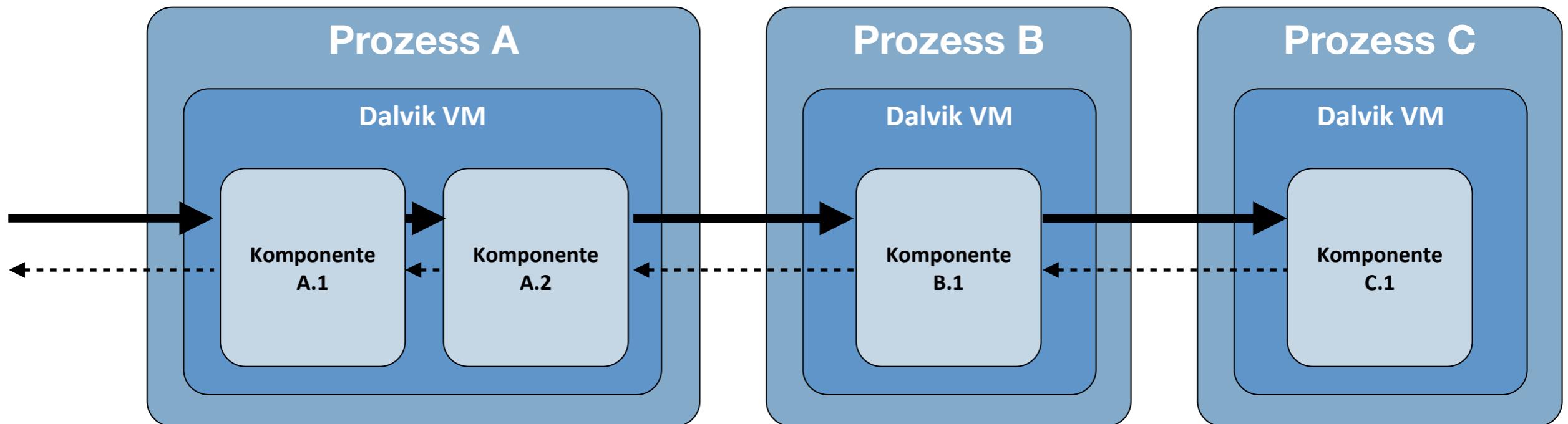


- Applikationen werden in Java programmiert
- Code und Daten werden in einem Android Package zusammengefasst (apk-Datei)
- Verteilung und Installation erfolgt über diese apk-Datei
- **Dalvik** = Android Java VM
  - <http://code.google.com/p/dalvik>



- Android-Anwendungen bestehen aus verschiedenen, modularen Komponenten
- Eine Anwendung kann Komponenten von anderen Anwendungen nutzen
- Komponententypen
  - Activity
  - Service
  - Broadcast Receiver
  - Content Provider

## Komponenteninteraktion



- Java SDK

- <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

und

- Android Studio (aktuell Version 0.5.2)

- <http://developer.android.com/sdk/installing/studio.html>
  - Basiert auf IntelliJ Idea 13
  - Enthält alle Tools für die Android-Entwicklung

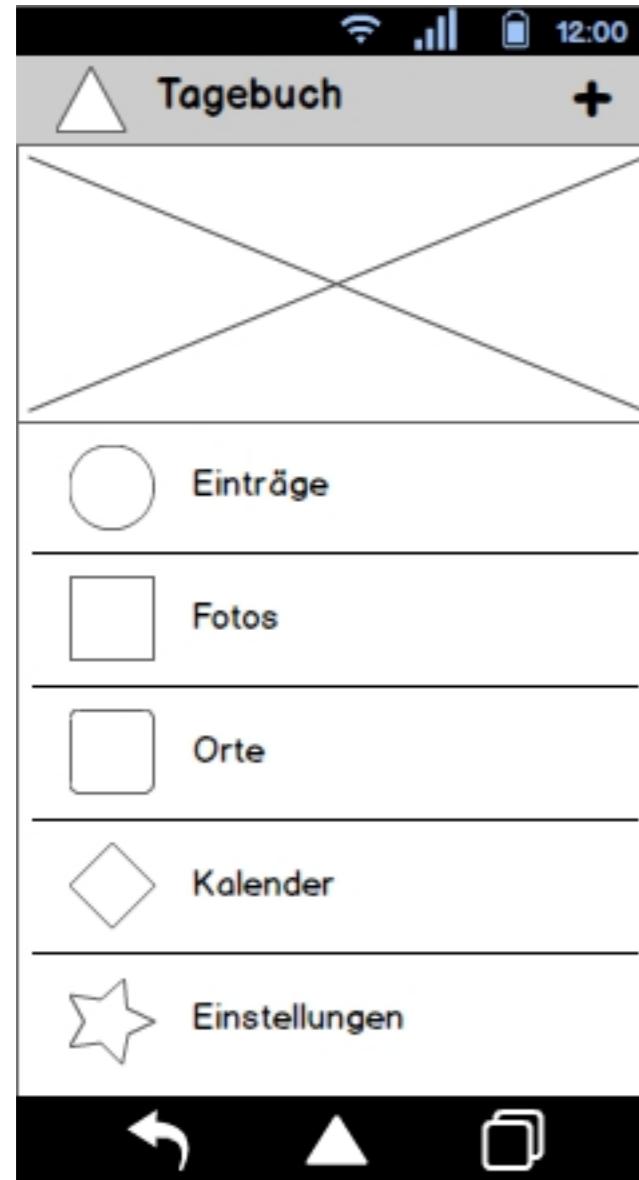
oder

- Eclipse + Android SDK + Plugin

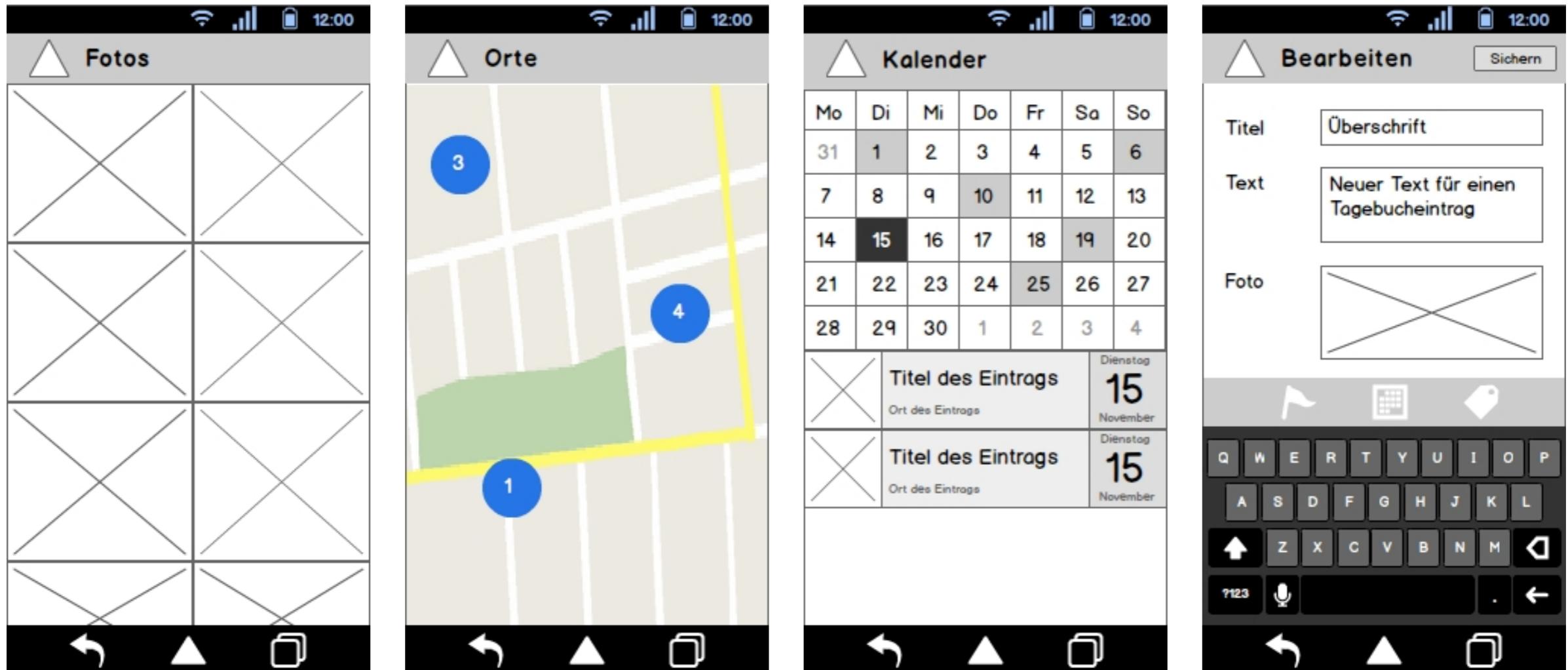
- Eclipse IDE for Java Developers: <https://www.eclipse.org/downloads/>
  - Android SDK: <http://developer.android.com/sdk/index.html>
  - Plugin: <http://developer.android.com/tools/sdk/eclipse-adt.html>

- Tagebuch-App
- Schreiben von Einträgen mit Foto und Geo-Position
- Auswahl oder Aufnahme eines Fotos
- Vergabe von Tags für Einträge
- Darstellung der Position auf einer Karte
- Übersicht über alle Einträge mit Foto
- Übersicht über alle Einträge mit bestimmten Tags
- Kalenderansicht

