Skript - Native App-Entwicklung: Android

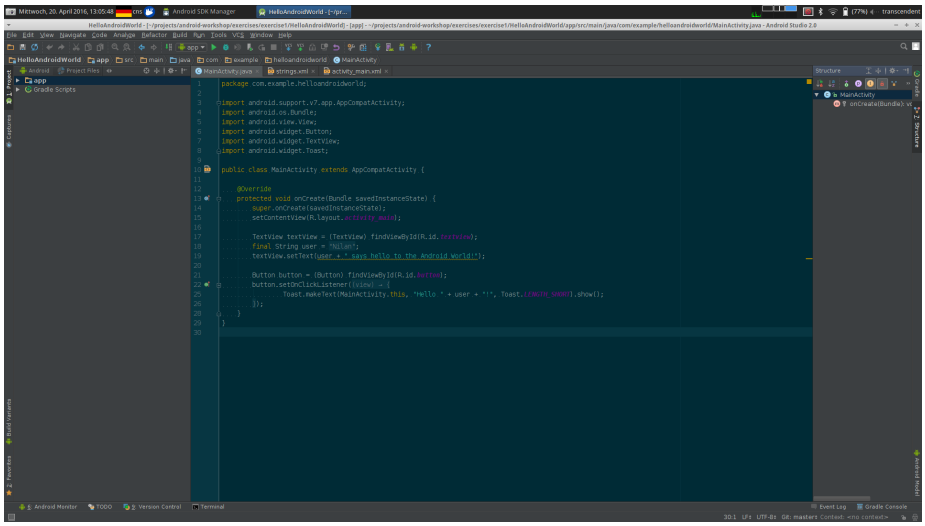
Nilan Marktanner, nilan.marktanner@gmail.com

27. April 2016

Inhaltsverzeichnis

1 Entwicklungsumgebung und Projektstruktur

1.1 Entwicklungsumgebung



Als Entwicklungsumgebung kommt Android Studio 2.0 zum Einsatz. Apps können im Emulator oder auf dem Smartphone ausgeführt werden.

1.2 Projektstruktur

Android Projekte sind in Projekt- und in Moduldateien gegliedert.

Von den Projektdateien muss im Kontext dieses Kurses höchstens die projektweite `build.gradle`-Datei angepasst werden. Gradle wird unter Android Studio zum Bauen und zur Abhängigkeitsverwaltung verwendet.

Die Moduldateien setzen sich u. A. so zusammen:

- eine modulweite `build.gradle`-Datei
- ein Manifest-Datei, die bestimmte Meta-Informationen zur App beinhaltet unter `AndroidManifest.xml`.
- Bilder, sonstige Daten, die in die `.apk` Datei kompiliert werden unter `main/assets/`.
- Sonstige Resource-Dateien wie Activity-Layouts, Strings, sonstige Werte unter `main/res/`.
- Tests unter `androidTest/`.
- der eigentlichen Sourcecode der Androidkomponenten und der Javaklassen, die in der App verwendet werden unter `src/main/java/`

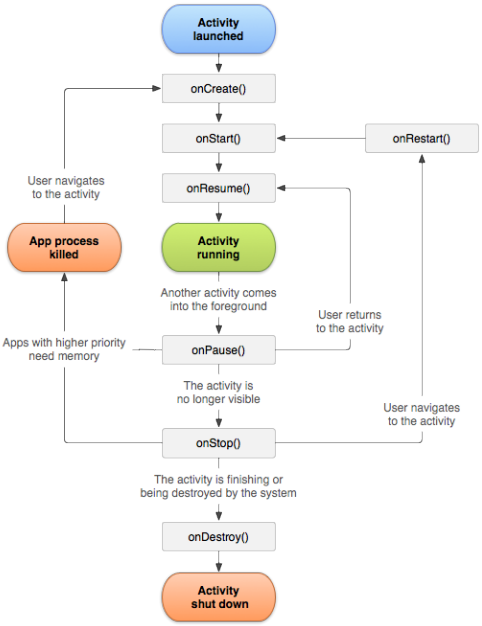
1.3 Gradle

Gradle wird unter Android Studio zum Bauen und zur Abhängigkeitsverwaltung verwendet.

