Hochschule für
Technik und Wirtschaft
Dresden
University of Applied Sciences

# Android-Zusammenfassung

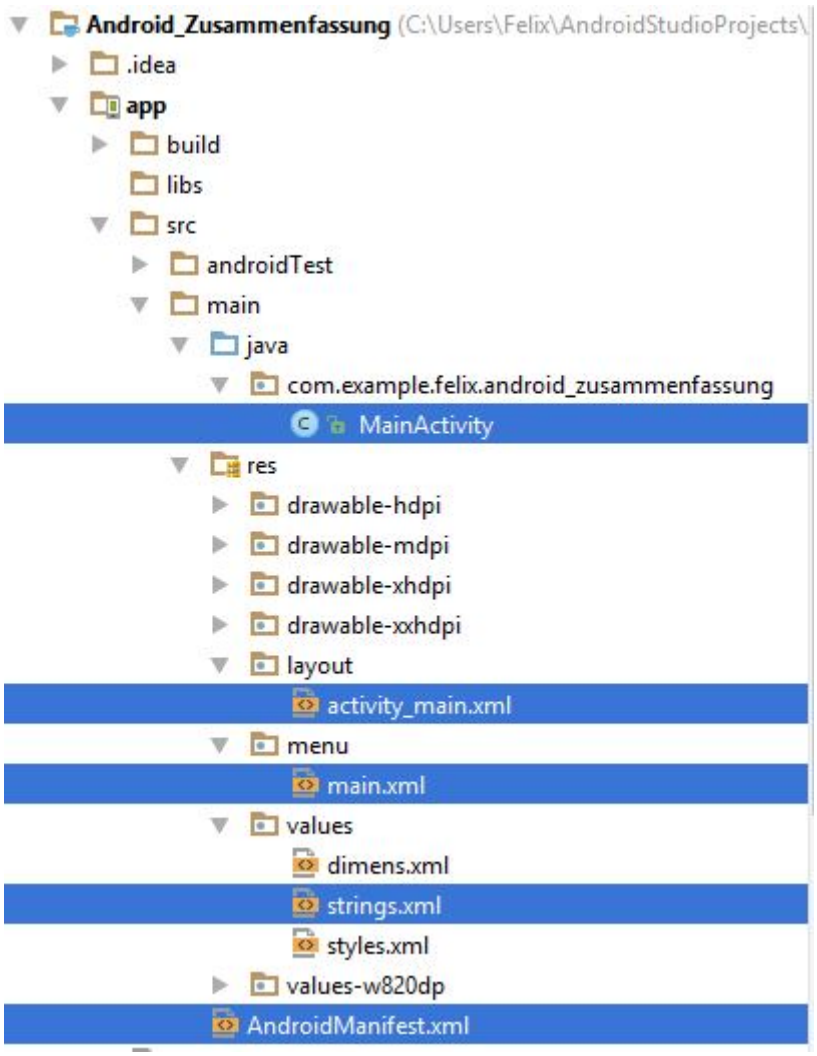
Felix Krautschuk
(Matrikelnummer: 34230)

