Programmierung mobiler Geräte

SoSe 2015

Native Entwicklung mit Android

Einführung

Markus Berg
Hochschule Wismar
Fakultät für Ingenieurwissenschaften
Bereich Elektrotechnik und Informatik
http://mmberg.net

Status Quo

- WebApps
- Hybride Apps
  - Zugriff auf Gerätefunktionen
  - Allerdings geringere Performance
  - Kein natives Look & Feel
  - Webtechnologien
    - Erschwerte Fehlersuche (kein Compiler)
    - Keine Codevervollständigung
  - Läuft auf allen Plattformen
    - Keine Anpassungen für verschiedene Geräte/ Betriebssysteme notwendig

Native Apps

- Werden in der Programmiersprache des Zielgerätes programmiert
  - Windows → C#
  - iOS → Objective C
  - Android → Java
- An die Zielplattform angepasst und optimiert

Ziel für heute

- Einführung in die Android Systemarchitektur
- Überblick über die Komponenten einer Android-Anwendung
- Programmieren einer ersten kleinen App

Android

- Fokus auf Smartphones und Tablets
- 2003: Aufgekauft von Google
- 2008: erstes Release
- Aktuelle Version: 5.1 (Lollipop)
- Basiert auf Linux-Kernel
  - Speicherverwaltung
  - Prozessverwaltung
  - Kommunikation/Netzwerk/Multimedia
  - Gerätetreiber

Dalvik / Android Runtime

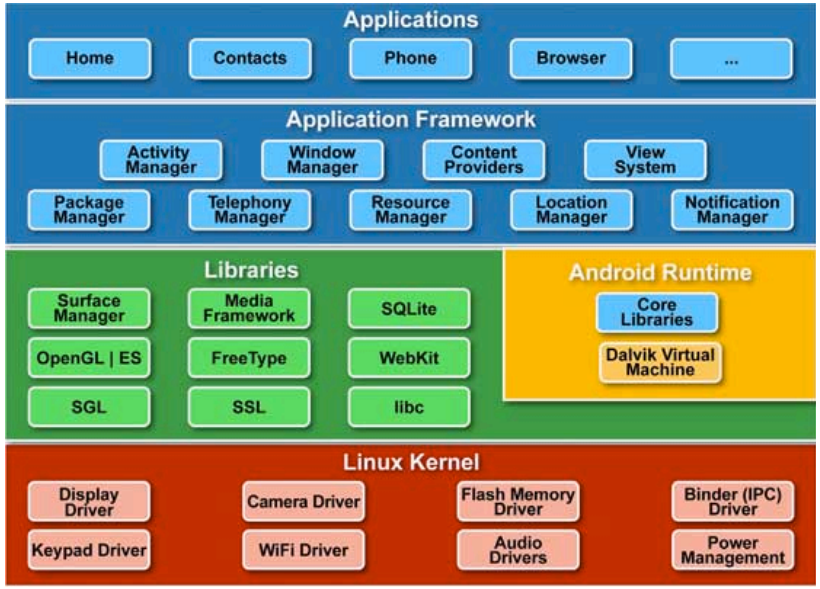
- Laufzeitumgebung (Java Virtual Machine)
  - Ähnelt der Java SE JVM
  - Führt ebenfalls Bytecode aus
    - Jedoch inkompatibler Bytecode (Dex-Bytecode), d.h. „normale“ Java-Programme laufen nicht auf Dalvik und umgekehrt
    - Java-Quellcode → Java-Bytecode → Dex-Bytecode
    - Unterschiedliche Prozessorarchitektur
  - Bis 2014: Dalvik Virtual Machine
    - Just-in-time-Compiler
  - 2014 ersetzt durch *Android Runtime (ART)*
    - *Führt identischen Bytecode aus*
    - Ahead-of-time-Compiler (wie schon zu C-Zeiten)
      - Nicht mehr plattformunabhängig
    - Bessere Garbage Collection
- Jede Anwendung läuft in eigener VM

Android

- Modular aufgebaut
  - Sämtliche Anwendungen können ausgetauscht werden
    - Inkl. SMS-App, Telefon-App
  - Ausnahme: die virtuelle Maschine und das Basissystem selbst
- Ressourcenbegrenzte Umgebung
  - CPU, RAM, Akku
  - Daher optimierte Laufzeitumgebung

Systemarchitektur von Android

- Verschiedene Ebenen
  - Low-Level (Betriebssystemebene; Kernel) bis High-Level (Applications)
    - Kernel
    - System-Bibliotheken
      - Nativ (C/C++)
    - Android Runtime
      - Mit Core-Libraries (Java)
    - Android Framework
      - Java
      - Nutzbar durch Apps
    - Apps (d.h. die Standardanwendungen)



http://www.tutorialspoint.com/android/images/android_architecture.jpg

Kernel

- Speicher-/Prozessmanagement
- Security
- Dateizugriff
- Netzwerkzugriff
- Treiber
- Powermanagement
- Interprozesskommunikation (IPC)

Libraries & Runtime

- Displayzugriff
- WebKIT (Browser)
- OpenGL
- In-Memory Database
- Runtime (ART/Dalvik)
  - Virtuelle Maschine
  - Core-Java-Bibliotheken
    - Datenstrukturen
    - Dateizugriff
    - Lebenszyklus
    - ...