# Beispielprojekt II



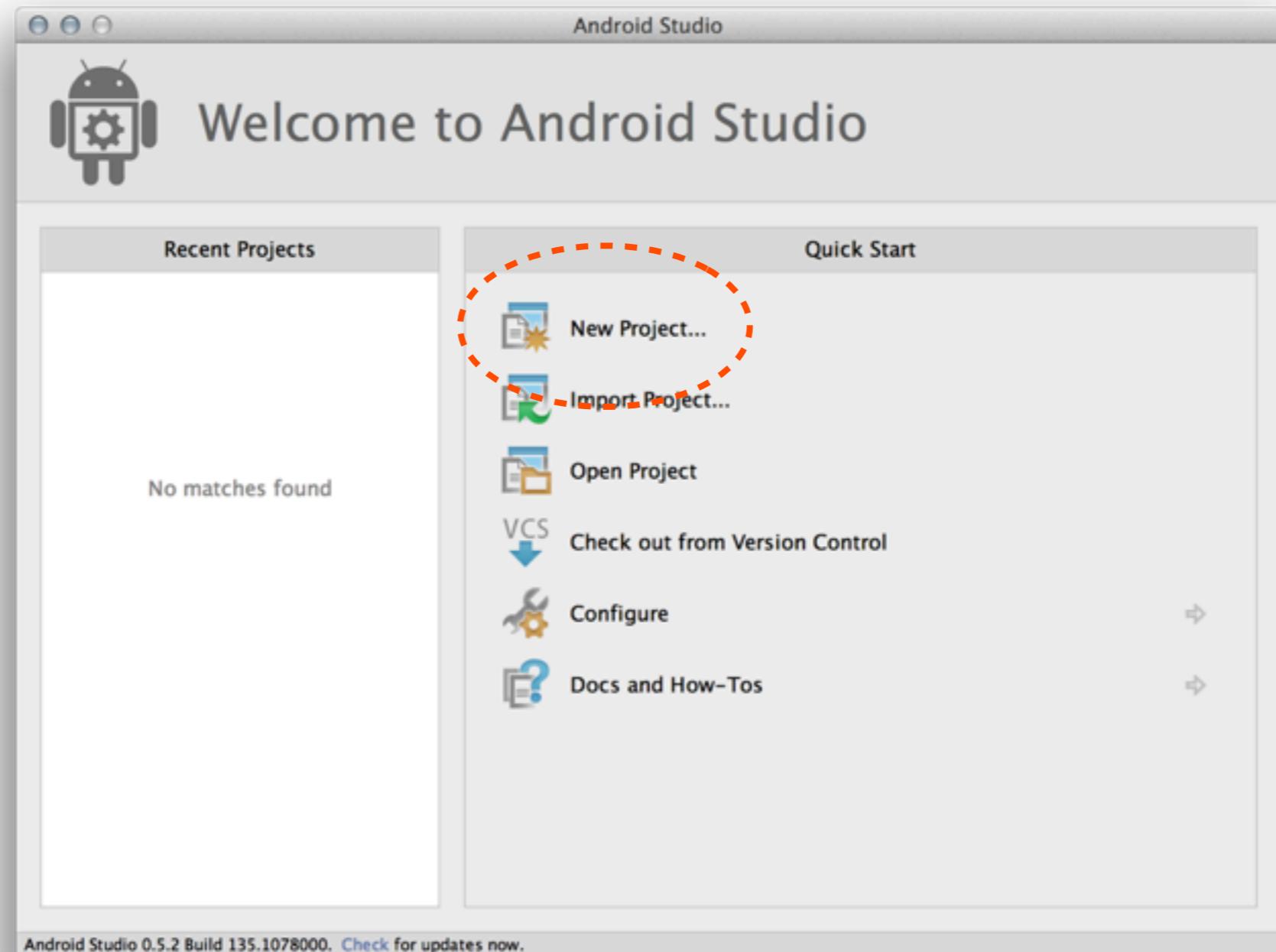
# Beispielprojekt III



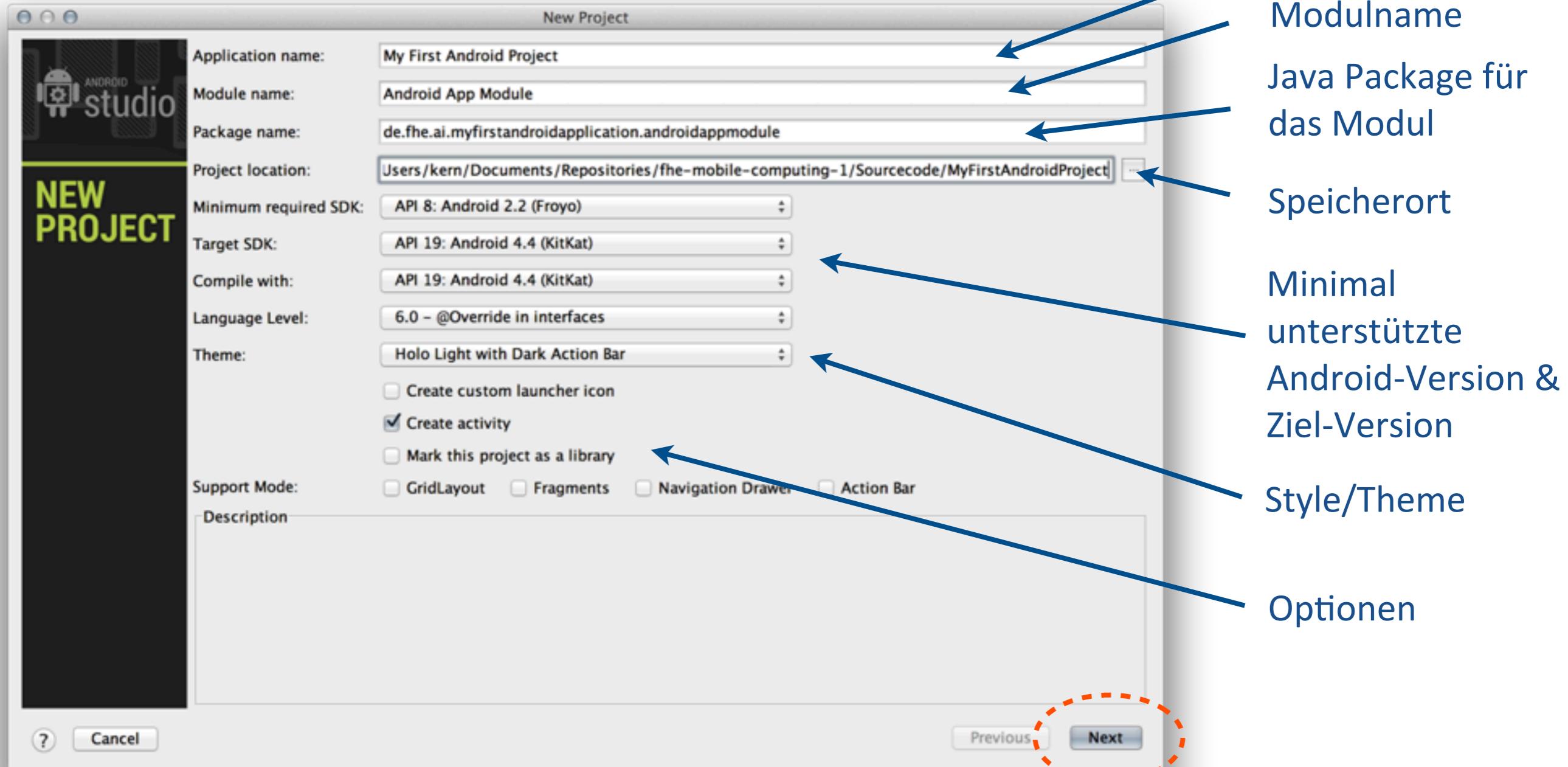
- Das Beispielprojekt wird sukzessive im Rahmen der Veranstaltung entstehen und in einem Repository zur Verfügung gestellt.

