Programmierung mobiler Geräte

SoSe 2015

Native Entwicklung mit Android

Internetzugriff, Permissions, AsyncTasks

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

Rückblick

- Letzte Woche: Dynamische GUIs
  - Adapter
  - Fragments
- Heute:
  - Permissions
  - Internetzugriff
  - REST (+JSON)

Internetzugriff

- Ziel: Abrufen von Inhalten aus dem Internet
- Wir nutzen eine REST-API
  - vgl. JQuery / AJAX
- Kurze Wiederholung:
  - HTTP
  - REST
  - JSON

HTTP - Hyper Text Transfer Protocol

- Zustandsloses Übertragungsprotokoll
  - TCP
- Statuscodes
  - 1xx Information
    - 102 Processing (Anfrage noch in Bearbeitung)
  - 2xx Erfolgreich
    - 200 OK
    - 201 Created (Ressource angelegt)
  - 3xx Umleitung
    - 301 Moved Permanently
  - 4xx Clientfehler
    - 404 Not Found
  - 5xx Serverfehler
    - 500 Internal Server Error
- Methoden (bzw. HTTP-Verben)
  - **GET**: Anfordern einer Ressource
  - **POST:** Senden von Daten / Anlegen von neuen Ressourcen
  - **PUT:** Ressource updaten oder unter bekanntem Namen anlegen
  - **DELETE:** Löschen von Ressourcen

REST: Representational State Transfer

- Zur Maschine-Maschine-Kommunikation
- Leichtgewichtiger als SOAP (kein WSDL, Schema, basiert auf HTTP,…)
- **Adressierbarkeit**: URIs beschreiben Ressourcen
- **Zustandslosigkeit**: jeder Request beinhaltet alle nötigen Informationen
- **Repräsenatationsform**: JSON, XML,…
- **Operationen**: Nutzung der Standard-HTTP-Verben statt selbstdefinierter Methoden
  - GET, PUT, POST, DELETE
  - Ressourcen durch Substantive beschrieben
    - `/quotes/1`
      - GET und POST auf der Ressource „quotes“ statt getQuote und setQuote

JSON: Java Script Object Notation

- Textbasiertes Datenformat
- Key/value-basiert
- Oftmals als leichtgewichtigere Alternative zu XML genutzt
- JSON ist gültiges Javascript!
- Allerdings keine Schemas
  - Schwach typisiert (nur implizit)
- Datentypen
  - Objekte
  - Arrays
  - Zahl
  - String
  - Boolean

```
{"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
]}
```

http://www.w3schools.com/json/

Zugriff auf das Internet: Webview

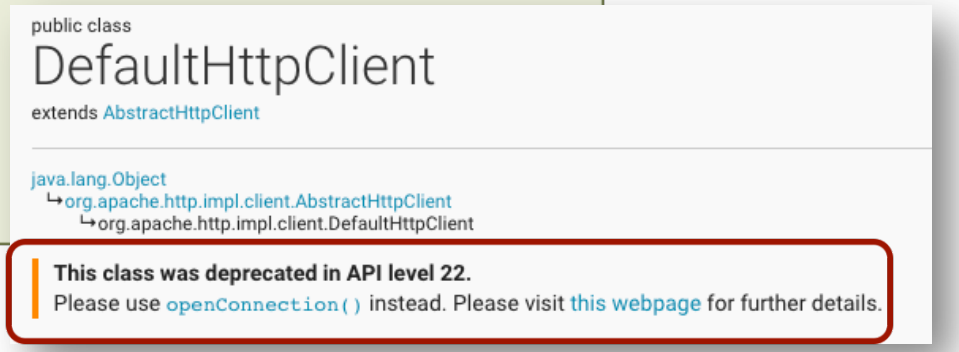
- WebView hinzufügen
- URL laden (Internet oder lokale Ressourcen)
  - Ggf. JS aktivieren
  - Lokale Ressourcen teilweise etwas knifflig:
    - Duplicate Resource Exception: Dateien ohne Punkte oder Unterstriche benennen
    - Ordner `/src/main/res/raw` benutzen
    - Keine Unterverzeichnisse unter `raw`

```
WebView webview = (WebView) findViewById(R.id.webView);
webview.getSettings().setJavaScriptEnabled(true);
webview.getSettings().setLoadWithOverviewMode(true);
webview.getSettings().setAllowUniversalAccessFromFileURLs(true);
webview.getSettings().setAllowFileAccessFromFileURLs(true);
webview.loadUrl("file:///android_res/raw/index.html");
```