Application Framework

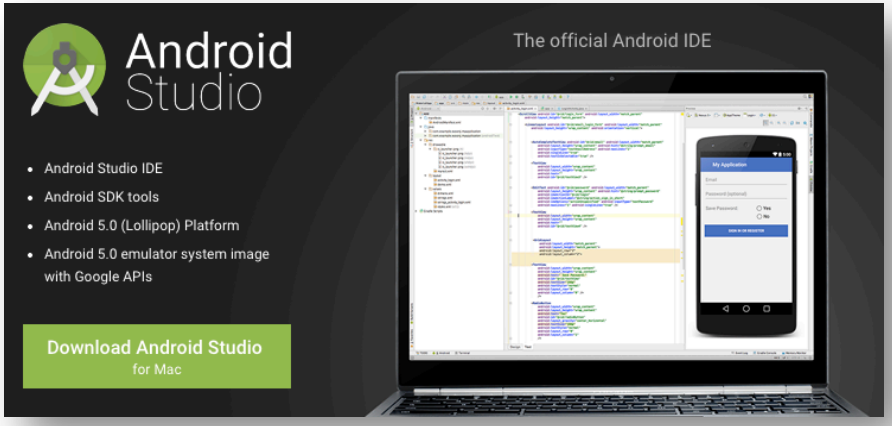
- Zur Nutzung durch die Anwendungen
  - Window Manager
    - Verwalten und Darstellen von Menüs, Dialogen, Notification Bar bzw. Hauptfenster einer Anwendung
  - View System
    - UI-Objekte: Darstellen von Icons, Buttons
  - Package Manager
    - Übersicht aller installierten Anwendungen
  - Activity-Manager
    - Verwaltung des Lebenszyklus einer Anwendung
    - Navigationsstack
  - ...

Applications

- Built-in Apps
  - Browser
  - Mailprogramm
  - SMS
  - ...

Entwicklung: SDK & IDE

- SDK
  - Zum Entwickeln von Android-Apps
  - Plattform-Bibliotheken
  - Tools/Entwicklungswerkzeuge
  - Dokumentation
- Systemimages für Emulator
- IDE
  - Früher: Eclipse mit ADT Plugin
  - Seit Ende 2014: Android Studio
    - Basiert auf IntelliJ
  - *(Haben wir bereits installiert)*



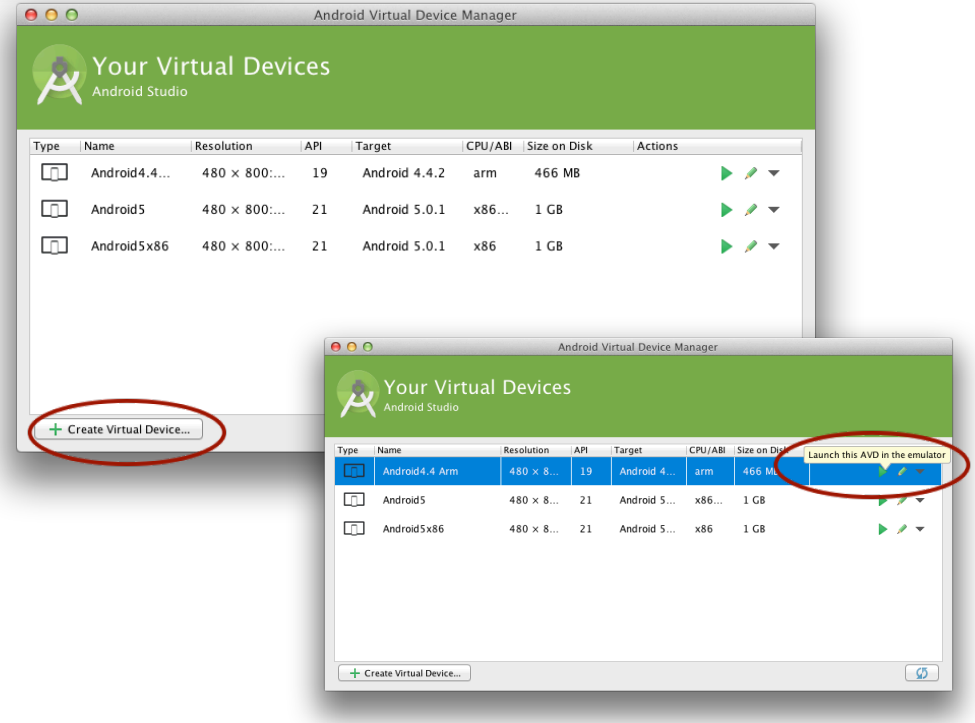
Emulator

- Zum Emulieren von verschiedenen Android Geräten
  - z.B. Tests verschiedener Bildschirmgrößen ohne Geräte kaufen zu müssen
  - z.B. Tests von verschiedenen Android Versionen
  - z.B. Simulation von Gerätevenents (low battery, new message) oder Zuständen (Geokoordinaten, langsame Internetverbindung)
  - Somit essentiell zur Entwicklung
  - Ersetzt jedoch nicht den Test auf ausgewählten echten Geräten (Emulator emuliert nicht alle Funktionalitäten, z.B. Bluetooth)
  - Größter Nachteil: Langsam!
    - Auf modernen Prozessoren Hardwarebeschleunigung aktivieren (Intel HAXM)
    - Alternative: Genymotion
- Sgn. Virtuelle Geräte (Virtual Devices, AVD)
  - Konfigurierbar
    - Bestimmte Hardware
      - CPU, RAM, Displaygröße
    - Bestimmte Androidversion
- Verwaltet über AVD Manager

AVD einrichten

- Android Virtual Device Manager starten

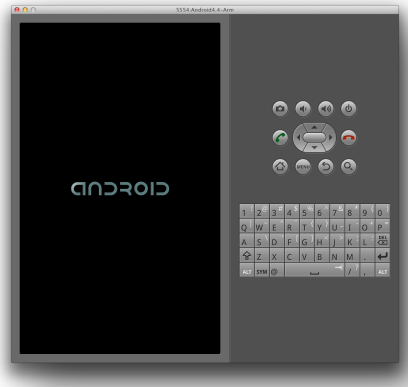
- Neues Gerät anlegen (bzw. starten falls schon angelegt)
  - Wenn kein HAXM verfügbar (z.B. bei Core2Duo), Gerät mit **ARM** anlegen, sonst **x86** (z.B. i5, i7)