<https://bitbucket.org/skern/fhe-mobile-computing-1-examples>

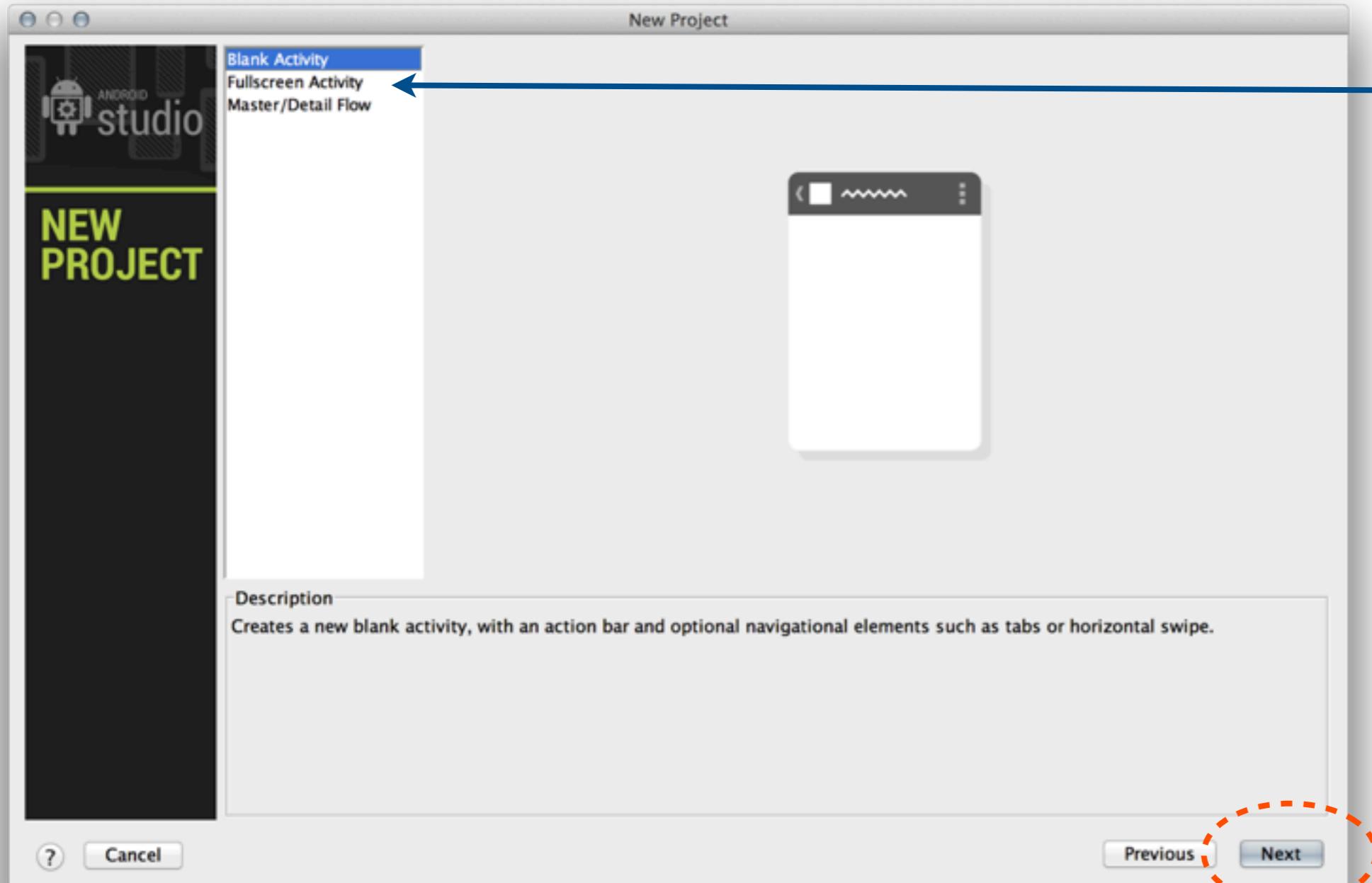
# Android Projekt erstellen I



# Android Projekt erstellen II

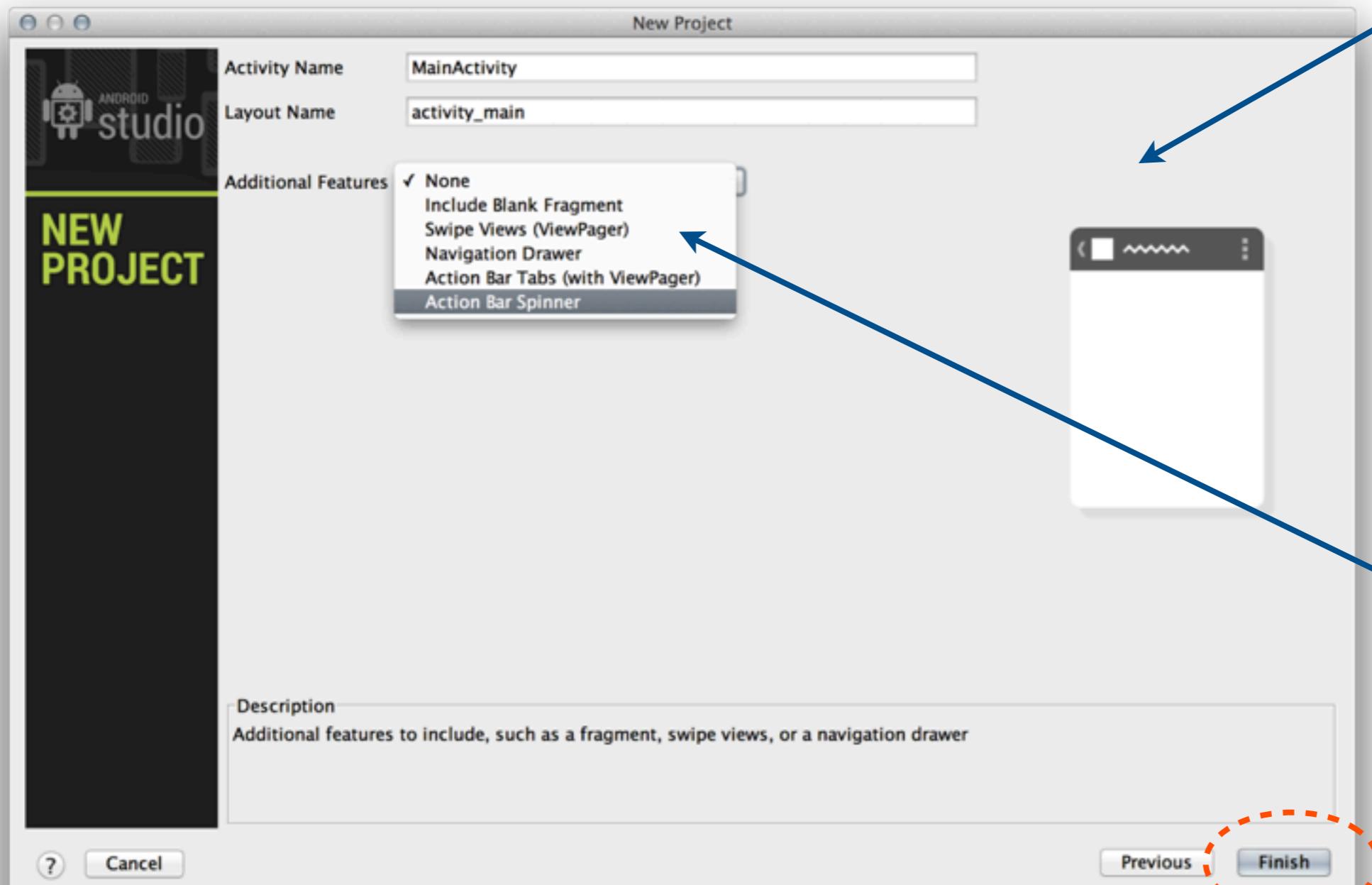


# Android Projekt erstellen III



Auswahl einer  
Projektvorlage

# Android Projekt erstellen IV



Falls „Create Activity“ gewählt wurde, erfolgt auf diesem Screen die Konfiguration dieser.

Auswahl gewünschter Features.

# Android Projekt erstellen V

The screenshot shows the Android Studio interface. The left pane displays the project structure for 'MyFirstAndroidProject'. The right pane shows the source code for 'MainActivity.java'.

```
package de.fhe.ai.myfirstrandroidapplication.androidappmodule;

import ...

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();
        if (id == R.id.action_settings) {
            return true;
        }
        return super.onOptionsItemSelected(item);
    }
}
```

A blue arrow points from the text 'Projektstruktur' to the project tree on the left. Another blue arrow points from the text 'Zugriff auf weitere Tool-Fenster' to the bottom status bar. A third blue arrow points from the text 'Start!' to the top right corner of the interface.

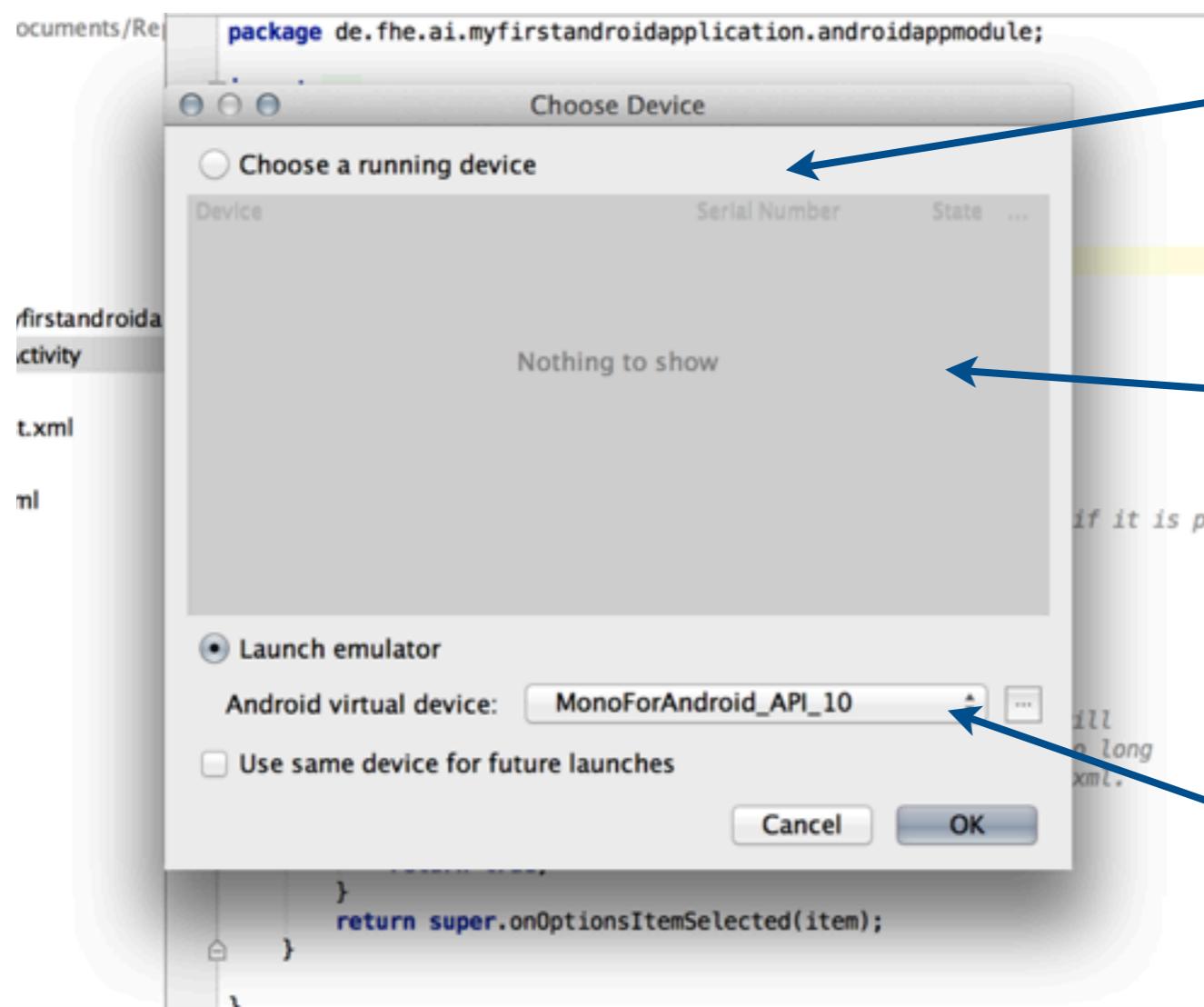
Projekt-  
struktur

Zugriff auf weitere Tool-Fenster

Sourcecode der  
erzeugten Activity

Start!