2 Wichtige Komponenten einer Android App

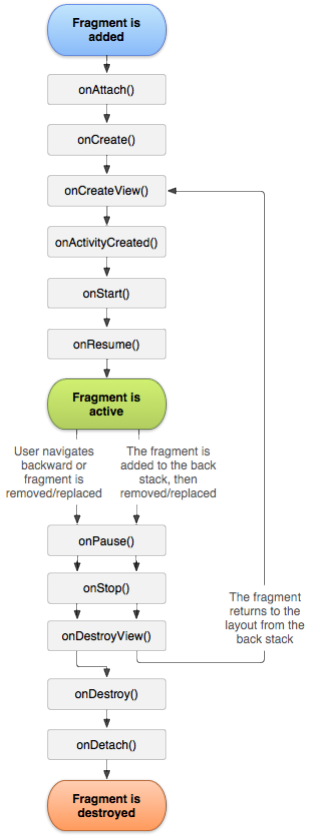
2.1 Activity

- Eine Activity stellt ein Fenster bereit, das als User Interface dient.
- Eine App besteht aus mehreren Activities, wobei es eine Main Activity gibt - das ist der Einstiegspunkt beim Starten der App.
- Eine Activity ist meist einer bestimmten Aufgabe zugeordnet - *Emails abrufen* , *Email verfassen*, *Neuen Account erstellen* sind Aufgaben, die in einer Email App je einer Activity zugeordnet werden könnten.
- Jede Activity einen eigenen *Lifecycle*, sie durchläuft also verschiedene Zustände. Dies kann nutzerbedingt geschehen (z.B. Nutzer schließt App), oder durch das Android System ausgelöst werden (Android zerstört App wenn der Arbeitsspeicher voll ist).
- Der Nutzer kann innerhalb einer App zwischen Activites navigieren oder Activites direkt von außerhalb der App aufrufen - beispielsweise wenn man ein Bild per Email teilt.
- Dabei werden die Activites in einem Back-Stack verwaltet. Beim Verlassen einer Activity mit der Zurück-Funktion wird die Activity vom Stack entfernt und es erscheint die vorherige Activity.



2.2 Fragment

- Ein Fragment kann man als Sub-Activity ansehen, wobei jedes Fragment ein Teil des UI darstellt.
- Ermöglicht flexible Anordnung des UI zum Beispiel für unterschiedliche Screengrößen.
- Jedes Fragment hat einen separaten *Lifecycle*.



2.3 Service

- Ermöglicht es, Operationen im Hintergrund laufen zu lassen, selbst wenn die Activity nicht im Vordergrund ist.
- **Achtung:** läuft standardmäßig im Main-Thread der Activity, kann diese also blockieren. Größere Operationen sollten in einen eigenen Thread ausgelagert werden.

2.4 BroadcastReceiver

- Kann auf beliebige Events reagieren und andere Events, Services, Activities anstoßen.
- Der Subscriber beim Publish/Subscribe-Pattern.
- Beispiele für Events: Eingehender Anruf, Batterystatus niedrig, Download abgeschlossen.

2.5 Intent

- Dienen als Objekt, um Information zwischen Komponenten auszutauschen.
- Drei hauptsächliche Anwendungen:
  - Starten einer Activity.
  - Starten eines Services.
  - Starten eines BroadcastReceivers.
- Mehr Informationen zu Intents befinden sich hier.

2.6 ContentProvider

- Ermöglicht es, Daten *zwischen verschiedenen Applications* auszutauschen.
- Beispiele: Contacts Provider, Calendar Provider.