Studiengang Informatik

5. Semester

# Contents

# 1 Programmstruktur - die wichtigsten Verzeichnisse und Dateien



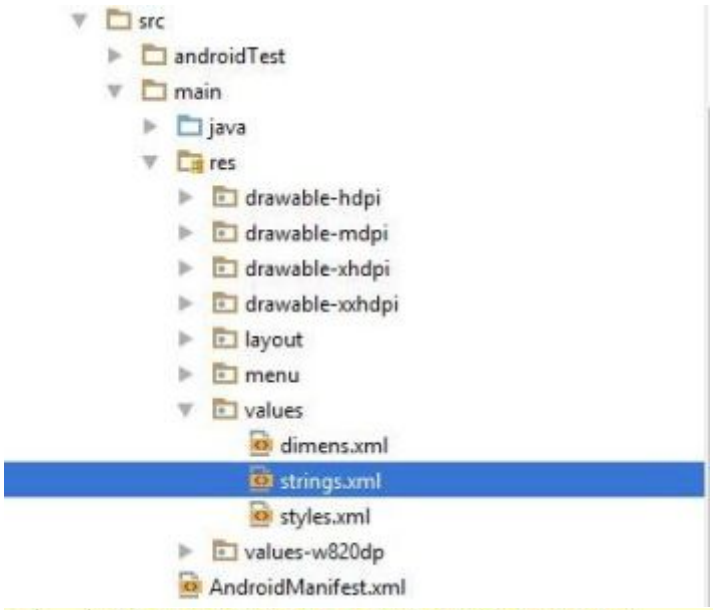

## 1.1 AndroidManifest.xml

Zu jeder App gehört eine zentrale Beschreibungsdatei. Sie enthält eine Liste der Komponenten, aus denen das Programm besteht und befindet sich in der obersten Ebene des Projektverzeichnisses. Außerdem werden in ihr die benötigten Berechtigungen sowie etwaige zusätzlich verwendete Bibliotheken vermerkt. Auch Angaben zur mindestens nötigen oder gewünschten Android-Version werden hier eingetragen.

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

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

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

</manifest>
```

Die Komponenten einer Anwendung sind Kinder des Elements *<application/>*. Wenn man im Assistenten zum Anlegen neuer Projekte Create Activity mit einem Häkchen versieht und einen Namen einträgt, enthält das Manifest ein Element *<activity />*. Dessen Attribut android:name beinhaltet den im Assistenten eingegebenen Activity-Namen. Wenn man manuell eine Activity-Klasse anlegt (eine Klasse anlegt und mit *extends Activity* versieht), muss man nachträglich die erzeugte Activity im Manifest bekannt machen. Mithilfe des Elementes *<intent-filter />* kann man eine Activity zur Haupt-Activity machen. Dessen Kindelement *<action />* kennzeichnet die Activity als Haupteinstiegspunkt in die Anwendung. *<category />* sorgt dafür, dass sie im Programmstarter angezeigt wird.

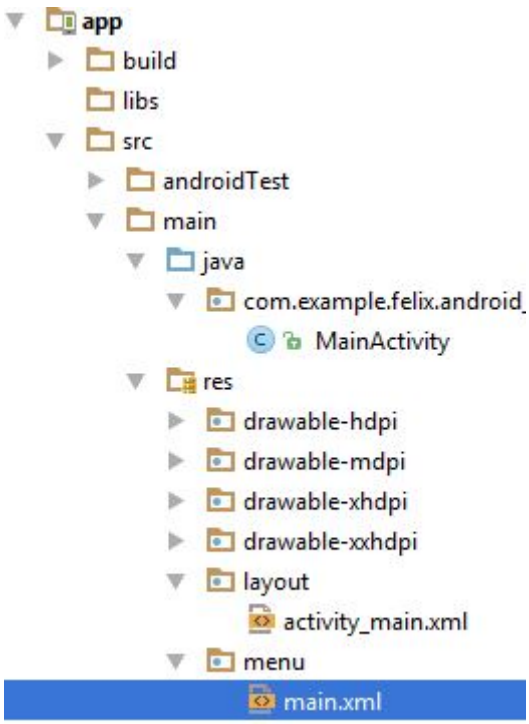
## 1.2 strings.xml




```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">Android_Zusammenfassung</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>

</resources>

```

Elemente haben einen Titel. Üblicherweise werden sämtliche Titel in die String-Resource Datei *string.xml* eingetragen und über *@string/...* referenziert. Die Speicherung von Texten an einem zentralen Ort hat zahlreiche Vorteile. Beispielsweise werden identische Textteile leichter entdeckt, als wenn diese in den Quelltexten der Klassen verborgen sind und man erst jede Klasse oder Layout-Datei durchsuchen muss. Damit lässt sich, wenn auch in eher bescheidenem Umfang, Speicherplatz sparen. Außerdem macht die Trennung von Daten und Programmlogik die Internationalisierung, also die Übersetzung einer App in verschiedene Sprachen, viel einfacher. Hierzu wird für jede zu unterstützende Sprache im Ordner *res* ein Verzeichnis angelegt. Dessen Name beginnt mit *values-* und endet mit dem ISO-Sprachschlüssel. Für Deutsch ist dies *de*. Das Verzeichnis muss also *values-de* heißen. Jeder dieser Ordner erhält eine eigene Version von *strings.xml*. Deren Bezeichner sind stets gleich, die Texte liegen hingegen in den jeweiligen Sprachen vor. Texte in der Standardsprache verbleiben in *values*.

## 1.3 main.xml