# Android Projekt erstellen V

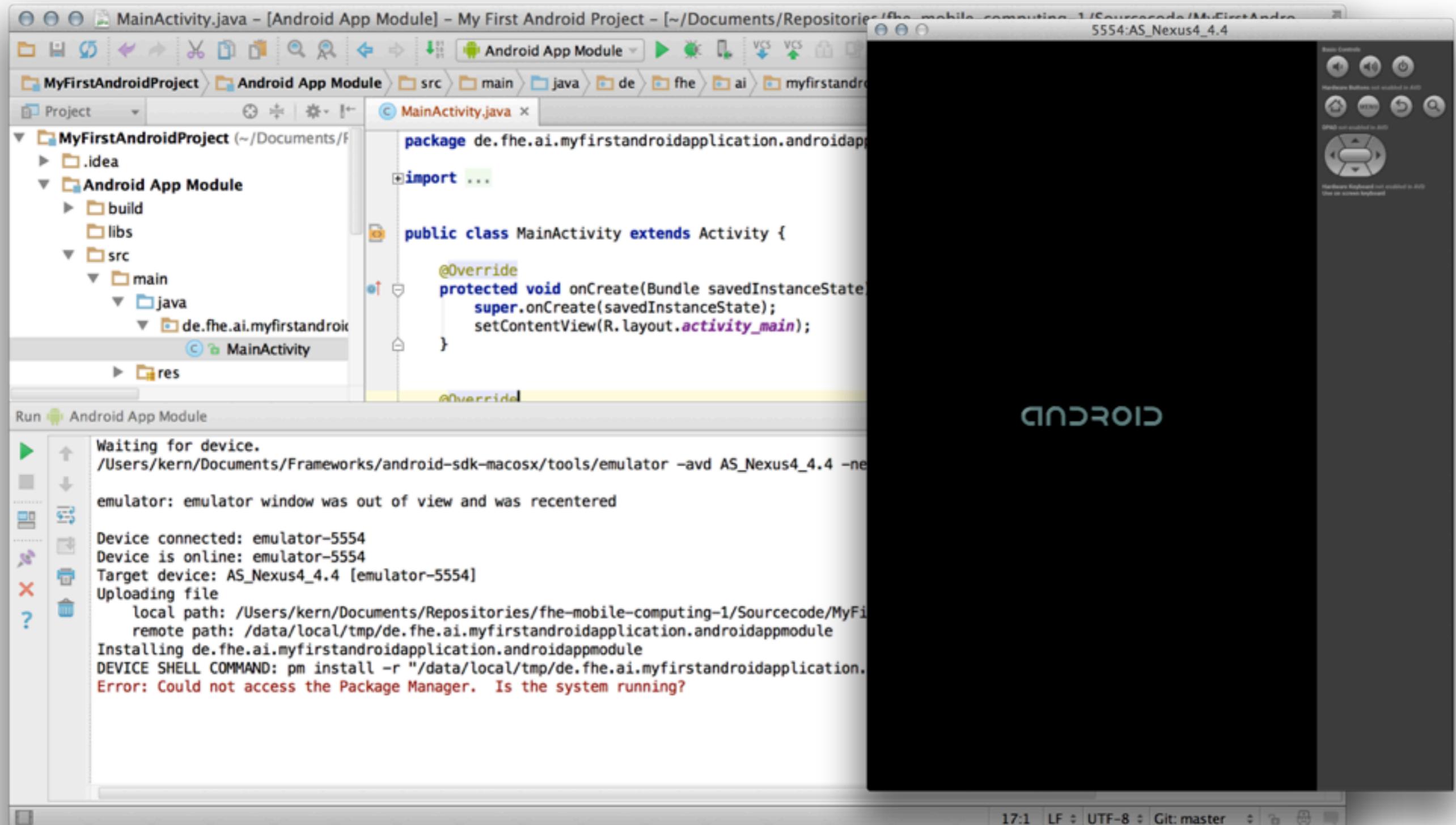


Auswahl des  
Zielgerätes

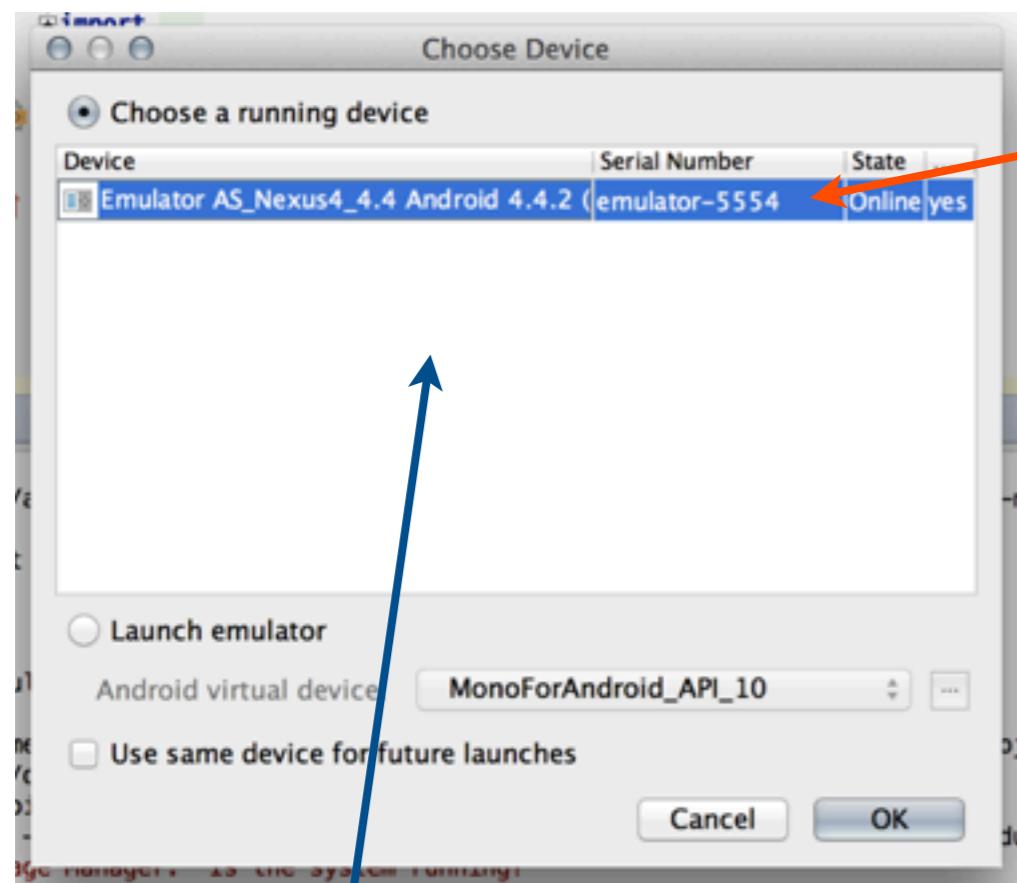
Liste der  
verfügbarer  
Android-Geräte  
(USB oder  
Emulator)