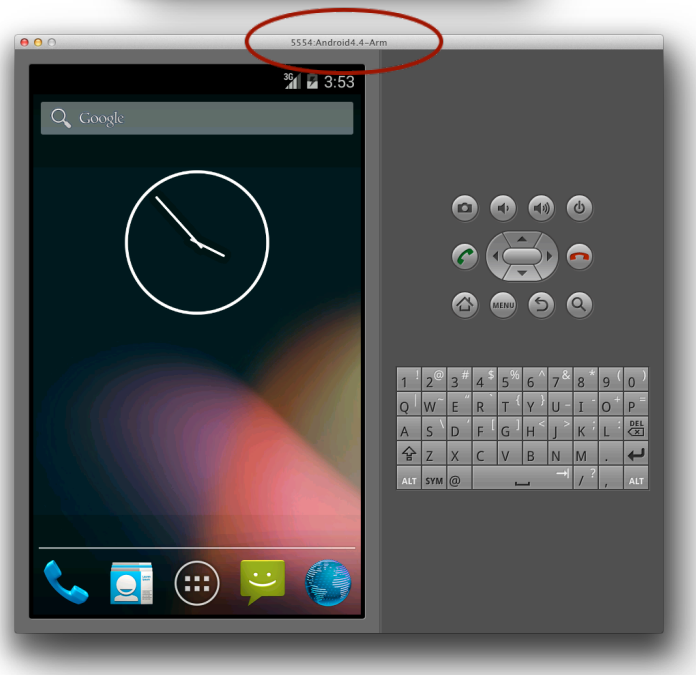


Emulator steuern

- Emulator (bzw. AVD) starten (dauert u.U. sehr lange)
- Verbinden per Telnet
  - Port in Titelzeile des Emulators sichtbar
  - Standard: 5554, d.h. `telnet localhost 5554`
  - Mögliche Befehle:

```
power capacity 50
sms send 0173123123 “Guten Tag!“
network speed (edge | full)
geo fix 53.889910 11.446331
```

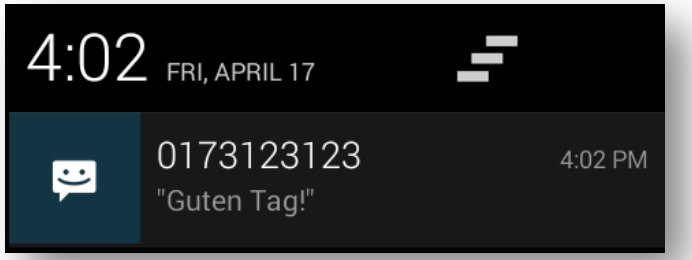




Emulator steuern

```
markus-bergs-mac-mini:~ markus$ telnet localhost 5554
Trying ::1...
telnet: connect to address ::1: Connection refused
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Android Console: type 'help' for a list of commands
OK
power capacity 50
OK
```


```
sms send 0173123123 "Guten Tag!"
OK
```



Wie immer: Moin Wismar!

- Ziel: Eine App, die den Text „Moin Wismar!“ anzeigt
  - Und schrittweise Erweiterung um ein User Interface

- Doch zunächst etwas Theorie...

Applikationslayout: Komponenten

- Grundlegende Einheit einer App
  - Legt Zweck und Einstiegspunkt (User, Event,...) fest
  - http://developer.android.com/guide/components/fundamentals.html
- **Activities**
  - GUI
  - Userinteraktion mit der App
- Services
  - Langlaufende Prozess im Hintergrund
  - Kein Userinterface
  - z.B. Musik abspielen obwohl andere App im Vordergrund
- Broadcast Receivers
  - Reagieren auf Geräteevents, d.h. Subscriber von Events (bzw. sgn. *Intents*)
  - z.B. Empfang einer SMS
- Content Providers
  - Daten speichern und zwischen Apps teilen

Komponente: Activity

- Repräsentiert eine bestimmte Aktivität, die der Anwender nutzen kann
  - Zeigt Daten an und reagiert auf Eingaben → Graphical User Interface (GUI)
  - Einzelner „Screen“ mit User Interface
  - Vergleichbar mit einem Fenster einer Desktopanwendung
    - Form (C#) oder JFrame (Java Swing)
  - z.B. neue Notiz anlegen oder Kontakt aus Telefonbuch auswählen
    - Oder selbst das Anzeigen von „Moin Wismar!“
  - App kann aus mehreren Activities bestehen
    - z.B. E-Mail-App: Liste der Mails anzeigen, Mail lesen, Mail schreiben
    - Activities können unabhängig aufgerufen werden, z.B. Klick auf Mailadresse im Browser öffnet Activity „Mail schreiben“ der E-Mail-App
  - Aufruf von Activities geschieht über *Intents* (es gibt keine main()-Funktion)

Vorteil von Activities

- Vermeiden von Redundanzen
  - Wiederverwendung über Applikationsgrenzen hinaus
  - Um ein Foto per Mail zu verschicken, muss im Mailprogramm keine Fotofunktion eingebaut werden
    - Mailprogramm ruft Kamera-App auf und nimmt Foto entgegen
- User merkt nicht, dass er zwei Apps genutzt hat
  - Setzt voraus, dass Activities einheitlich designed sind
- Wenn User verschiedene Kamera-Apps hat, wird er aufgefordert eine auszuwählen

Intents

- Aktivitäten werden über asynchrone Nachrichten aufgerufen (Intents)
- Dies kann über Applikationsgrenzen hinaus passieren
- Nachricht kann eine Activity aktivieren
  - *Explicit Intent:* URL mit Chrome anzeigen, GMail
  - *Implicit Intent:* URL anzeigen, Mail schreiben, Bild anzeigen, ...
    - D.h. *view* or *send* something (URI)
    - Wenn mehrer Anwendungen diese Funktionalität besitzen, wird der Nutzer gefragt welche er nutzen möchte
      - *Wir erinnern uns an die share-Funktion des letzten Praktikums*

Manifest

- Alle Komponenten müssen der App im Manifest bekannt gemacht werden
- Darüber hinaus können hier Berechtigungen erteilt werden (z.B. ob die App auf‘s Internet zugreifen darf)
- *Intent Filters*: Deklaration welche Funktionalitäten (auf Activity-Ebene) von anderen Apps aufgerufen werden können
  - „Ich kann Bilder anzeigen“
- Minimal erforderliche Version der Android API
- Weitere Anforderungen (*uses-feature*)
  - z.B. Kamera
- Icon der App

Ressourcen

- Android Applikationen bestehen aus mehr als nur Quelltext
- Sgn. Ressourcen
  - Layout-Files
  - Bilder
  - Texte (Strings)
- Vorteile
  - Anpassen der Anwendung ohne Quelltext ändern zu müssen (z.B. Übersetzen in andere Sprache)
  - Verschiedene Layouts für verschiedene Displaygrößen