3 Eine Activity erstellen

Die Erstellung einer neuen Activity umfasst mehrere Schritte:

- **Activity in Manifest eintragen:** Damit die Activity aufgerufen werden kann, muss sie dem System bekannt gemacht werden, indem sie im Manifest hinzugefügt wird. Hier sind zusätzliche Konfigurationen möglich.
- **Activity-Layout erstellen:** das Layout der Activity wird meist durch eine Layoutdatei unter `src/main/res/layout` festgelegt.
- **Activity-Layout setzen:** in der `onCreate()`-Methode der Activity wird das UI entweder direkt mit Hilfe einer vorher erstellen Layoutdatei gesetzt, oder dynamisch aufgebaut.

Diese Schritte können bereits in einem neuen Projekt nachvollzogen werden, falls im Projekt-Wizard eine Activity erstellt wurde.

3.1 Activity im Manifest eintragen

Um eine Activity dem System bekannt zu machen, muss man sie unter dem `application`-Tag in `AndroidManifest.xml` hinzufügen.

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

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

</manifest>
```

Es gibt verschiedene Intent-Filter, die hier verwendeten eignen sich für die MainActivity. Mehr Informationen zu Intents befinden sich hier. So kann man

eine Activity, die sich zum Abspielen von Audio-Dateien eignet zum Beispiel dementsprechend markieren. Beim öffnen einer Audio-Datei kann der Nutzer dann zwischen allen Apps wählen, die so markiert worden sind.

3.2 Activity-Layout erstellen

Das Layout einer Activity wird in ihrer `onCreate`-Methode festgelegt. Es gibt zwei Arten, wie das ablaufen kann. Entweder wird das Layout aus einer vorher angelegten Layoutdatei gelesen, oder das Layout wird dynamisch aufgebaut.

3.2.1 Aufbau und Inhalt einer Layoutdatei

Android bietet bereits viele Komponenten, die man für seine UI verwenden kann. Diese Komponenten werden `View` genannt. Ein View kann alles Mögliche sein, vom einfachen Text-Label, über einen Button bis hin zum OpenGL-SurfaceView. Eine `ViewGroup` ist selbst ein View und dient dazu, Views auf dem Display anzuordnen. Zu den geläufigsten ViewGroups gehören `LinearLayout`, `RelativeLayout` und `FrameLayout`. Bei einem LinearLayout werden Views der Reihe nach entweder horizontal oder vertikal angeordnet. Bei einem RelativeLayout hat man mehr Kontrolle und kann einen View relativ zu einem anderen, zum Beispiel darunter oder links davon, positionieren. Ein FrameLayout ist ein spezielles RelativeLayout, mit dem man Views auch direkt *übereinander* zeichnen kann.

Ein View benötigt zumindest die Angabe zur Breite und Höhe. Dabei kann man entweder `match_parent` oder `wrap_content` angeben, oder eine festen Wert angeben. Um einen View in der Activity zu identifizieren benötigt er eine (layoutweite) eindeutige ID.

Am folgenden Beispiel sieht man eine mögliche Layoutdatei `activity_main.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<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:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.helloandroidworld.MainActivity">

    <TextView
        android:id="@+id/textview"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>

    <Button
        android:id="@+id/button"
        android:text="OK"
        android:layout_below="@id/textview"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>

</RelativeLayout>
```

Diese kann man dann in der `onCreate`-Methode der Activity auslesen um das Layout zu setzen:

```
// Auszug aus MainActivity.java
@Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // hier wird das Layout gesetzt
        setContentView(R.layout.activity_main);

        // es folgen weitere Änderungen an den Komponenten
        TextView textView = (TextView) findViewById(R.id.textview);
        final String user = getResources().getString(R.string.user_name);
        textView.setText(user + " says hello to the Android World!");

        Button button = (Button) findViewById(R.id.button);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Toast.makeText(MainActivity.this,
                               "Hello " + user + "!",
                               Toast.LENGTH_SHORT).show();
            }
        });
    }
```

3.2.2 Eine Activity programmatisch aufbauen

Man kann ein Layout auch komplett programmatisch in der `onCreate`-Methode aufbauen, das heißt ohne, dass eine Layoutdatei ausgelesen wird. Meist reicht es aber aus, eine Layoutdatei auszulesen und gegebenenfalls benötigte Änderungen hinzuzufügen.