```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context=".MainActivity" >
    <item android:id="@+id/action_settings"
        android:title="Settings"
        android:orderInCategory="100"
        android:showAsAction="never"
    />
</menu>
```


Die *main.xml* ist eine sogenannte Menu-Resource-Datei. Menu-Resource-Dateien sind im Verzeichnis '*src/menu*' hinterlegt und dienen dazu Options-Menüs oder Kontextmenüs zu erstellen (Erläuterungen dazu im späteren Kapitel Menüs), die man z.B. mit der Menütaste des Smartphones betätigen aufrufen kann. Menü-Items sollten immer in solchen Menü-Dateien aufgebaut werden statt in den Activity-Dateien. Man kann die erstellten Menüs später in Activities oder Fragments einbinden.

Das Element *<menu />* dient dazu ein *Menu* anzulegen, welches einen Container für alle anderen Menü-Elemente. Es muss das Wurzelelement der XML-Datei sein und es kann

mehrere Elemente des Typs *item* oder *group* enthalten.
Durch <*item* /> wird ein Menü-Element angelegt, welches widerum ein geschachteltes <*menu* /> enthalten kann, um quasi ein Submenü erstellen zu können. *item*-Elemente können widerum Attribute besitzen um deren Erscheinungsbild und das Verhalten zu definieren, z.B. eine *id*, ein *icon*, ein *titel* (Resource-String in der string.xml definieren!) oder *showAsAction*, welches angibt wie ein Menü-Element in der ActionBar erscheint und z.B. die Werte *never*, *always* annehmen kann.
<*group* /> ist ein optionaler, nicht sichtbarer Container um Menü-Bestandteile zu kategorisieren, sodass diese dann bestimmte Eigenschaften miteinander teilen können.

## 1.4 activity_main.xml


```
<RelativeLayout xmlns:android="http://schemas.androi
    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=".MainActivity">