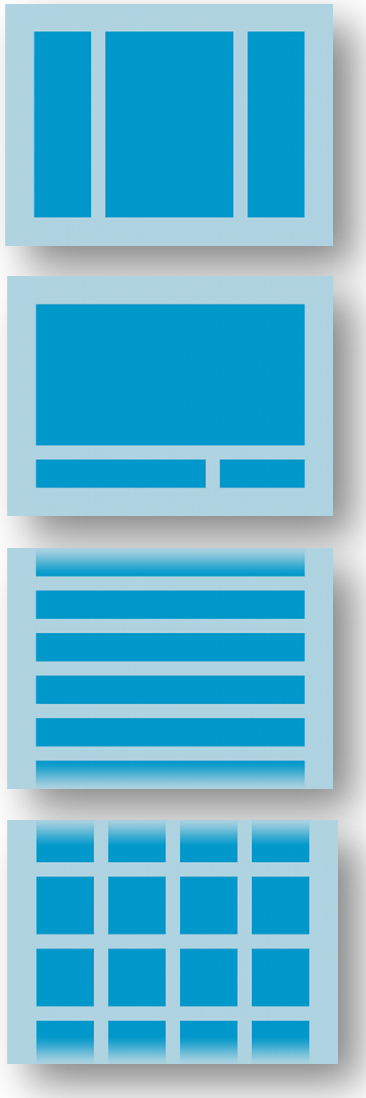
http://developer.android.com/guide/topics/resources/overview.html

Layout

- Definiert die visuelle Struktur
- Zwei Möglichkeiten
  - In XML deklarieren (empfohlen)
  - Im Quelltext deklarieren und zur Laufzeit instanziieren
- Mischung möglich: in XML deklarieren und zur Laufzeit verändern
- Vorteil XML: Trennung von Präsentation und Verhalten
  - Schlankerer Code
  - Leichte Anpassung des Layouts ohne Quelltext zu ändern
  - Somit kein Neukompilieren erforderlich
- Layoutdefinitionen sind Ressourcen und somit im Ordner `/res/layout/`
- Mehr Details: http://developer.android.com/guide/topics/ui/declaring-layout.html

Layouts

- Linear Layout
  - Alle Kinder werden in einer Richtung ausgerichtet
  - Vertikal oder horizontal
  - D.h. eine Zeile oder eine Spalte
- Relative Layout
  - Position wird relativ zu anderen Elementen angegeben
  - z.B. Unter X und links neben Y
- List View
  - Stellt eine scrollbare Liste dar
  - Dynamischer Inhalt: Automatisch aus Array oder anderer Quelle befüllbar
  - Performanter als Linear Layout (nur aktuell sichtbare Items werden geladen, nicht alle 100 einer Liste wenn nur 10 gleichzeitig sichtbar sind)
- Grid View
  - Zweidimensionales Raster
  - Ebenfalls dynamisch und performant



Projektstruktur

- manifests
  - **androidManifest.xml**
- java
  - net.mmberg.myApp
    - **MainActivity.java**
- res
  - drawable
  - values
    - **strings.xml**
  - Layout
    - **activity_main.xml**
  - ...

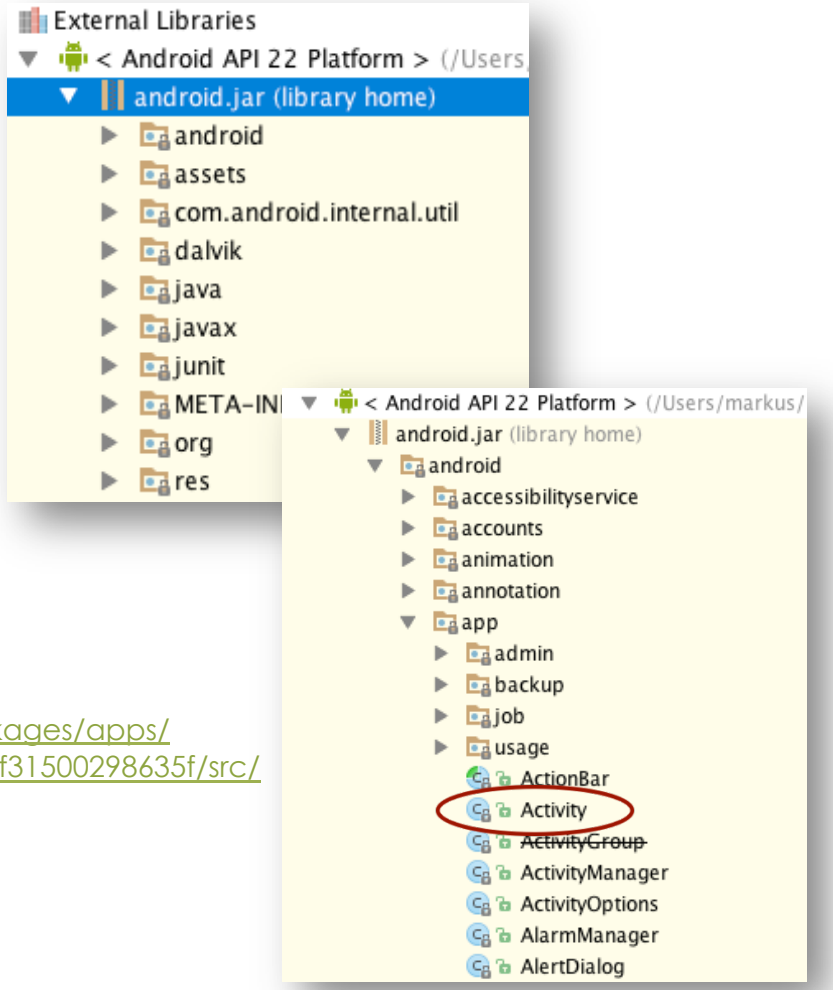
Manifest

Quellcode
Packages mit Java-Klassen

Ressourcen
Icons, Strings, Layouts

Android API: Ein Blick in den Quelltext

- Android ist OpenSource
  - Quelltext der API ist einsehbar
  - z.B. Quelltext der Activity-Klasse die Grundlage für alle Activities ist