4 Eine Activity aus einer anderen heraus starten

Um eine Acticity zu starten benötigt man einen Intent. Dabei gibt es auch die Möglichkeit, aus der aufgerufenen Activity ein Ergebnis an die aufrufende Activity zu senden.

4.1 startActivity

Um eine Activity zu starten ist zunächst ein Intent nötig. Ein Intent kann zusätzliche Information enthalten, die mit Hilfe der Methode `putExtra` hinzugefügt werden können. Extras werden als key-value Paare verwaltet, weshalb jedes Extra durch einen eindeutigen String identifiziert wird.

```
// in MainActivity
Intent intent = new Intent(this, AnotherActivity.class);

// add some extra information
String info = "some information";
intent.putExtra("EXTRA_INFORMATION", info);

// start activity with the intent
startActivity(intent)
```

Dies startet eine neue Activity, die über der alten erscheint. Der Intent, mit dem die Activity gestartet wurde, kann mit dem Methodenaufruf `getIntent` erhalten werden.

```
// In AnotherActivity
Intent intent = getIntent();
String information = intent.getStringExtra("EXTRA_INFORMATION");
```

Wird der Zurück-Knopf betätigt oder die Activity programmatisch beendet kehrt der Nutzer zurück zur ursprünglichen Activity.

4.2 startActivityForResult

Um ein Ergebnis von einer aufgerufenen Activity zu erhalten, kann die `startActivityForResult`-Methode verwendet werden. Dieser muss ein Request-Code übergeben werden, der in der `onActivityResult`-Methode verwendet werden kann, um das erhaltene Ergebnis zuzuordnen.

```
// in MainActivity
Intent intent = new Intent(this, AnotherActivity.class);

// add some extra information
String info = "some information";
intent.putExtra("EXTRA_INFORMATION", info);

// start activity for result
int requestCode = 1;
startActivityForResult(intent, requestCode)
```

In der aufgerufenen Acticity kann dann ein Ergebnis mit der `setResult`-Methode gesetzt werden. Wird die Activity dann mit der `finish`-Methode beendet, wird automatisch die `onActivityResult`-Methode der aufrufenden Acticity ausgelöst, in der auf das Ergebnis reagiert werden kann.

```
// in AnotherActivity
Intent intent = new Intent();

/* put extras into intent... */

// set result and finish this Acticity
setResult(RESULT_OK, intent);
finish();
```

5 Permissions

Im Android System benötigen Apps die explizite Berechtigung des Nutzers, um Zugriff auf gewisse Funktionen zu erhalten. Eine benötigte Berechtigung muss im AndroidManifest eingetragen werden.

Möchte eine App beispielsweise über einkommende SMS benachrichtigt werden, muss `<uses-permission android:name=ändroid.permission.RECEIVE_SMS"/>` hinzugefügt werden.

In diesem Kurs wird nicht auf das Berechtigungsmodell von Android 6 eingegangen, das sich wesentlich von dem bisherigen Modell unterscheidet.

Mehr Informationen zu Permissions findet man hier.

6 User-Feedback

Oft möchte man den Nutzer über ein bestimmtes Ereignis informieren oder möchte sichergehen, dass eine Aktion wirklich umgesetzt werden soll (“soll diese

Email wirklich gelöscht werden?"). Dafür gibt es verschiedene Möglichkeiten: ein Toast dient als kurze Mitteilung an den Nutzer, die nach kurzer Zeit wieder verschwindet; eine Snackbar ermöglicht eine zusätzliche Aktion, zum Beispiel, das Löschen der Email rückgängig zu machen; ein Dialog erlaubt meist zwei Auswahloptionen (OK/Cancel oder Yes/No). Dialoge unterscheiden sich hierbei von Toasts und Snackbars, denn sie benötigen zwingend eine Reaktion des Nutzers. Toasts und Snackbars sind subtiler und stören den Programmablauf kaum.

Eine schöne Übersicht findet sich beispielsweise hier.