    <TextView
        android:text="Hello world!"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</RelativeLayout>
```


Die *activity_main.xml* ist die Layout-Datei der Main-Activity, also der Startseite des Programmes. Hier können nun entweder über XML selbst (wie im Bild) oder über den Designer View-Elemente hinzugefügt werden. Bei letzterer Herangehensweise wird das

XML-Dokument automatisch generiert, man kann sich also die Oberfläche "zusammenklicken".
Das User-Interface besteht aus einem Wurzelelement, welches eine Art Container für alle anderen View-Elemente darstellt. Das Wurzelelement kann z.B. eine ViewGroup sein (LinearLayout, ListView,...). Auf jeden Fall muss die XML Datei den XML-Namespace (*xmlns* = ...) enthalten. Er Teil des öffnenden Tags des Wurzelelementes. Desweiteren werden die Attribute *layout_width* und *layout_heigth* benötigt. Also sieht die kleinstmögliche Layout-Datei für die MainActivity folgendermaßen aus:

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

Meistens bekommen die Attribute *layout_width* und *layout_heigth* für das Wurzelelement beide den Wert *fill_parent* damit das Layout den gesamten Bildschirm des Gerätes einnimmt.

# 2 Layouts, Views und Komponenten

## 2.1 Layouts

Jede Android-App besteht aus mindestens einer oder mehreren Activities und Layouts. Ein Layout ist ein Rahmen mit speziellen Eigenschaften zur Ausrichtung und Anordnung von View- Elementen auf der Oberfläche. Es gibt in Android keine Klasse Layout, jeder einzelne Layout-Typ ist eine eigene Klasse, die die Klasse ViewGroup erweitert. Layouts können allerdings nicht nur View-Elemente aufnehmen, sondern auch wieder Layouts. Damit ist eine Verschachtelung der Oberflächenelemente möglich.
Folgende Layout-Tags sind möglich:

- *LinearLayout*:
  - alle View-Elemente, je nach Orientierung, Element für Element untereinander oder nebeneinander aneinandergereiht dargestellt
- *TableLayout*:
  - Möglichkeit, gleichförmig zusammenhängende Oberflächen spalten- und zeilenbasiert zu erstellen (z.B. Eingabeformulare,...)
- *GridLayout*:
  - ähnlich dem TableLayout, kann jedoch so einfach wie das LinearLayout angewendet werden

  - orientiert sich wie das TableLayout bei der Ausrichtung an Zeilen und Spalten
  - Grafikelemente, die eine unterschiedliche Größe besitzen können sehr effizient positioniert werden
  - Größe jeder einzelnen Zelle wird dynamisch angepasst

- *RelativeLayout*
  - alle View-Elemente werden relativ zueinander positioniert
  - Ausrichtung eines View-Elements erfolgt am äußeren Rahmen, der vertikalen bzw. der horizontalen Mitte oder dem vorhergehenden View-Element

- *FrameLayout*:
  - einfachstes und performantestes Layout
  - alle Layout-Elemente werden in der linken oberen Ecke ausgerichtet und übereinander gestapelt

## 2.2 Views

Die Basisklasse aller Bedienelemente ist android.view.View. Die Benutzeroberfläche einer App besteht also aus einer oder mehreren Views oder von ihr abgeleiteten Klassen. Sie kümmern sich um die Bearbeitung von Tastatur-, Touch- und Trackball-Ereignissen. Zum Ermitteln von Referenzen auf spezifische Komponenten dient die Methode *findViewById()*.

## 2.3 Basiskomponenten einer App

Jede Android-Anwendung besteht aus einem oder mehreren Bausteinen (Basiskomponenten). Jede Basiskomponente kann mehrfach in einer Android-Anwendung vorkommen. Alle Basiskomponenten haben einen Lebenszyklus (Lifecycle). Diesen Lifecycle gilt es bei der Programmierung zu beachten. Abhängig vom Lifecycle der Komponente entscheidet sich, ob Daten oder Funktionen zur Laufzeit zur Verfügung stehen, oder ob diese verloren gehen.

### 2.3.1 Activity

Activities leiten von android.app.Activity oder deren Kindern ab. Normalerweise ist jeder Activity eine Benutzeroberfläche, also ein Baum bestehend aus Views und ViewGroups, zugeordnet. Activities bilden demnach die vom Anwender wahrgenommenen Bausteine einer App. Sie repräsentieren also meist die Benutzeroberfläche und Interaktionen. Jede Android- Anwendung besteht deshalb aus mindestens einer Activity. Activities können

andere Activities aufrufen und mit ihnen Daten austauschen. Jede Activity besteht aus einer in XML definierten Layout-Datei und einer dazugehörigen Java-Klassendatei, welche bei Android Studio im Verzeichnis '*%Appname/src/main/java/%packagename/*' abgelegt ist. Die dazugehörige Layout-Datei (XML-Datei) ist im Verzeichnis '*%Appname/src/main/res/layout*' zu finden.
Wichtige Methoden:

- protected void onCreate(Bundle savedInstanceState)
  - wird von praktisch jeder selbst geschriebenen Activity überschrieben
  - Android ruft sie während der Initialisierungsphase einer Aktivität auf
  - Aufgaben:
    - Aufrufen der gleichnamigen Elternmethode: *super.onCreate()*, sonst gibt es zur Laufzeit eine SuperNotCalledException
    - Initialisieren von Instanzvariablen mittels *findViewById()* , z.B. buttonAdd = (Button) findViewById(R.id.buttonAdd);
    - Setzen der Benutzeroberfläche durch *setContentView(setContentView(R.layout. <Layout-Datei >)* wobei die Endung *xml* weggelassen wird
    - Wiederherstellen eines gespeicherten Zustands um Wiederanlaufzeiten einer Activity zu optimieren. Android stoppt Activities unter bestimmten Umständen automatisch, z.B. beim drehen des Gerätes, so wird vorher der Zustand in einem Zwischenspeicher *savedInstanceState* gesichert.
- onCreateOptionsMenu(Menu menu)
  - Ereignisbehandlung der Menüeinträge in der ActionBar erfolgt durch *actions*
  - diese Methode ist für die Erstellung des Menüs zuständig, es wird das Menü aufgeklappt und es werden Items hinzugefügt
- onOptionsItemSelected(MenuItem item)
  - wird immer dann ausgeführt, wenn ein Menü-Item betätigt wurde
  - hier erfolgt die Ermittlung, welcher Menüeintrag betätigt wurde, z.B. mit switch(item.getItemId()) { case R.id.main_menu_about: ... }

### 2.3.2 Events

- Klick-Ereignisse
  - es muss festgelegt werden welche View-Elemente ein Klick-Ereignis auslösen und was mit diesem Ereignis geschehen soll

  - man kann dem Attribut *onClick* eines Items einen Wert zuweisen, z.B. *onClick* (*android:onClick="onClick"*), wodurch die Methode festgelegt wird, die bei einem Klick-Ereignis im Code aufgerufen werden soll
  - die Methode *onClick(View view)* muss in jedem Fall in der Java-Klasse der Activity definiert werden
  - man kann diese Methode auch mittels Erweiterung der Klasse durch *implements View.onClickListener* implementieren
  - hier ist eine switch/case-Anweisung sinnvoll, die die Id der View auswertet, erweitert
    switch (view.getId()) { case R.id.mainActionBar_imageButton1: ... }

- Laden eines anderen Layouts (eine Activity) aus der aktuellen Activity mittels Intents (Definition *Intent* im nächsten Abschnitt)

- Zustandsänderungen im System

### 2.3.3 Intents

Intents werden benötigt um eine Activity aus der aktuellen Activity zu starten oder Informationen an eine andere Activity weiterzugeben. Vereinfacht ausgedrückt sind Intents Nachrichten bzw Absichtserklärungen, welche Aktion ausgeführt werden soll. Es gibt

- Explizite Intents
  - senden ihre Absicht an einen Empfänger
  - der Empfänger muss explizit angegeben werden
  - Empfänger können Komponenten einer Android-App, also Activities, Services oder BroadcastReceiver sein
  - Empfänger wird über den voll qualifizierten Klassennamen angesprochen
  - Beispiel