- Auch die Apps und deren Komponenten sind zugreifbar
  - z.B. Alarm Clock
    - SetAlarm (Activity)
      - https://android.googlesource.com/platform/packages/apps/AlarmClock/+/062d863a156a3564a839ab5a718f31500298635f/src/com/android/alarmclock/SetAlarm.java
    - AlarmReceiver (BroadcastReceiver)
    - AlarmProvider (ContentProvider)

Überblick / Zusammenfassung

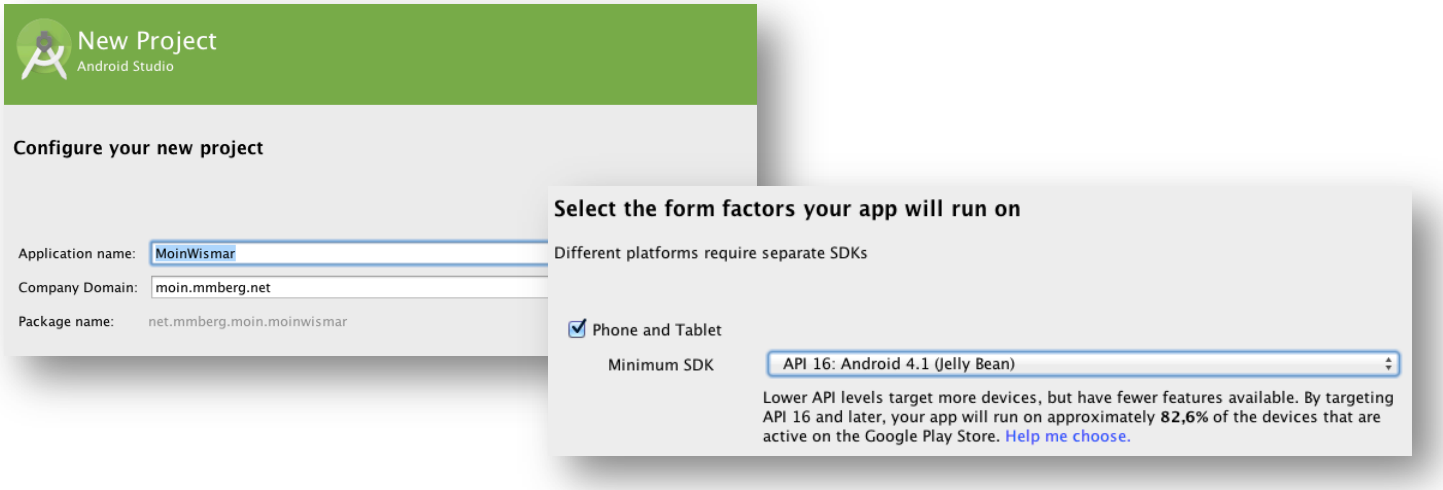
- Android-Projekt
  - Quelltext (Java-Klassen)
    - Activities
      - Activity1
      - Activity2

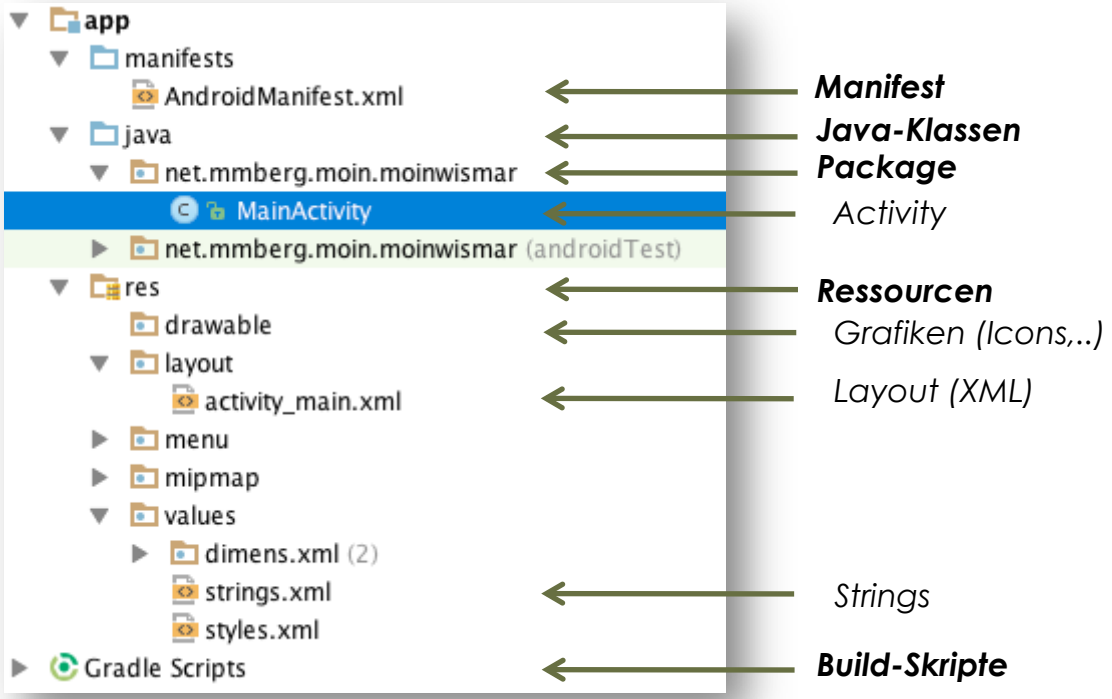
      Aufruf über Intents
    - (Services, BroadcastReceivers, ContentProviders)
  - Ressourcen
    - Grafiken
    - Strings
    - ...
  - Manifest
    - Liste der Komponenten
    - Berechtigungen
    - Minimale Anforderungen
  - Verwendete Bibliotheken
    - Android API
    - ...