Abrufen von Daten über HTTP

- Alte Variante über *DefaultHttpClient*

```
DefaultHttpClient client = new DefaultHttpClient(new BasicHttpParams());
HttpGet request = new HttpGet("http://mmberg.net");

InputStream input=null;
String result=null;

try{
  HttpResponse response = client.execute(request);
  HttpEntity entity = response.getEntity();
  input = entity.getContent();

   //...
}
catch(Exception ex){
...
}
```



Abrufen von Daten über HTTP

- Neue Variante:

```
URL url = new URL("http://mmberg.net");
URLConnection conn = url.openConnection();
input = conn.getInputStream();
```

- Verarbeiten des InputStreams:

```
BufferedReader reader = new BufferedReader(new InputStreamReader(input));
StringBuilder sb=new StringBuilder();
String line = null;

while ((line = reader.readLine()) != null)
{
   sb.append(line + "\n");
}
String result = sb.toString();
```

Ergebnis (HTML):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
<meta name="keywords" content="Markus Berg, dialogue
modelling, dialogue systems, natural language“>
```

Abrufen von JSON-Inhalten

- Anpassen der URL
  - URL: http://mmberg.net:8080/QuoteRESTServer/quotes
  - sonst keine Änderung im Request
  - Anpassung der Auswertung (nächste Folie)

```
URL url = new URL("http://mmberg.net:8080/QuoteRESTServer/quotes");
URLConnection conn = url.openConnection();
input = conn.getInputStream();
```

Ergebnis: JSON-String

```
[{"author":"Gottlob Frege","quote":"Urteilen kann als Fortschreiten von einem Gedanken zu seinem
Wahrheitswerte gefasst werden"},{"author":"Nick Black","quote":"...if you aren't, at any given time,
scandalized by code you wrote five or even three years ago, you're not learning anywhere near
enough"},...]
```

- Verarbeiten des Strings
  - Android besitzt Methoden zum Verabeiten von JSON-Objekten
    - `import org.json.JSONArray` bzw. `JSONObject`
    - Wir haben es im Beispiel mit einem JSONArray zu tun, das aus mehreren JSONObjects besteht
      - author
      - quote

```
String result; //filled with String from HTTP-Request
//...
JSONArray json = new JSONArray(result);
for(int i=0; i<json.length();i++){
      list.add(json.getJSONObject(i).getString("quote"));
}
```

Permissions

- Wir erhalten eine Exception beim Zugriff auf‘s Internet

java.lang.SecurityException: Permission denied (missing INTERNET permission?)

- Hinzufügen der Berechtigung im Manifest

```
<uses-permission
android:name="android.permission.INTERNET" />
```

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.mmberg.rest.rest" >

    <uses-permission android:name="android.permission.INTERNET" />

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

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".DetailsActivity"
            android:label="@string/title_activity_details"
            android:parentActivityName=".MainActivity" >
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value="net.mmberg.rest.rest.MainActivity" />
        </activity>
    </application>

</manifest>
```

Weitere Permissions

- `android.permission`
  - `CAMERA`
  - `FLASHLIGHT`
  - `READ_CALENDAR`
  - `READ_CONTACTS`
  - `READ_SMS`
  - `VIBRATE`
- Berechtigungen werden vom Nutzer beim Installieren der App erteilt
- Im Manifest werden die Berechtigungen angegeben, die bei der Installation erfragt werden und die vom Nutzer akzeptiert werden MÜSSEN (sonst wird App nicht installiert)
- Teilweise werden die Permissions zur Filterung im PlayStore genutzt
  - Eine App mit Camera-Permission wird nicht auf einem Smartphone ohne Kamera angezeigt