Verfügbare  
Emulatoren

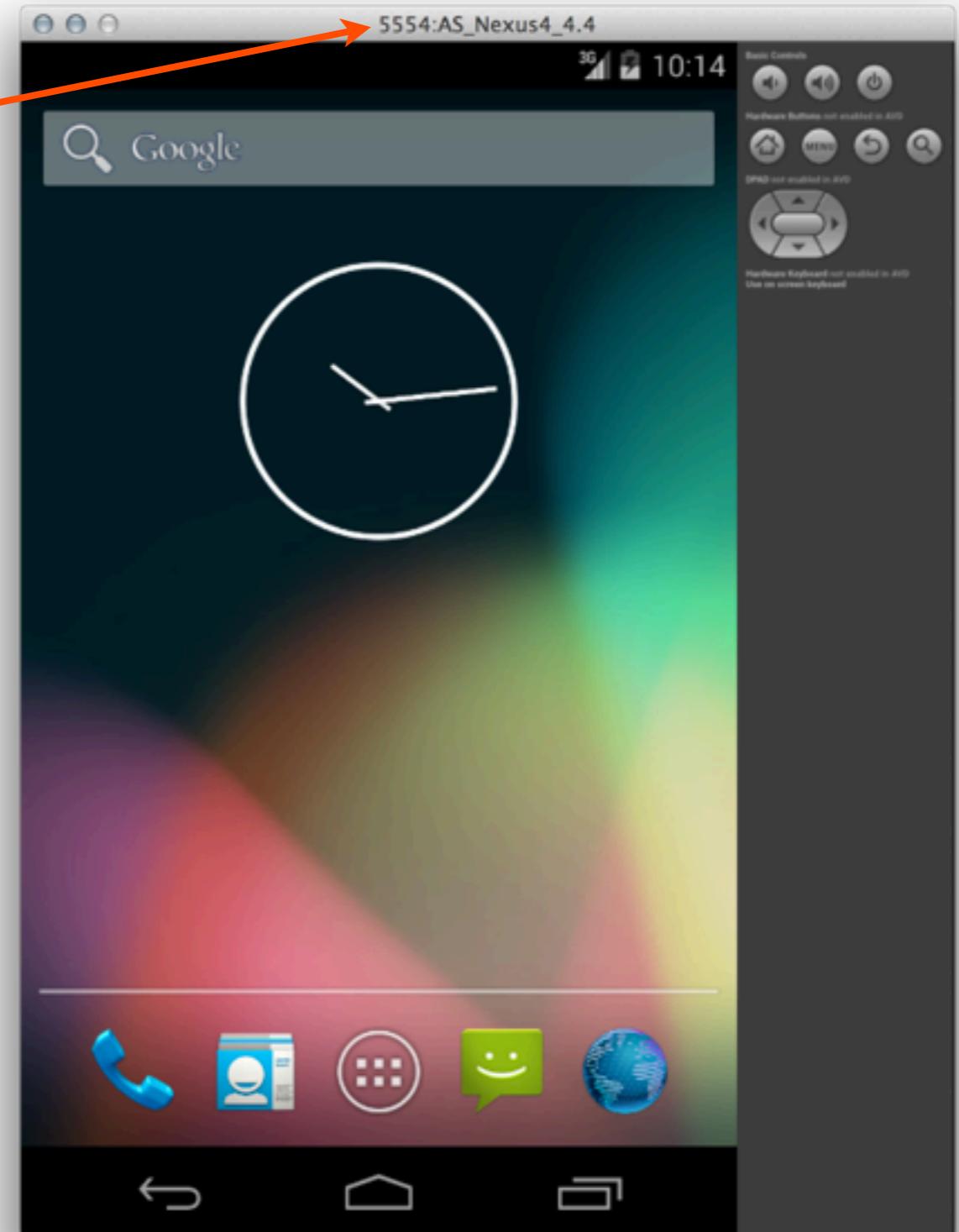
# Android Projekt erstellen VI



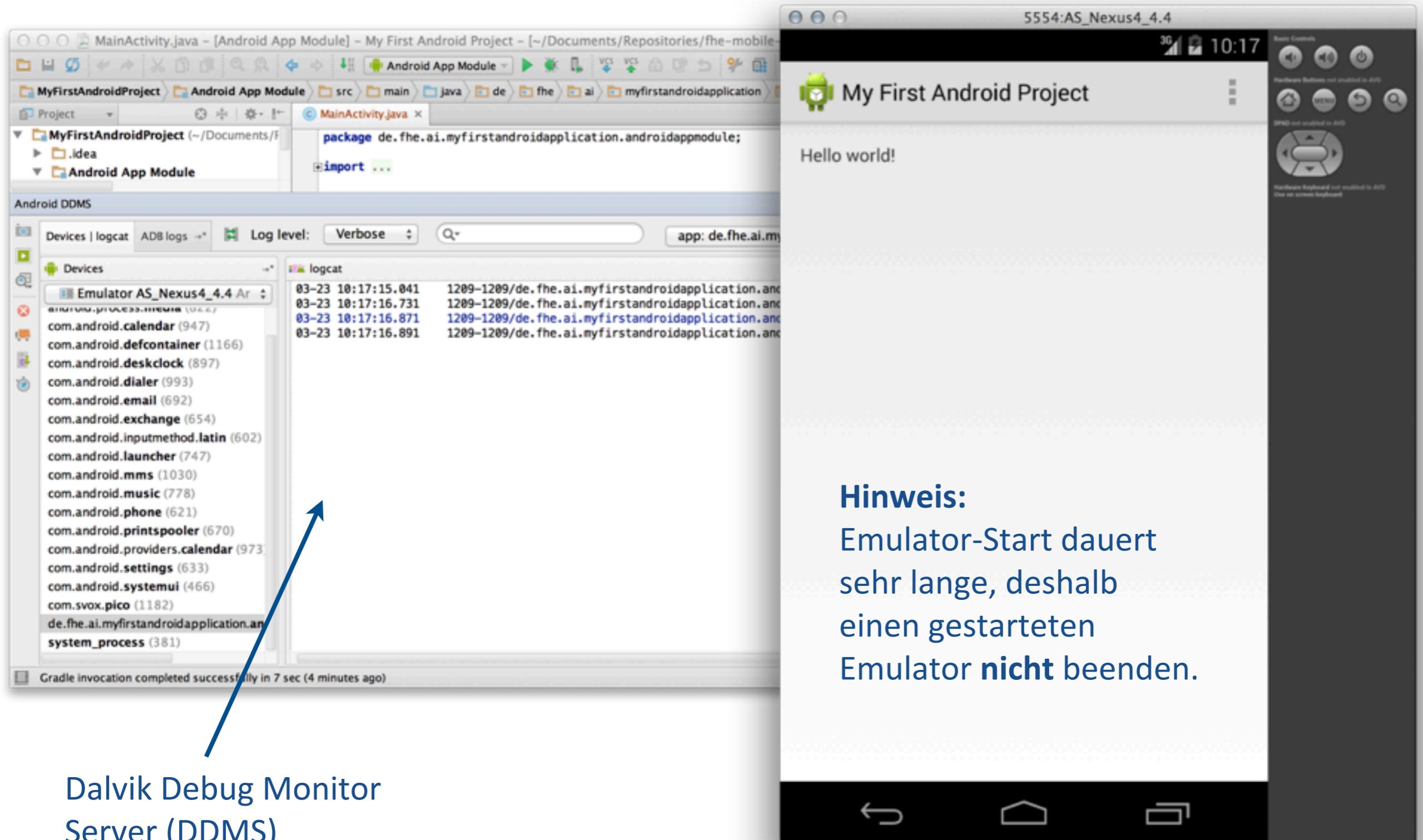
# Android Projekt erstellen VII



Neustart der Anwendung  
und Auswahl des  
laufenden Emulators

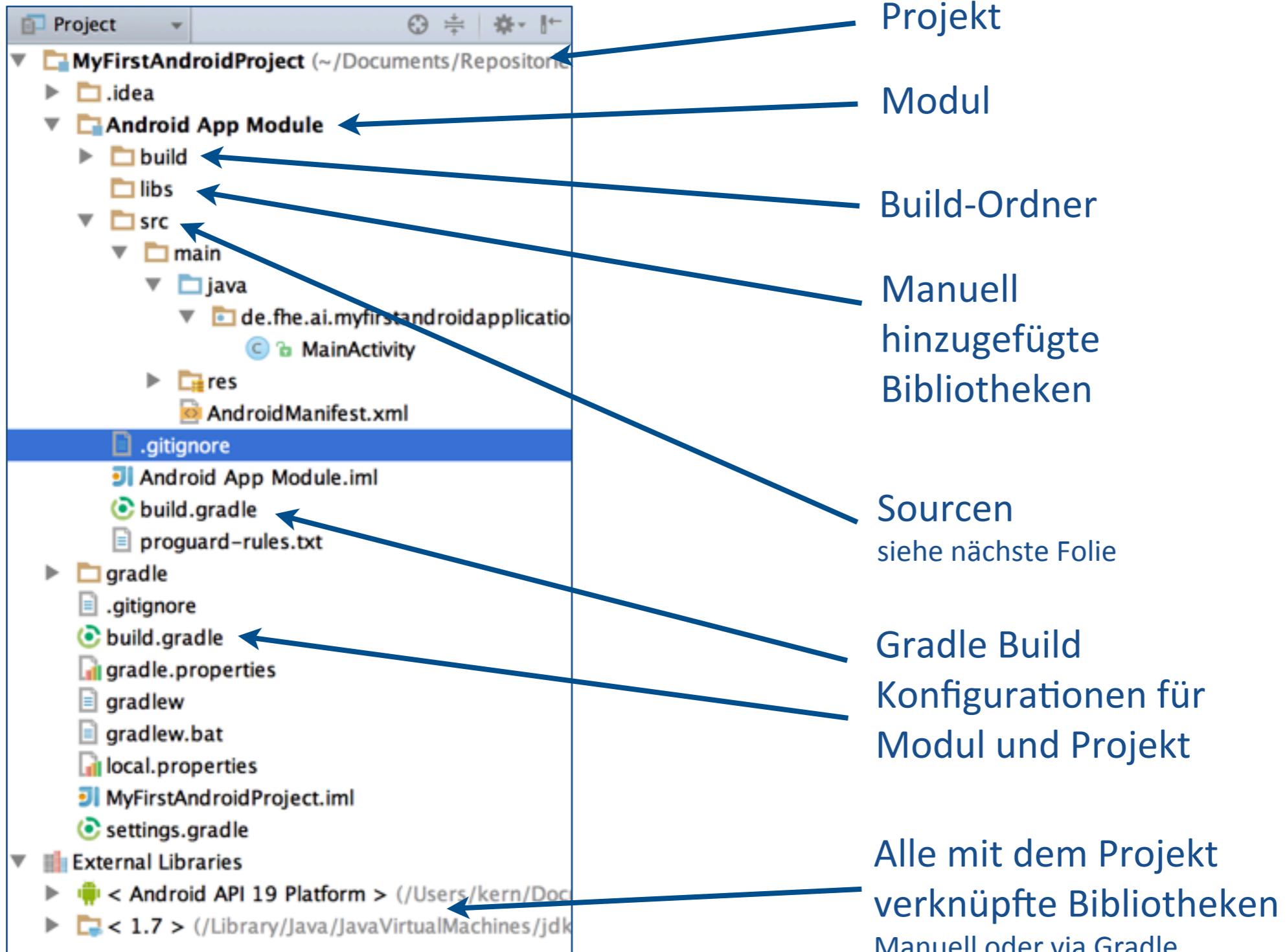


# Android Projekt erstellen IX

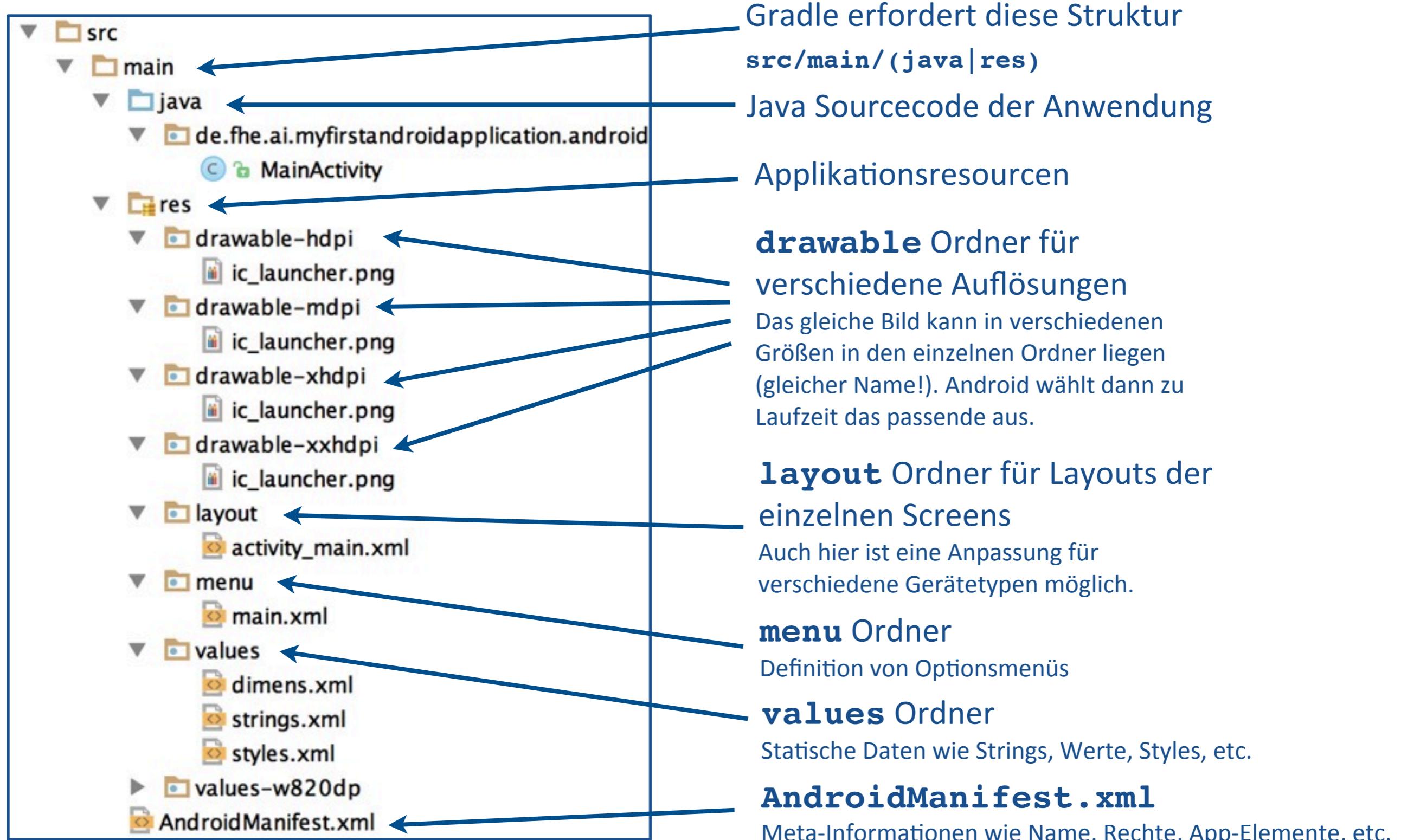


- Genymotion
  - <http://www.genymotion.com/>
- Virtual Box & Android x86 Image
  - Virtual Box: <https://www.virtualbox.org/>
  - x86 Image: <http://www.android-x86.org/download>
- Micro-Controller mit Android
  - z.B. <http://cubieboard.org/>
- Ein echtes Gerät ;-)