```
Intent intent = new Intent(this, TestActivity.class);
intent.putExtra("message", "inhalt der message");
startActivity(intent);
```

    * mit dem Intent werden Informationen (.putExtra(...)) an die TestActivity übergeben (können verschiedenste Informationen sein)
    * Information werden als Key-Value-Paar übergeben
    * in der TestActivity ist es dann möglich, die übergebenen Informationen auszuwerten

- Implizite Intents

- senden ihre Nachricht/Absicht ab, aber der Empfänger steht noch nicht fest (es ist nicht klar welche Komponente genau sich der Nachricht annimmt)
- Beispiel bei Windows: nach einem Rechtsklick auf eine Datei, wählt man Senden an und dann E-Mail-Empfänger, so startet automatisch das Standard-E-Mail-Programm Ihres Systems
- Intents erklären, was ihre Absicht ist, und das System entscheidet, welche Komponente sich darum kümmert
- Beispiel

```
Intent intent = new Intent(Intent.ACTION_VIEW,
Uri.parse("http://www.scyte.eu"));
startActivity(intent);
```

  * impliziter Intent startet Browser und ruft die Web-Adresse *www.scyte.eu* auf

Mit einem expliziten Intent lässt sich aus einer Activity eine neue Activity starten:

```
case R.id.mainBottomActionBar_imageButton_textNote:
   startActivity(new Intent(this, NoteTextActivity.class));
   break;
```

### 2.3.4 Fragments

Fragments kann man als Teile einer Activity verstehen. Fragments bestehen, wie auch Activities, aus einer Java-Datei und einem Layout und haben einen Lebenszyklus (Lifecycle), der der Activity ähnlich ist. Jedoch benötigen Fragments immer eine Activity, die sie umgibt und in deren Lifecycle sich Fragments integrieren. Der Nutzen von Fragments liegt darin, dass Teile eines Layouts mit Code in kleinere und wiederverwendbare Einheiten ausgliedert und somit schneller und effizienter Anwendungen für verschiedene Gerätetypen oder Displaymodi erstellt werden können. Mann kann diese Fragmente also in Layouts für verschiedene Auflösungen und Orientierungen des Bildschirms einbinden und wiederverwenden.
Fragmente leiten entweder direkt von der Basisklasse *android.app.Fragment* ab oder von einem ihrer spezialisierten Kinder, beispielsweise *ListFragment* oder *DialogFragment*. Beispiel:

```
public class TestFragment1 extends Fragment {

   @Override
   public View onCreateView(LayoutInflater inflater,
         ViewGroup container,
         Bundle savedInstanceState) {
      return inflater.inflate(R.layout.fragment_layout,
         container,
         false);
   }

   @Override
   public void onStart() {
      super.onStart();
      TextView tv = (TextView) getView();
      tv.setText(getString(R.string.text1));
   }
}
```

*TestFragment* überschreibt zwei Methoden. *onCreateView()* wird aufgerufen, wenn ein Fragment den Komponentenbaum seiner Benutzeroberfläche instanziieren soll.
Fragment in Activity in Activity einbetten:

```
<fragment
   android:id="@+id/fragment"
   android:name=
      "com.thomaskuenneth.fragmentdemo.TestFragment1"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
/>
```

In diesem Ausschnitt wird das Fragment in die Layout-Datei einer Activity eingebettet. Das Attribut android:name legt fest, welche Klasse das hier definierte Fragment implementiert (nämlich die aus obigem Beispielcode).
Im Gegensatz zu Activities, Services oder Broadcast Receivern trägt man Fragmente nicht in die Manifestdatei ein. Man verbindet sie mit einer Activity, indem man sie in deren Benutzeroberfläche integriert. Dies geschieht normalerweise deklarativ in Layoutdateien.

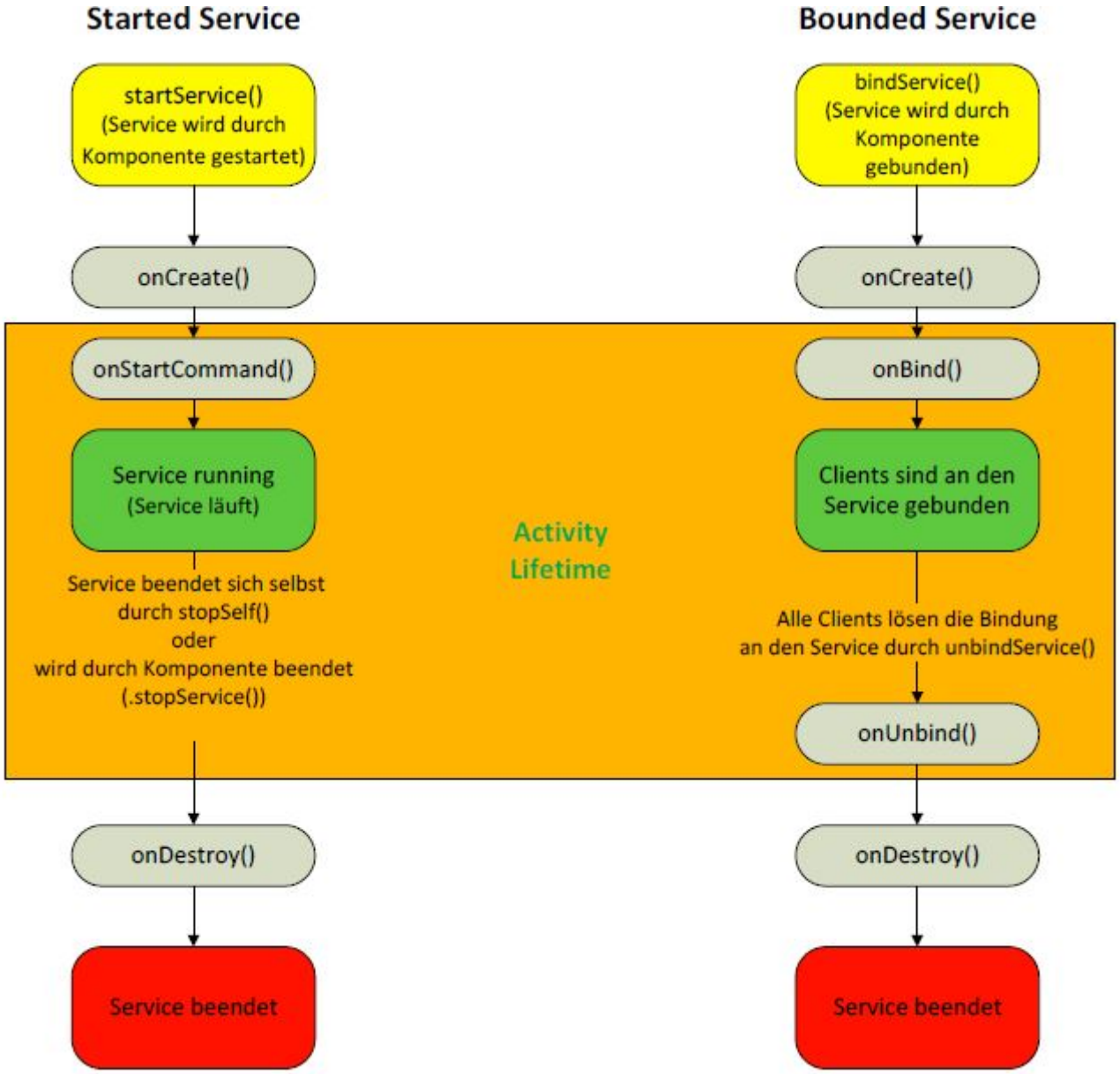
### 2.3.5 Services

Services haben kein User-Interface und dienen zur Abarbeitung von Aufgaben im Hintergrund. Sie sind gerade für länger andauernde Tätigkeiten gedacht. Es gibt 2 Formen von Services:

- Started:
  - wird von einer anderen Anwendungskomponente gestartet, läuft eigenständig

und beendet sich selbst nach Abarbeitung seiner Aufgabe

- Bound:
  - wird an eine oder mehrere Anwendungskomponenten mit bindService() gebunden und bietet eine Client-Service-Schnittstelle zur Kommunikation
  - wird beendet, sobald keine Komponente mehr an ihn gebunden ist




### 2.3.6 Menüs

Es gibt nun zwei Arten von Menüs, das Optionsmenü und das Kontextmenü. Das Optionsmenü ist mit einer klassischen Menüleiste anderer Anwendungen vergleichbar und sollte nur Funktionen enthalten, die für die aktuelle Activity sinnvoll sind. Zusätzlich ist eine Hilfe- und Informationsseite der Anwendung sinnvoll. Das Kontextmenü ist in

klassischen Anwendungen meist durch einen Rechtsklick auf ein Objekt aufrufbar. Dieser Rechtsklick wird in Android durch das lange Antippen eines Objekts auf der Oberfläche ersetzt.

**Optionsmenüs** Die Erstellung eines Menüeintrages in der ActionBar ist schnell erledigt. Standardmäßig wird vom Projektassistenten nämlich schon ein Menüeintrag in der Mainactivity erstellt.
Um das Optionsmenü für eine Activity zu spezifizieren, muss die Funktion *onCreateOptionsMenu(Menu menu)* überschrieben werden.