Threads

- Wir führen die Anwendung erneut aus und erhalten nun folgende Exception:

  android.os.NetworkOnMainThreadException

- Hintergrund: (Potentiell) Langdauernde Operationen (wie Netzwerkzugriff) sollten nicht im Hauptthread stattfinden, da diese die GUI blockieren
  - Ausführung einer Anweisung im MainTask darf nicht länger als 5 Sekunden dauern
  - Einige Aktionen sollten (und dürfen) nicht im MainTask erfolgen (s. Bsp.)
  - → *Responsive GUI*

- Prozesse können aus mehreren Threads bestehen
  - Lassen sich (quasi)parallel ausführen
    - mehrere Prozessoren: parallel, ein Prozessor: quasiparallel
  - D.h. mehrere Aufgaben „*gleichzeitig*“ ausführen
  - Jede Anwendung hat min. einen Thread (den Hauptthread)

AsyncTask vs Thread

- **Threads** (Java-Standard)
  - sind schwergewichtiger und etwas aufwändiger zu implementieren
    - Manuelle Synchronisation mit Haupttask (und Rückgabe der Ergebnisse) notwendig
    - Abbrechen des Threads
- **AsyncTasks** (Android Hilfsmittel)
  - Leichter zu implementieren
  - Zugriff auf GUI-Elemente möglich
  - **Sind für GUI-bezogene Operationen bestimmt und garantieren, dass die GUI trotzdem responsive bleibt**
  - Mögliche Probleme:
    - sind nur lose mit dem Lebenszyklus der Activity verbunden
    - Configuration Change bedeutet Zerstörung der Activity aber AsyncTask wird nicht abgebrochen
      - AsyncTask läuft bis er fertig ist und versucht (evtl.) GUI der alten (zerstörten) Activity zu aktualisieren → Exception
    - Memory Leaks möglich wenn als innere Klasse der Activity realisiert, da Task eine Referenz auf die Activity hält und diese somit nicht „garbage-collected“ werden kann
  - **Schwere (bzw. nicht GUI-relevante Aktivität) in herkömmliche Threads auslagern!**
    - Große Downloads
    - CPU-intensive Berechnungen

AsyncTask

AsyncTask enables proper and easy use of the UI thread. This class allows to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers.

http://developer.android.com/reference/android/os/AsyncTask.html

- AsyncTask kein Ersatz für Threads
  - Für Operationen, die an die Oberfläche zurückgegeben werden sollen
  - Richtwert: Maximal einige Sekunden Laufzeit
- Asynchronität
  - Nach Abschluss wird Methode `onPostExecute` aufgerufen
  - Kein Rückgabewert, da Aufrufer nicht warten soll (würde blockieren)
  - Ggf. innere Klasse anlegen und `onPostExecute`-Methode überschreiben

Async Task: Methoden

- **onPreExecute**() (optional)
  - Wird im UI-Thread aufgerufen bevor der Task ausgeführt wird
  - Zu Setupzwecken
- **doInbackground**(Params..)
  - Wird im Background-Thread aufgerufen direkt nachdem `onPreExecute` fertig ist
  - Muss auf jeden Fall überschrieben werden
  - Die eigentliche (langdauernde) Operation
- **onProgressUpdate**(Progress..) (optional)
  - Wird im UI-Thread aufgerufen nachdem `publishProgress(Progress...)` aufgerufen wurde
  - Oberfläche über aktuellen Zustand informieren (z.B. Statusbar), während Background-Thread weiter arbeitet
- **onPostExecute**(Result) (optional)
  - Wird im UI-Thread aufgerufen, wenn Task fertig ist
  - Enthält das Ergebnis von `doInBackground()` im Parameter `Result`

Übung: AsyncTask: Anzeigen eines Zitats

- Unsere MainActivity besitzt eine TextView
- Abrufen der Zitate per REST-Service
  - Nutzung eines AsyncTasks, da Abrufen länger dauern kann bzw. Internetzugriffe generell außerhalb des UI-Threads durchgeführt werden müssen
- Zunächst Anzeigen des ersten Zitates in Oberfläche

AsyncTask implementieren

- Klasse anlegen, die das Laden der Quotes per REST managed
  - Erbt von AsyncTask
    - Implementieren von `doInBackground()`