# Android Projektstruktur I



# Android Projektstruktur II



- Über Suffixe der entsprechenden Ordner können Bilder, Layouts, Styles - im Grund alle Anwendungsressourcen - gerätespezifisch erstellt und abgelegt werden
- Android wählt dann zu Laufzeit die passenden Ressourcen aus

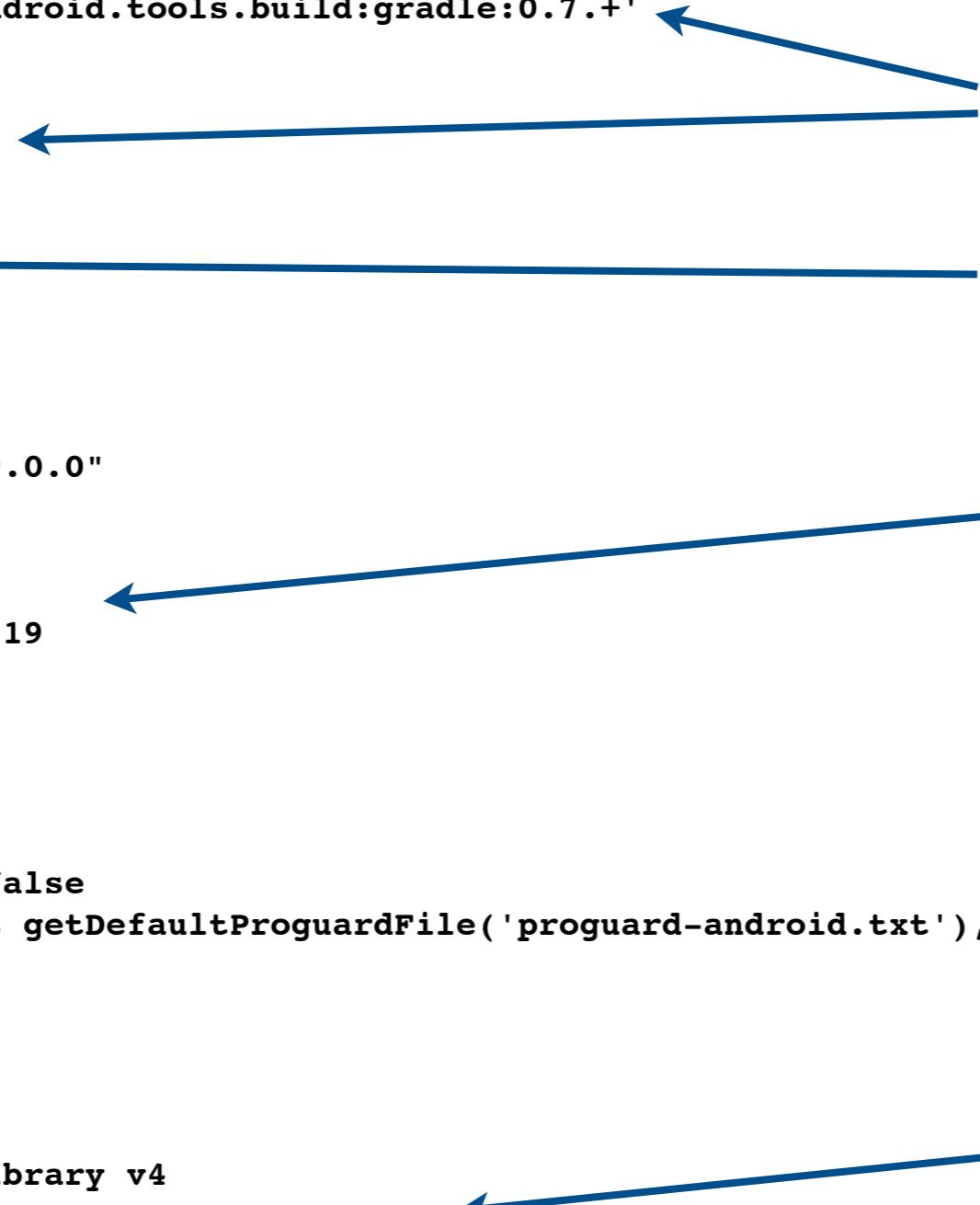
```
res/layout/my_layout.xml           // layout for normal screen size ("default")
res/layout-small/my_layout.xml     // layout for small screen size
res/layout-large/my_layout.xml     // layout for large screen size
res/layout-xlarge/my_layout.xml    // layout for extra large screen size
res/layout-xlarge-land/my_layout.xml // layout for extra large in landscape orientation

res/drawable-mdpi/my_icon.png      // bitmap for medium density
res/drawable-hdpi/my_icon.png      // bitmap for high density
res/drawable-xhdpi/my_icon.png     // bitmap for extra high density
```

- Vor Android Studio war zum Erstellen/Bauen einer Android-Anwendung Eclipse mit dem Android SDK/Android Plugin notwendig
  - Schlecht für automatisierte Deployments
- Deshalb Wechsel zu einer Build-Tool a la **make** oder **Ant**
  - Gradle: <http://www.gradle.org/>
  - Gradle & Android: <http://www.gradleware.com/resources/tech/android>
- Gradle erlaubt neben dem Erstellen von Anwendung auch das Verwalten von externen Bibliotheken

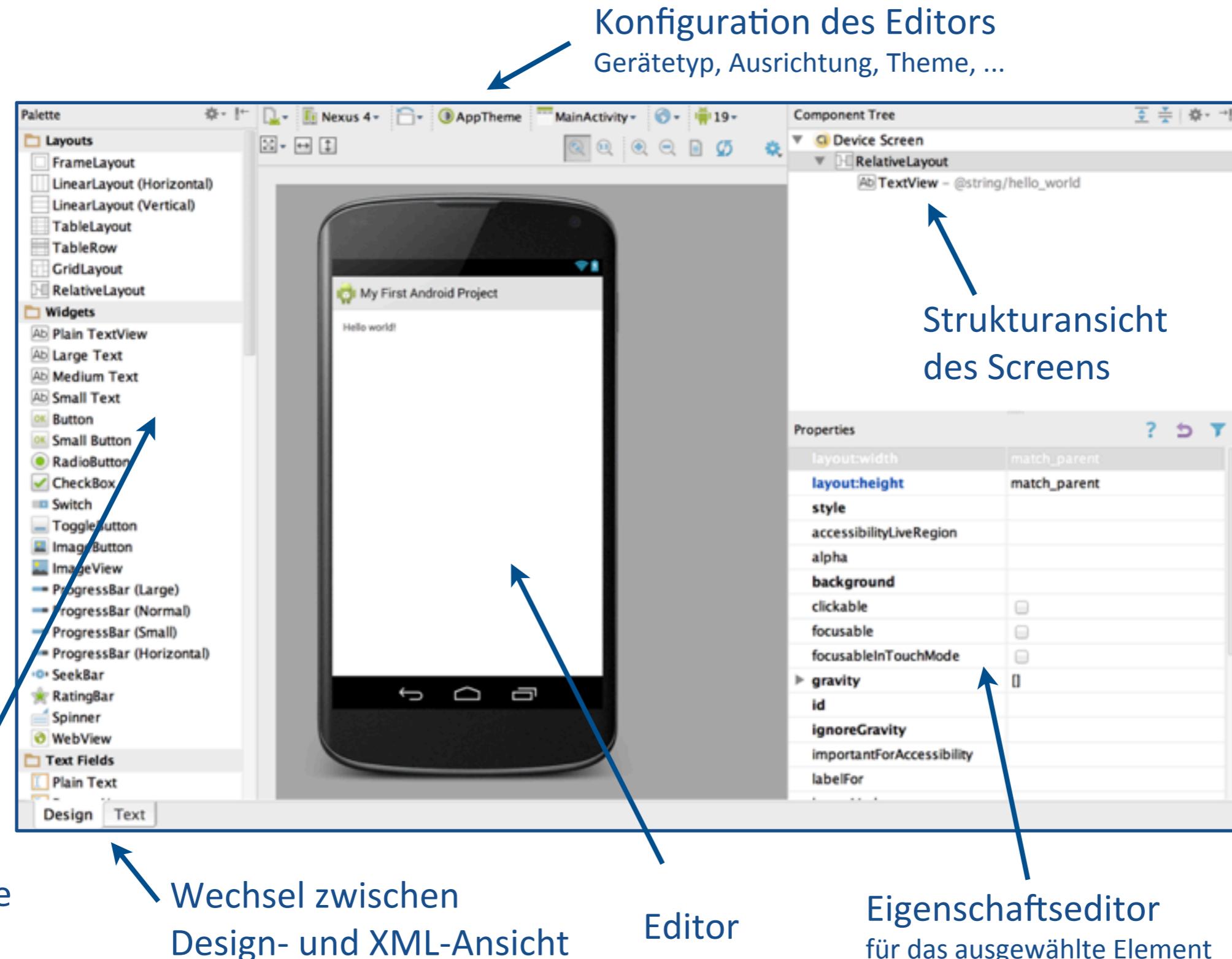
# Gradle Build File

```
buildscript {  
    repositories {  
        mavenCentral()  
    }  
    dependencies {  
        classpath 'com.android.tools.build:gradle:0.7.+'  
    }  
}  
apply plugin: 'android'  
  
repositories {  
    mavenCentral()  
}  
  
android {  
    compileSdkVersion 19  
    buildToolsVersion "19.0.0"  
  
    defaultConfig {  
        minSdkVersion 16  
        targetSdkVersion 19  
        versionCode 1  
        versionName "1.0"  
    }  
    buildTypes {  
        release {  
            runProguard false  
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.txt'  
        }  
    }  
}  
  
dependencies {  
    // Android Support Library v4  
    compile 'com.android.support:support-v4:+'  
}
```