Übung: Moin Wismar

- Neues Projekt
  - Projektname: MoinWismar
  - API 16
  - Blank Activity
  - Default-Werte übernehmen
  - Details: http://developer.android.com/training/basics/firstapp/creating-project.html



Projekt


app
manifests
AndroidManifest.xml
java
net.mmberg.moin.moinwismar
MainActivity
net.mmberg.moin.moinwismar (androidTest)
res
drawable
layout
activity_main.xml
menu
mipmap
values
dimens.xml (2)
strings.xml
styles.xml
Gradle Scripts
Manifest
Java-Klassen
Package
Activity
Ressourcen
Grafiken (Icons,..)
Layout (XML)
Strings
Build-Skripte

Layout

- Öffnen der `activity_main.xml`
  - Design-Ansicht (Tab ganz unten)
  - Wir sehen bereits eine Textview mit dem Inhalt „Hello world!“
- Wechseln in Textansicht

```
<RelativeLayout ...>

    <TextView android:text="@string/hello_world"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content" />

</RelativeLayout>
```

  - Relatives Layout, d.h. Positionierung in Abhängigkeit von anderen Elementen
  - TextView: stellt einen Text dar
    - Inhalt kommt aus Ressourcen-Datei → Verweise über @
    - Größe passt sich an den Inhalt an

String-Ressourcen

- `@string/hello_world` verweist auf Datei */res/values/strings.xml*

```
<resources>
    <string name="app_name">MoinWismar</string>

    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>
</resources>
```

MainActivity

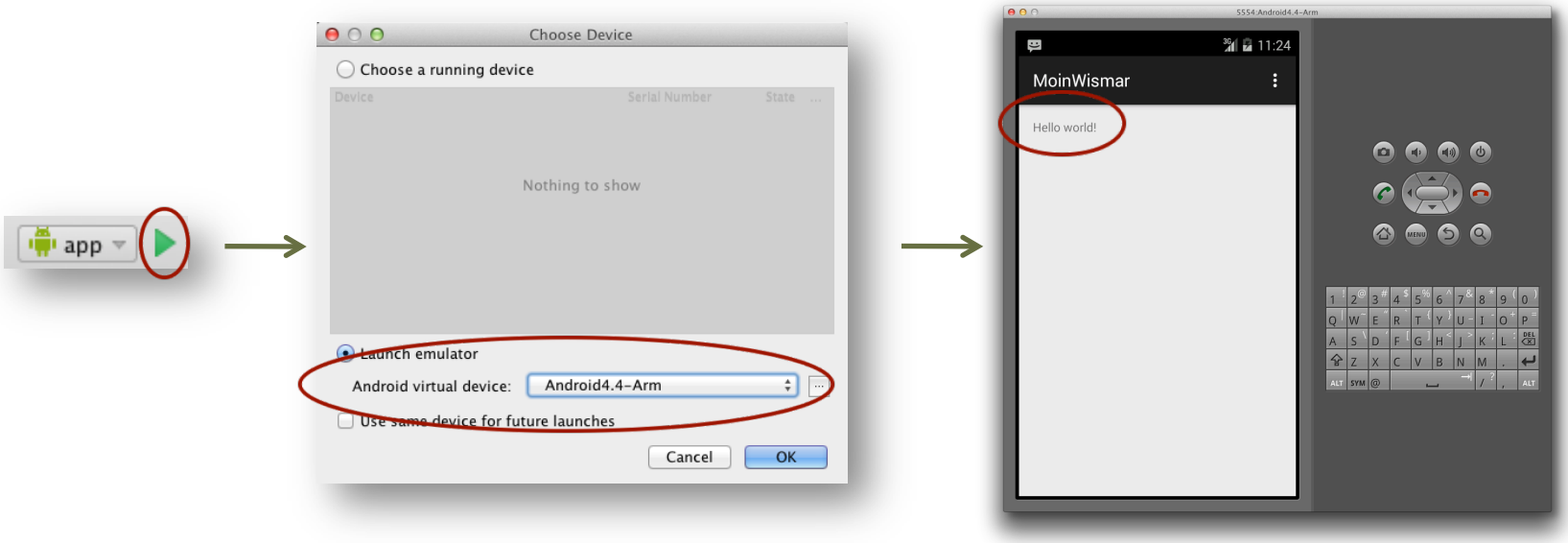
- Öffnen der MainActivity.java
- *onCreate* ist der Einstiegspunkt für die Activity (mehr Details später)
  - Hier wird das Layout geladen

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

Zugriff auf Ressourcen aus Java über R-Klasse

Starten der Anwendung

- Mit dem Emulator
  - Wenn noch nicht geschehen, AVD einrichten
  - (sollte bereits im letzten Praktikum erfolgt sein)
  - Auf „Run“ klicken und AVD auswählen, falls mehrere eingerichtet sein sollten



Starten der Anwendung

- Auf dem Gerät
  - Unter Windows müssen erst Treiber installiert werden
    - http://developer.android.com/tools/extras/oem-usb.html
  - USB-Debugging auf dem Smartphone aktivieren
  - Smartphone mit PC verbinden
  - In Android Studio auf „Run“ klicken und Gerät aus der Liste der verbundenen Geräte auswählen

Anpassen der Beispielanwendung

- In Anlehnung an http://developer.android.com/training/basics/firstapp/building-ui.html
- Wir starten von Null und entfernen das vorhandene Layout bzw. ersetzen das relative Layout durch ein Lineares
- Anpassung von `activity_main.xml` in */res/layout*
  - Höhe und Breite sind Pflichtangaben. Die Größe soll sich am Elternelement ausrichten
  - Layout soll horizontal sein, d.h. Elemente sollen waagerecht nebeneinander angeordnet werden

```
<LinearLayout
    android:layout_height="match_parent"
    android:layout_width="match_parent"
    android:orientation="horizontal"
    xmlns:android="http://schemas.android.com/apk/res/android">
</LinearLayout>
```