```
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;
}
```

Neue Menüeinträge müssen in der Datei *main.xml* definiert werden.

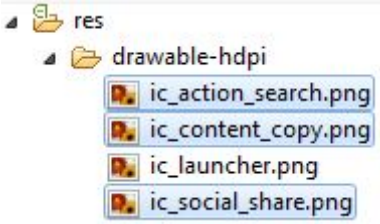
```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context=".MainActivity" >
    <item android:id="@+id/action_settings"
        android:title="Settings"
        android:orderInCategory="100"
        android:showAsAction="never"
    />
</menu>
```


Die Menüdefinition erfolgt durch <*menu* /> Tags, welche die Menüeinträge als <*item* /> enthalten. Die Items können selbst natürlich widerum Menüs enthalten, wodurch Untermenüs entstehen.
Wie auf obigem Bild zu sehen ist, wird das Menü aufgeklappt und damit der Menüeintrag *Settings* angezeigt sobald die Menütaste des Gerätes betätigt wird. Im XML-Quellcode

der Menü-Datei wird dies durch die Zeile *android:showAsAction="never"* festgelegt. Das Attribut *android:showAsAction=" "* legt fest, wann und wie der jeweilige Menüeintrag in der ActionBar angezeigt wird. Man kann solche Menüeinträge auch für immer in der ActionBar anzeigen lassen, ohne dass die Menütaste des Gerätes betätigt werden muss. Dies geschieht durch Änderung der eben besprochenen Zeile in: *android:showAsAction="always"*.


Mit *android:showAsAction="ifRoom"* werden die Menüeinträge in der ActionBar angezeigt, sofern genügend Platz vorhanden ist. Ist dort nicht genügend Platz vorhanden, wird der Eintrag in der Liste des Optionsmenüs angezeigt.
Menüeinträge können auch **Icons** zugewiesen werden, die anstatt des festgelegten Textes angezeigt werden, z.B. durch die Zeile *android:icon="@android:drawable/ic_delete"*.


Man kann auch eigene Icons erstellen und diese ins Programm einbinden. Hierzu müssen diese in den folgenden Ordner eingefügt werden:




und die icons in der Menü-Ressourcendatei *main.xml* beim entsprechenden Menüelement(item) eingebunden werden.