- Android nutzt XML zur Beschreibung der Layouts einzelner Screens.
  - Alternativ können Layouts auch vollständig im Programmcode erstellt werden, was aber sehr mühselig ist.
- Diese XML-Dokumente definieren, welche Elemente auf einem Screen enthalten und wie diese angeordnet/ verschachtelt sind.
- Android Studio bietet einen grafischen Editor zu Erstellung dieser Layout-Dokumente.
- -> Definition = XML, Dynamische Anpassung = Java

# Layout Editor

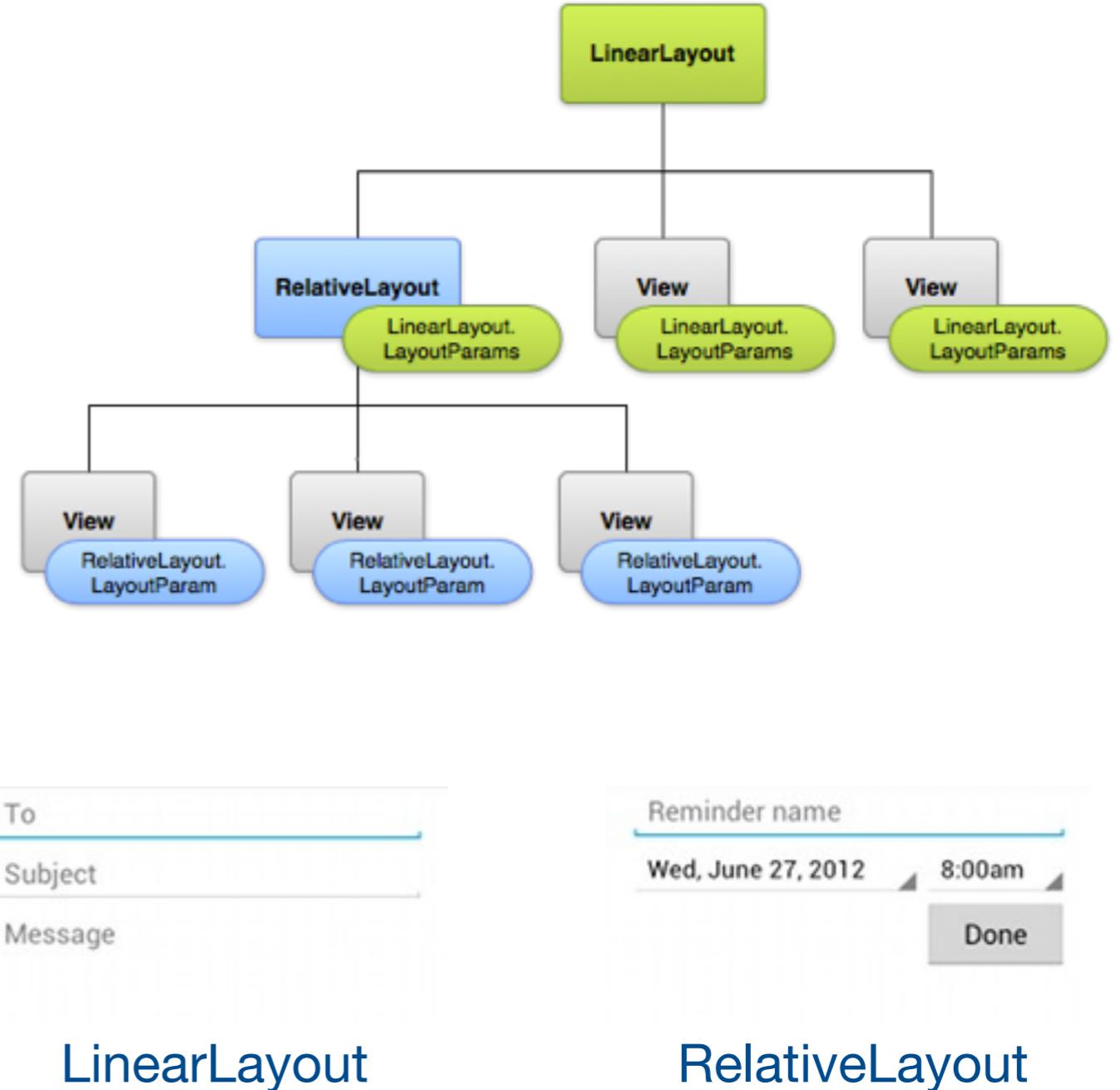


# XML Ansicht des Layouts

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:paddingLeft="64dp"  
    android:paddingRight="64dp"  
    android:paddingTop="16dp"  
    android:paddingBottom="16dp"  
    tools:context="de.fhe.ai.myfirstandroidapplication.androidappmodule.MainActivity">  
  
    <TextView  
        android:text="Hello world!"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content" />  
  
</RelativeLayout>
```

# Layouts

- Vorgefertigte Widgets (Basisklasse `View`) wie Buttons, Listen, etc.
- Hierarchie von Views beschreibt den Bildschirminhalt
- Layouts definieren die Anordnung der Views
- Definition via XML oder via Java Code



<http://developer.android.com/guide/topics/ui/layout-objects.html>

- Um zur Laufzeit Änderungen an den Elementen des Screens vornehmen zu können, muss man via Java-Code darauf zugreifen.
- Wie erfolgt also die Verknüpfung der in XML definierten Elemente mit dem Code?

## 1. Vergabe von eindeutigen IDs im XML

- Dokumente im res-Ordner erhalten ebenfalls IDs

## 2. „Mapping-Klasse“ `R.java`

- Wird durch die IDE automatisch generiert

## Auszug aus **MainActivity**

```
public class MainActivity extends Activity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState)  
    {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        TextView tv = (TextView)this.findViewById( R.id.myTextView );  
  
        tv.setText( "New Text" );  
    }  
}
```

Referenzieren des gesamten Layouts

**setContentView()**  
liest das XML-Layout und erzeugt einen Objektbaum

**3. findViewById()**  
liefert ein Objekt von Typ **View** zurück - deshalb Casten auf **TextView**

**2. Referenzieren der ID via R-Klasse**

**1. Definition einer ID für den TextView**

## Auszug aus **activity\_main.xml**

```
<TextView  
    android:text="Hello world!"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:id="@+id/myTextView" />
```

```
public final class R {  
    public static final class attr {  
    }  
    public static final class dimen {  
        public static final int activity_horizontal_margin=0x7f040000;  
        public static final int activity_vertical_margin=0x7f040001;  
    }  
    public static final class drawable { ←  
        public static final int ic_launcher=0x7f020000;  
    }  
    public static final class id { ←  
        public static final int action_settings=0x7f080001;  
        public static final int myTextView=0x7f080000;  
    }  
    public static final class layout { ←  
        public static final int activity_main=0x7f030000;  
    }  
    public static final class menu { ←  
        public static final int main=0x7f070000;  
    }  
    public static final class string {  
        public static final int action_settings=0x7f050000;  
        public static final int app_name=0x7f050001;  
        public static final int hello_world=0x7f050002;  
    }  
    public static final class style {  
        /** Customize your theme here.  
         */  
        public static final int AppTheme=0x7f060000;  
    }  
}
```

Innere Klassen für die einzelnen Resourcentypen (Layout, Menü, Drawable, IDs, ...)

Diese inneren Klassen enthalten dann Konstanten für die einzelnen Elemente.

## Beispiele für Referenzierung

Resource	Referenz in Java	Referenz in XML
<code>res/layout/main.xml</code>	<code>R.layout.main</code>	<code>@layout/main</code>
<code>res/drawable-hdpi/icon.png</code>	<code>R.drawable.icon</code>	<code>@drawable/icon</code>
<code>@+id/home_button</code>	<code>R.id.home_button</code>	<code>@id/home_button</code>
<code>&lt;string name="hello" /&gt;</code>	<code>R.string.hello</code>	<code>@string/hello</code>

- DIE Haupt-Komponenten einer Android-Applikation
- Dient der Interaktion mit dem Nutzer
- Vergleichbar mit den Seiten einer Website
- 1 Activity = 1 Screen (wenn man Fragmente ignoriert)
- Zugehöriges XML-File für die UI
- Meist eine oder mehrere Activities pro Applikation
- Wird beendet, wenn der „Back“-Button gedrückt wird!
- Wechsel in eine andere Applikation führt zum „Stopped“-Zustand

- Basisklasse `android.app.Activity`
- Jede Activity in einer Applikation muss in der **AndroidManifest.xml** deklariert werden

```
<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="My First Android Project"
    android:theme="@style/AppTheme" >

    <activity
        android:name="de.fhe.ai.myfirstrandroidapplication.androidappmodule.MainActivity"
        android:label="My First Android Project" >