Textfeld hinzufügen

- Wir benötigen ein Eingabeelement: **EditText**
  - **ID**: eindeutiger Indentifizierer um aus Java-Quellcode und aus XML-Ressourcen-Dateien Zugriff auf das Element erhalten zu können
    - „@“ leitet eine Referenz aus XML heraus ein
    - „+“ wird benötigt, wenn eine Ressource das erste Mal definiert wird
  - **Breite**: statt absoluter Angabe so groß wie der Inhalt: `wrap_content`
  - **Höhe**: dito
  - **Hint** (vorausgefüllter Text wenn Nutzer noch nichts eingegeben hat)
    - Verweis auf String aus */res/values/strings.xml*

```
<LinearLayout
...>
    <EditText
     android:id="@+id/edit1"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:hint="@string/edit1_hint"/>
</LinearLayout>
```

strings.xml

```
<string name="edit1_hint">
        Name
</string>
```

Button hinzufügen

- Wir ergänzen im linearen Layout einen Button
  - Größe soll sich wieder an Inhalt anpassen
  - Anzuzeigender Text wird wieder über String-Ressource definiert
    - Aufschrift „Senden“
    - Attribut `android:text`

```
<Button
  android:id="@+id/button_send"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/button_send_text"/>
```

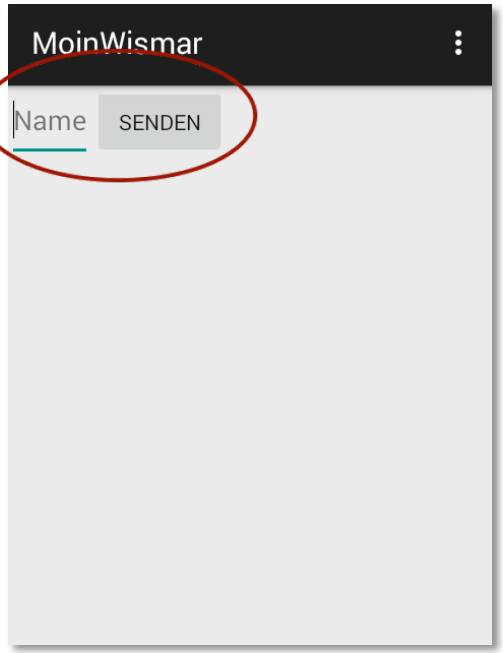
Optional, da wir später im Quellcode nicht darauf referenzieren

- An welcher Stelle werden die beiden Elemente vermutlich positioniert?

Layout testen

- App ausführen
- Platz nicht optimal genutzt
  - Eingabefeld sehr klein
  - Lösung: **Weights**
    - Relativer Anteil an Gesamtgröße in Bezug auf Elternelement
    - Ein Element mit der Weight 3 und eins mit der Weight 1 (Summe 4) führt zu einer Größe von ¾ bzw. ¼.

```
<EditText
    android:id="@+id/edit1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:hint="@string/edit1_hint"
    android:layout_weight="2"/>
```



Überprüfen in Vorschau

    - EditText (mit Weight von 2) doppelt so groß wie Button (mit einer Weight von 1)
      - ABER: Button größer als nötig, da er nun 1/3 fix einnimmt

Layout anpassen

- Standardweight ist 0
  - Element ist dann so groß wie es mindestens sein muss, um den Inhalt darzustellen
  - Wenn nun ein anderes Element einen Wert > 0 hat, nimmt es den kompletten noch freien Platz ein
- Weight von Button entfernen bzw. auf 0 setzen
- Weight von EditText auf 1 setzen

- Zur Performanceoptimierung wird empfohlen die Breitenangaben auf 0dp zu setzen (für Elemente, die Weights nutzen), da die Berechnung für *wrap_content* etc. überflüssig ist

Button zum Leben erwecken

- Ziel: Bei Buttonklick soll ein Text ausgegeben werden
- Auf Buttonklick reagieren über onClick-Event
  - Dieses wird in Layoutdatei definiert
  - Angabe der Methode, die beim Klick aufgerufen wird

```
<Button
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send_text"
    android:layout_weight="0"
    android:onClick="sendMessage"/>
```

- Diese Methode muss nun in der Activity definiert werden

OnClick-Methode

- Wird im Java-Code definiert
- Muss exakt so heißen wie im XML-File angegeben
- Darüber hinaus:
  - `public`
  - Rückgabetyp `void`
  - 1 Parameter vom Typ „`View`“
    - View-Klasse importieren (`import android.view.View;`)

```
public void sendMessage(View view){
   Toast toast = Toast.makeText(this, „Hallo“, Toast.LENGTH_SHORT);
   toast.show();
}
```

Toast: kurzes Einblenden eines Textes

- Diese Methode wird aufgerufen, wenn der Button geklickt wird
  - Der View-Parameter referenziert das angeklickte Objekt, d.h. den Button

Zweite Activity

- Bis jetzt haben wir nur eine Activity
- Beim Buttonklick soll nun statt des Textes eine zweite Activity angezeigt werden, die den Inhalt des Textfeldes der ersten Activity darstellt
- Zunächst erzeugen einer Activity, die beim Buttonklick aufgerufen werden soll
  - Rechte Maustaste auf Projekt und „Blank Activity“ hinzufügen
    - Name: **SecondActivity**
    - Als **Hierarchical Parent „MainActivity“** auswählen
    - Sonst Standardwerte übernehmen
      - LayoutName: activity_second
      - ...
  - Java-Klasse und Ressourcendateien werden angelegt und Manifest wird aktualisiert
    - Layout enthält wieder eine TextView, die „Hello world!“ anzeigt

Intent

- Nachricht zum Aufrufen der zweiten Activity (expliziter Intent) notwendig
- In der sendMessage-Methode, die beim Buttonklick aufgerufen wird, erzeugen wir nun ein *Intent* (statt des Toasts)
  - Zunächst Intent-Klasse importieren (`import android.content.Intent;`)

```
public void sendMessage(View view){
        Intent intent = new Intent(this, SecondActivity.class);
        startActivity(intent);
    }
```

  - Neuen Intent instanziieren
    - Erster Parameter: Context (Meist `this`, also die aktuelle Instanz selbst. Jede Activity ist vom Typ **Context**.)
    - Zweiter Parameter: Angabe welche Activity gestartet werden soll
  - `startActivity` aufrufen mit erzeugtem Intent als Parameter
    - startActivity ist eine Methode der Klasse Activity

Ausprobieren

- Anwendung starten
- Auf Button klicken
- Zweite Activity mit dem Text „Hello world!“ sollte angezeigt werden