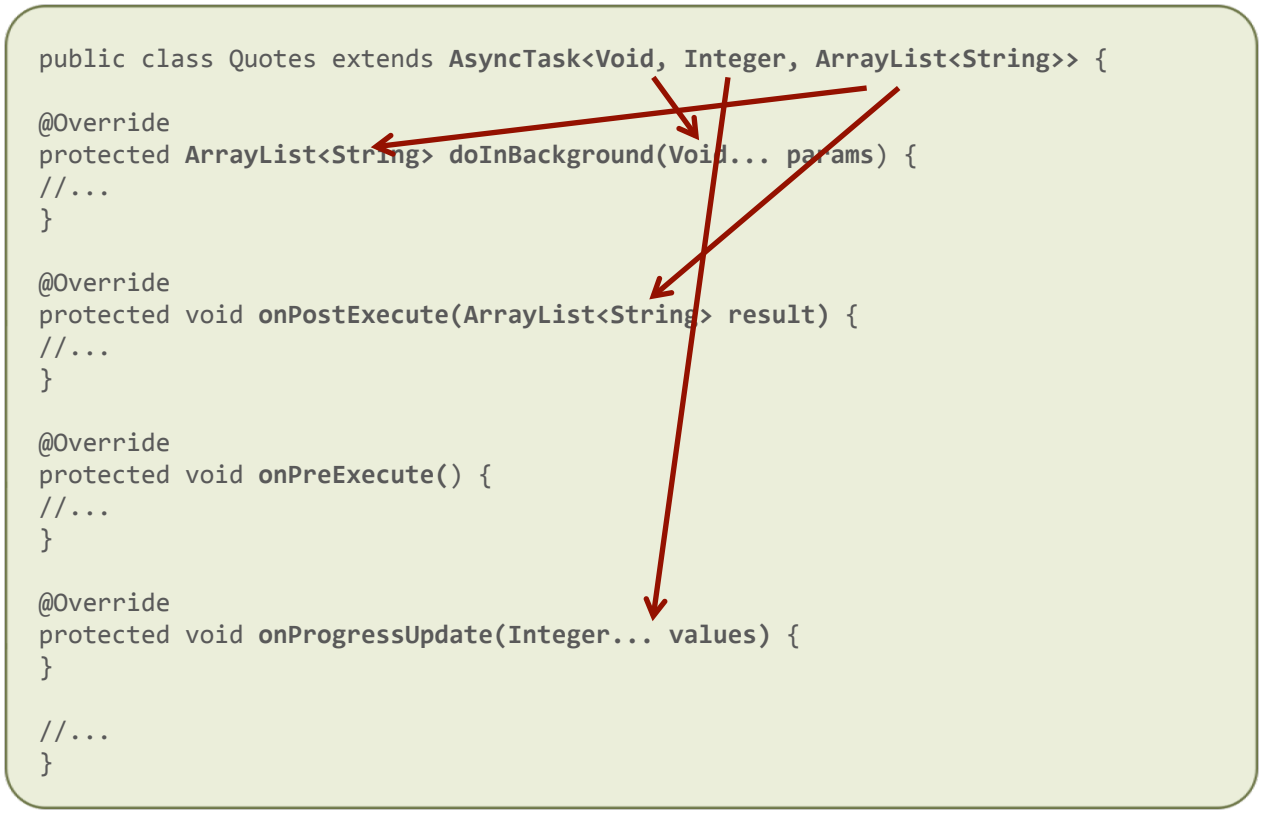
- Liste der Zitate als ArrayList zurückgeben
- Zugriff auf's Internet per URLConnection
- JSON in ArrayList umwandeln

```
public class Quotes extends AsyncTask<Void, Void, ArrayList<String>> {

@Override
protected ArrayList<String> doInBackground(Void... params) {
   return getQuotes();
}

private ArrayList<String> getQuotes(){
    ArrayList<String> list=new ArrayList<>();

    Log.i(TAG,"Quotes laden...");
    InputStream input=null;
    String result=null;

    try{
       URL url = new URL("http://mmberg.net:8080/QuoteRESTServer/quotes");
       URLConnection conn = url.openConnection();
        //...
    }
    //process JSON...
}
```

AsyncTask: doInBackground()