        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

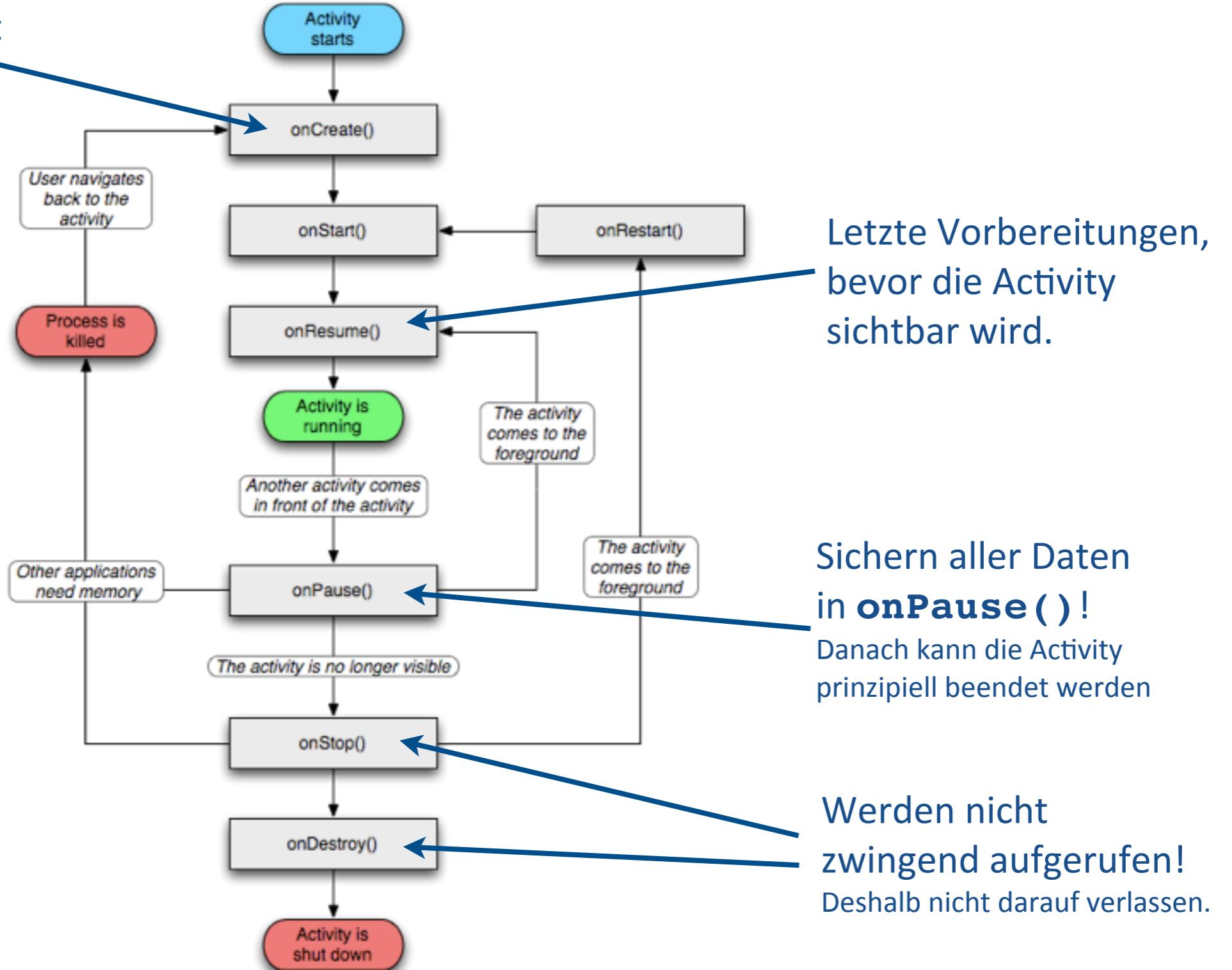
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>

    </activity>
</application>
```

Auszug aus **AndroidManifest.xml**

# Activities III - Lebenszyklus

Laden der Layout  
XML-Datei



## Beispiel

```
import android.app.Activity;

public class MyActivity extends Activity
{
    protected void onCreate(Bundle savedInstanceState) { ... }

    protected void onStart() { ... }

    protected void onRestart() { ... }

    protected void onResume() { ... }

    protected void onPause() { ... }

    protected void onStop() { ... }

    protected void onDestroy() { ... }
}
```

**Wichtig:** Lebenszyklus-Methoden müssen die Implementierung der Super-Klasse aufrufen!

## Schritte zum Anlegen einer neuen Activity

1. Java-Klasse anlegen, welche von **android.app.Activity** erbt
2. XML-Datei für das Layout im Ordner **res/layout** anlegen
3. Die Activity in der **AndroidManifest.xml** eintragen
4. In der Activity-Klasse die **onCreate(...)** Methode überschreiben und die Verknüpfung zur Layout XML-Datei herstellen

- Klasse `android.util.Log`
- Statische Methoden für Error ( `e()` ), Warning ( `w()` ), Info ( `i()` ), etc.
- Parameter
  - `tag` : String - Identifier (Klassenkonstante sinnvoll!)
  - `msg`: String - Die Log-Meldung
- Ansicht DDMS Console

# Logging - Beispiel

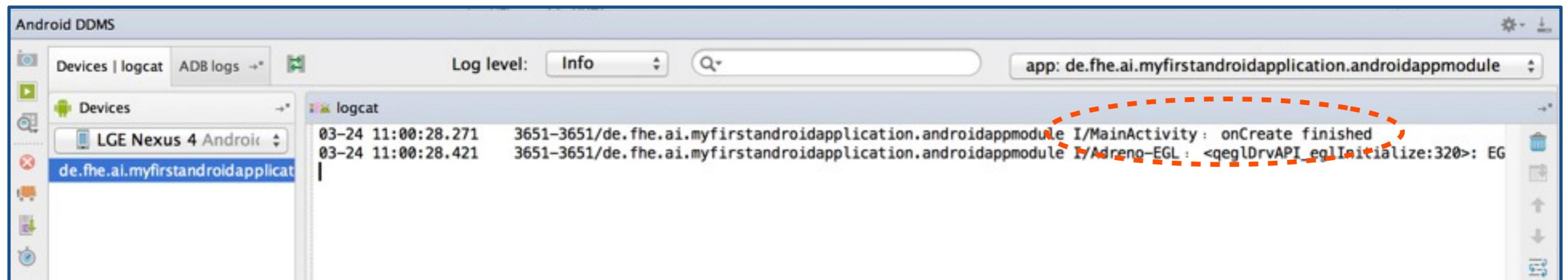
```
public class MainActivity extends Activity {

    private static final String LOG_TAG = MainActivity.class.getSimpleName();

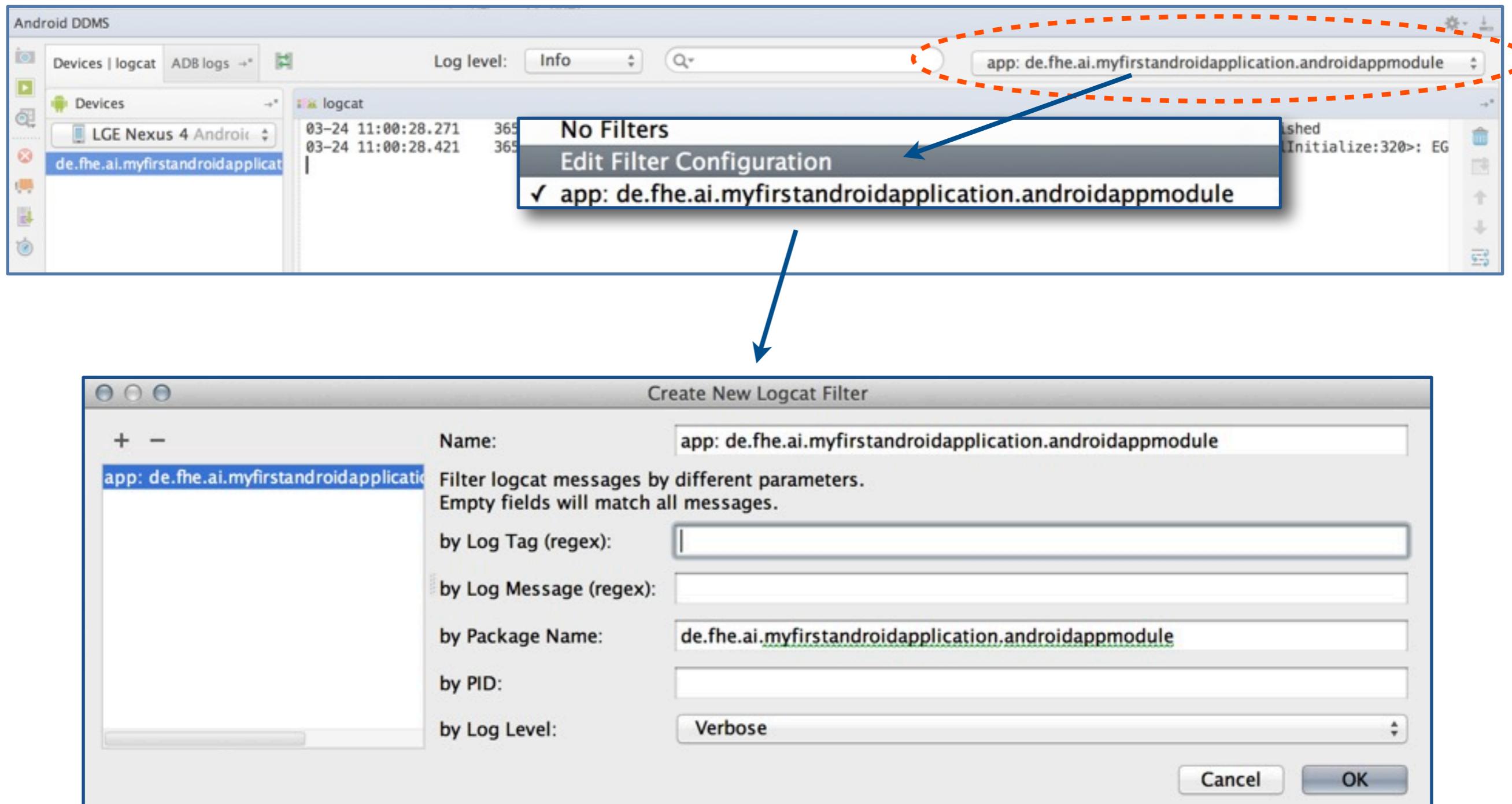
    // Oder
    // private static final String LOG_TAG = "Mein Log Tag";

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Log.i( LOG_TAG, "onCreate finished" );
    }
}
```



# Logging - Filter



- Android Projekt erstellen
- Main Activity erstellen
- Callback-Methoden des Lebenszyklus implementieren
- Applikation ausführen
- Lebenszyklus der Activity über die Log-Meldungen nachverfolgen