```xml
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:id="@+id/action_search"
        android:orderInCategory="100"
        android:showAsAction="always"
        android:icon="@drawable/ic_action_search"
    <item
        android:id="@+id/action_copy"
        android:orderInCategory="100"
        android:showAsAction="always"
        android:icon="@drawable/ic_content_copy"
    <item
        android:id="@+id/action_share"
        android:orderInCategory="100"
        android:showAsAction="always"
        android:icon="@drawable/ic_social_share"
</menu>
```

Um auf **Klickereignisse** reagieren zu können, muss die Funktion *onOptionsItemSelected(MenuItem item)* überschrieben werden.

```java
public boolean onOptionsItemSelected(MenuItem item)
{

    int id = item.getItemId();
    if (id == R.id.action_settings)
    {
        return true;
    }
    return super.onOptionsItemSelected(item);
}
```

Das vom Nutzer betätigte Menüelement kann mittels Aufruf der Funktion *getItemId* identifiziert werden. Sie gibt die eindeutige ID des Menüelements zurück, welche durch das *android:id* Attribut in der Menü-Ressourcendatei *main.xml* definiert wurde.

**Kontextmenüs** Sie werden im Gegensatz zu Optionsmenüs nicht durch Betätigen der Menütaste des Gerätes (oder wie bei Windows durch die rechte Maustaste) ausgelöst, sondern durch langes Antippen (Tippen und Halten) eines Elementes. Besonders beliebt sind die Kontextmenüs im Zusammenhang mit ListView-Elementen. Aber natürlich können auch das lange Antippen beliebiger View Elemente, z.B. Buttons, ein Kontextmenü auslösen. Hierfür muss das jeweilige Viewelement an die Methode *registerForContextMenu* übergeben werden.

```
public class MainActivity extends Activity
{
    private Button button;
    private TextView tv;

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

        button = (Button) findViewById(R.id.button);
        registerForContextMenu(button);

        tv = (TextView) findViewById(R.id.textview);
    }
```

...

Als nächstes muss die Methode *onCreateContextMenu()* übrschrieben werden. Hier muss das Menü angegeben werden, welches beim langen Antippen des View-Elementes angezeigt werden soll.

```
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenu.ContextMenuInfo menuInfo)
{
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.main, menu);
}
```

Um auf **Klickereignisse** reagieren zu können muss die Methode *onContextItemSelected* überschrieben werden. Hier gibt man für jedes Menü-Item des angezeigten Kontextmenüs an, was jeweils ausgeführt werden soll (wie schon bei Optionsmenüs).

```
public boolean onContextItemSelected(MenuItem item)
{
    switch (item.getItemId())
    {
        case R.id.menu_item_1:
            tv.setText(item.getTitle());
            return true;
        case R.id.menu_item_2:
            tv.setText(item.getTitle());
            return true;
        default:
            return super.onContextItemSelected(item);
    }
}
```

# 3 Allgemeiner Ablauf bei einer einfachen Beispiel-App

## 3.1 Anlegen eines Projektes und Erstellung einer MainActivity

## 3.2 Layout festlegen und Views hinzufügen

## 3.3 ID und Namen eines jeden Views festlegen

## 3.4 Einbinden der View-Elemente in die Activity-Klasse

## 3.5 Actionbar mit Image-Buttons erstellen

## 3.6 Intents zum Aufruf einer Activity aus der aktuellen Activity

## 3.7 Anmelden der Activities im Manifest