- `Void` (Achtung: groß geschrieben) für Generics vom „Typ“ `void`, d.h. solche, an denen wir nicht interessiert sind
  - Hier: `doInBackground()` braucht keine Parameter in unserem Bsp.

```
public class Quotes extends AsyncTask<Void, Integer, ArrayList<String>> {

@Override
protected ArrayList<String> doInBackground(Void... params) {
//...
}

@Override
protected void onPostExecute(ArrayList<String> result) {
//...
}

@Override
protected void onPreExecute() {
//...
}

@Override
protected void onProgressUpdate(Integer... values) {
}

//...
}
```



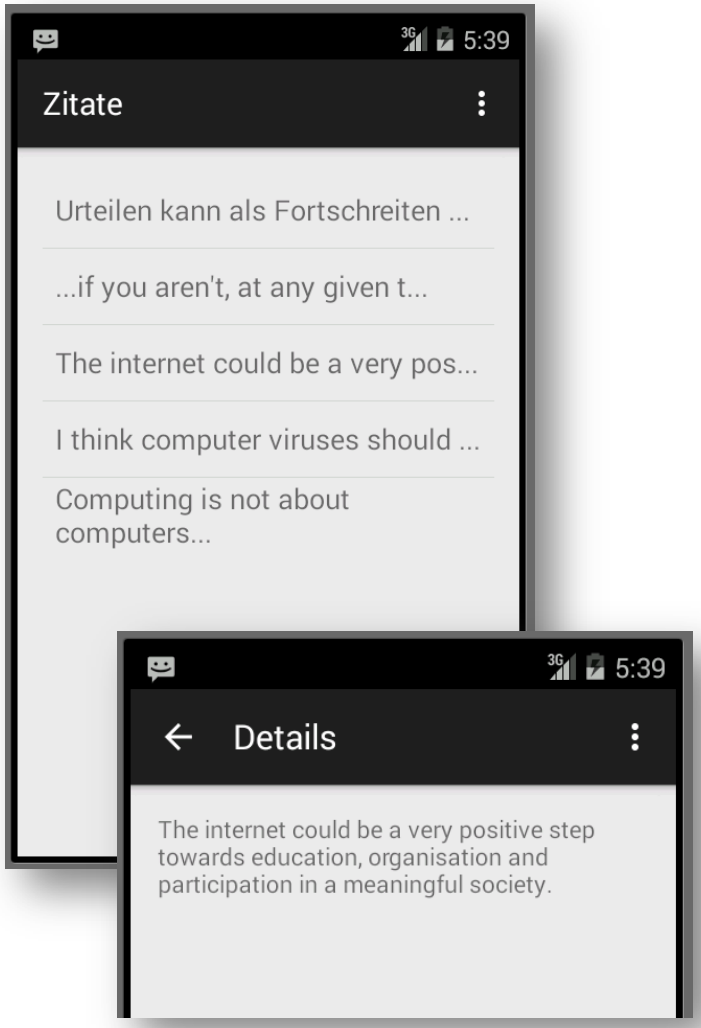
MainActivity: Aufrufen von AsyncTask

- Aufruf per `execute()`
- Niemals `doInBackground()` oder eine der anderen Methoden manuell aufrufen
- Hier: `onPostExecute()` lokal (in Activity) überschrieben
  - Stattdessen auch Implementierung in AsyncTask-Klasse möglich

```
new Quotes(){
            @Override
            public void onPostExecute(ArrayList<String> list){
                String quote;
                if (list.size()>0) {
                    quote = list.get(0);
                }
                else quote="leer";*/

                textview.setText(quote);
            }
        }.execute();
```

Übung: ListView und zweite Activity

- Wir ersetzen die TextView durch eine ListView und binden die Liste aller Zitate an diese
- Die ListView soll nur die ersten 32 Zeichen als Vorschautext enthalten
- Wir fügen eine zweite Activity hinzu, die eine TextView zum Anzeigen des vollständigen Zitates enthält
- Wir implementieren das onItemClick-Event und starten bei dessen Auslösung die zweite Activity. Diese soll nun das gesamte Zitat anzeigen.