Zugriff auf Daten aus Ressourcen

- Ziel: Wert von EditText übermitteln
  - Dafür zunächst Text auslesen und später dem Intent hinzufügen
- Zugriff auf Ressourcen aus Java
  - Über R-Klasse (statt per „@“ wie in XML-Ressourcen-Dateien)
    - EditText haben wir bereits in Layout-Datei definiert
      - Besitzt eine ID (`edit1`)
      - `findViewById` liefert eine View zurück → Cast erforderlich

```
EditText editText = (EditText) findViewById(R.id.edit1);
```

```
<LinearLayout
...>
    <EditText
      android:id="@+id/edit1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:hint="@string/edit1_hint"/>
</LinearLayout>
```

Intent mit Daten senden

- Dem Nachrichtenaufruf können auch Daten hinzugefügt werden
  - In unserem Beispiel: Der Text aus dem Eingabefeld
  - Realisiert über „Extras“, die dem Intent hinzugefügt werden können
    - Jedes Extra bekommt dabei einen eindeutigen Identifier

```
public final static String EXTRA_MESSAGE = "net.mmberg.EDITMESSAGE";
...

public void sendMessage(View view) {
 Intent intent = new Intent(this, SecondActivity.class);
 EditText editText = (EditText) findViewById(R.id.edit1);
 String message = editText.getText().toString();
 intent.putExtra(EXTRA_MESSAGE, message);
 startActivity(intent);
}
```

Layout von SecondActivity anpassen

- Dynamisch im Java-Quellcode statt über XML-Ressourcen-Datei
  - Wir benötigen das in der XML definierte Layout nicht
  - `setContentView(R.layout.activity_second)` aus onCreate() entfernen, sodass nicht mehr das Layout aus der Ressourcendatei verwendet wird
  - Stattdessen Layout dynamisch aufbauen, sodass wir den darzustellenden Text zur Laufzeit ermitteln können, statt vordefiniert in Ressourcen:

```
protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     TextView textView = new TextView(this);
     textView.setTextSize(40);
     String message = "Moin Wismar";
     textView.setText(message);
     setContentView(textView);
}
```

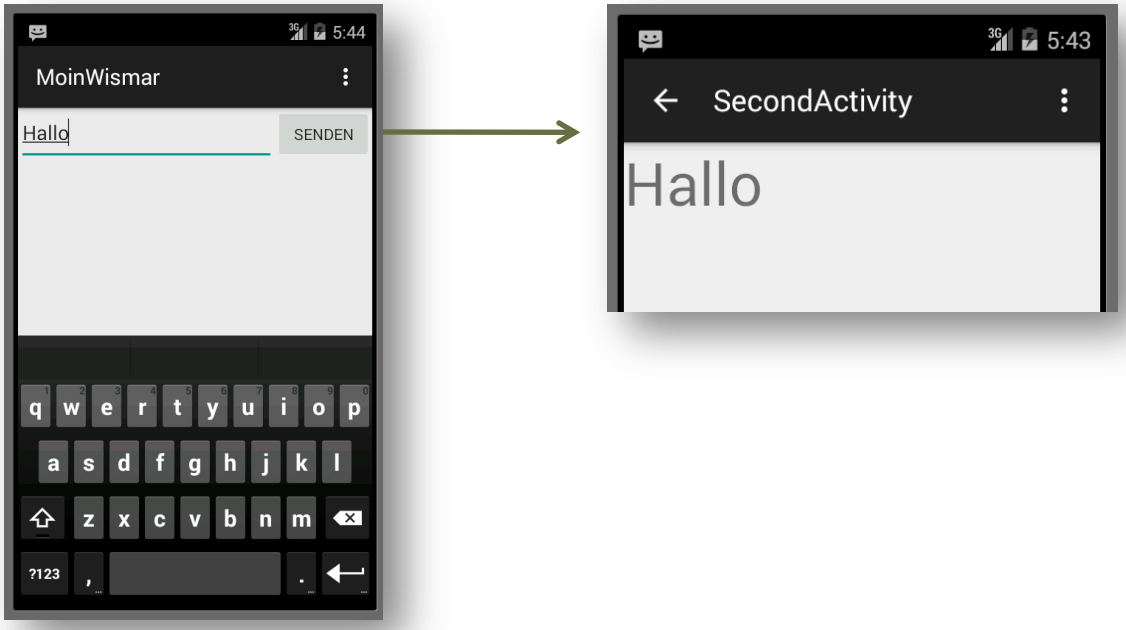
Hinzufügen der TextView zu ContentView. Alternativ vorher Linear Layout erzeugen (analog zu XML).

  - Statt festem String können wir auch die Daten aus dem Intent auslesen:

```
String message = getIntent().getStringExtra(MainActivity.EXTRA_MESSAGE);
```

Test

- Wir haben nun eine App, die aus zwei Activities besteht. In der ersten kann ein Text eingegeben werden, der dann über ein Intent an die zweite Activity übertragen und dort dargestellt wird.



Zusammenfassung

- Android-Quellcode wird in Java geschrieben. Allerdings wird nicht in Standard-Bytecode kompiliert, sondern in eine spezielle Variante, die auf ressourcenarme Geräte optimiert ist und von der Android Runtime ausgeführt wird.
- Jede Anwendung besitzt ein Manifest, das die grundlegenden Eigenschaften und die verwendeten Activities beschreibt.
- Activities sind die einzelnen Screens einer Anwendung. Sie stellen das User-Interface dar und erlauben die Interaktion des Nutzers mit der Anwendung. Eine Activity stellt Daten dar und reagiert auf Eingaben.
- Intents sind Nachrichten, die zum Aufrufen von Aktivitäten genutzt werden. Sie können auch Nutzdaten enthalten, die der aufgerufenen Activity mitgeteilt werden sollen.
- Layouts können per XML oder dynamisch im Java-Quellcode definiert werden.
- Strings werden in einer Ressourcendatei definiert, um möglichst einfach Beschreibungen austauschen oder die Anwendung in eine andere Sprache übersetzen zu können.

Nächste Woche...

- ... vertiefen wir die Entwicklung mit Android.
  - Lebenszyklus
  - Activity-Hierarchie
  - Speichern von Activity-Zuständen
- ... wird es eine neue Praktikumsaufgabe geben.