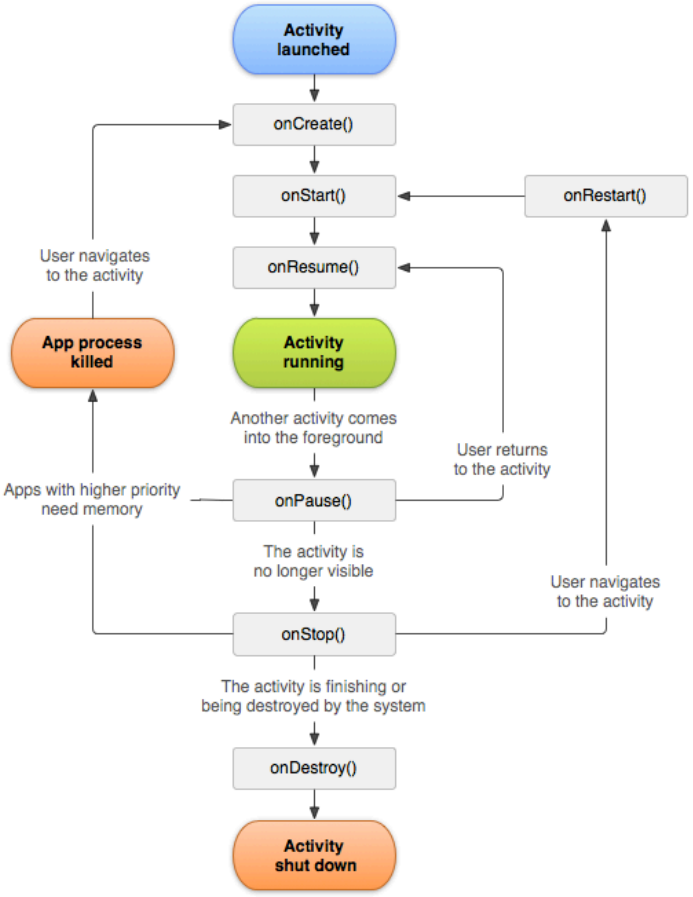


ListView & onClick

```
new Quotes(){
            @Override
            public void onPostExecute(ArrayList<String> list){
                quotesList=list;

                ListView listview = (ListView) findViewById(R.id.listView);
                listview.setOnItemClickListener(new AdapterView.OnItemClickListener() {
                    @Override
                    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
                        Intent detailsIntent = new Intent(MainActivity.this, DetailsActivity.class);
                        detailsIntent.putExtra(QUOTE, quotesList.get(position));
                        startActivity(detailsIntent);
                    }
                });

                ArrayList<String> shortlist = new ArrayList<>();
                for(String quote : list){
                    shortlist.add(quote.substring(0,32)+"...");
                }
                ArrayAdapter<String> adapter = new ArrayAdapter<>(MainActivity.this,
                                android.R.layout.simple_list_item_1,shortlist);
                listview.setAdapter(adapter);


            }
 }.execute();
```

Quotes nur einmalig laden

- Bei Rotation werden Quotes neu geladen
  - Ersichtlich über Logmessage bei Methodenaufruf
  - `Log.i(TAG,"Quotes laden...");`
- Wir erinnern uns an den Lebenszyklus
- → Zwischenspeichern
  - Bundle



Praktikum

- TV App
  - TV-Daten per REST abrufen
    - http://mmberg.net:8080/TVProgramm/now
    - Permissions nicht vergessen!
  - GUI (ListView mit Adapter)
    - *Siehe Übung letzte Woche*
    - Anzeigen der Titel
  - *Zweite Activity (per Klick)*
    - *Anzeigen der Details (Sender, Uhrzeit)*
      - *Übergeben per Intent (Extras)*
    - Teilen-Funktion (per Button) für die ausgewählte Sendung
      - *Siehe letztes Praktikum (implizite Intents)*
  - Bei Rotation Daten nicht erneut abrufen (Zwischenspeichern: Bundle)
  - Abgabefrist: 2. Juni 2015

Was haben wir gelernt?

- Anwendung von:
  - Adapter & Listview
  - Bundles
  - Permissions
  - AsyncTasks
  - Internetzugriff
  - JSON-Verarbeitung
  - Activities & Intents