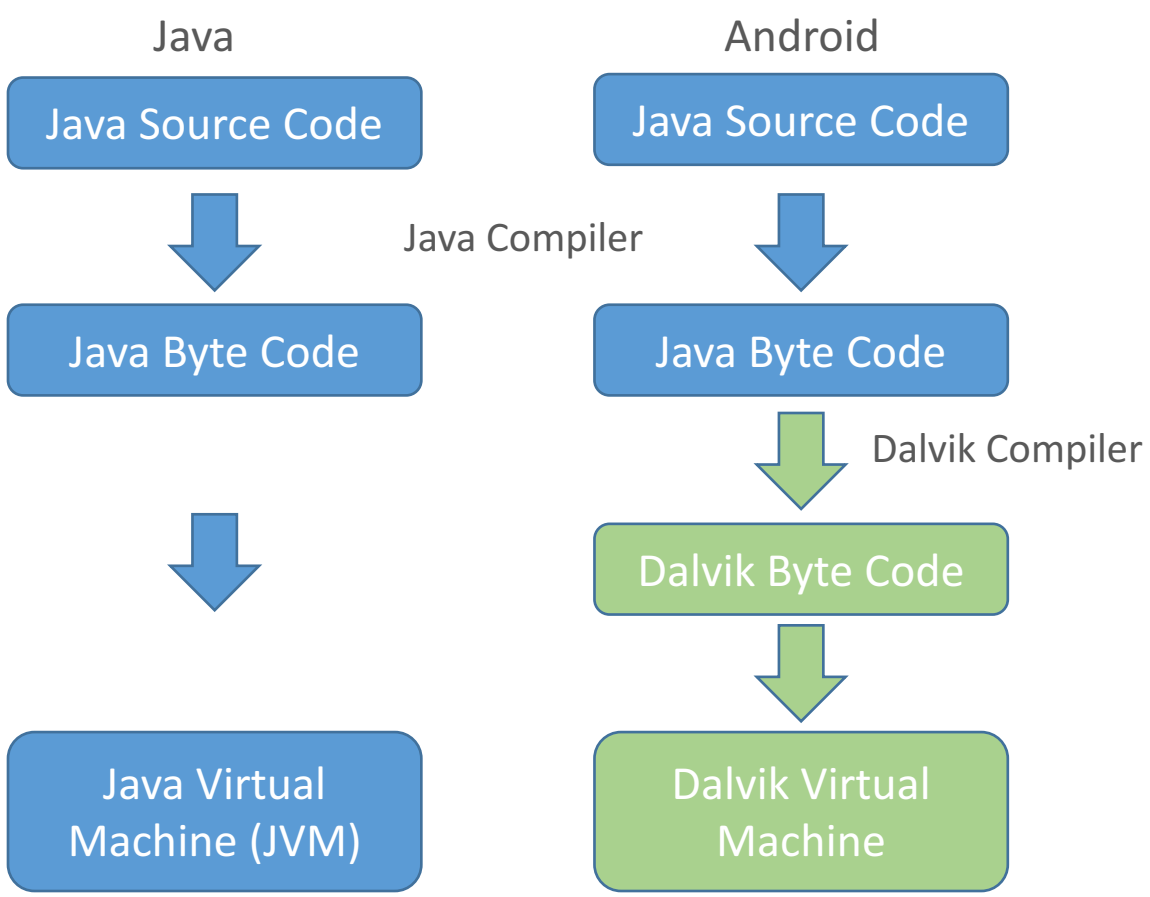
Android Developer Tools

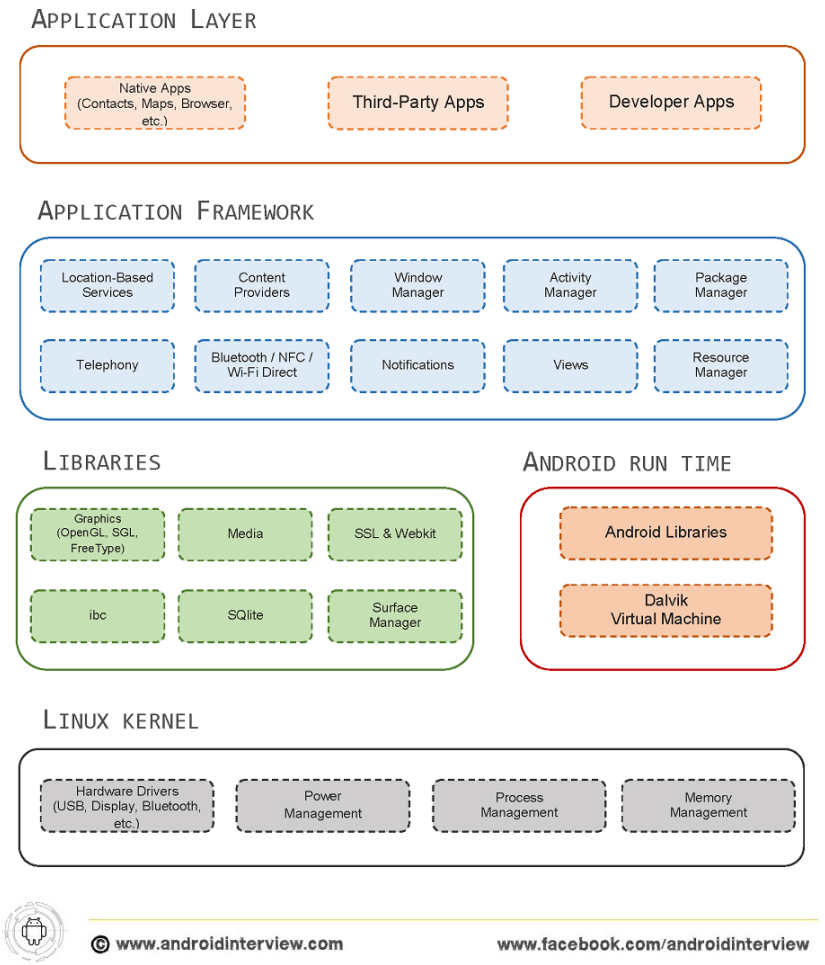
- **Android Studio (AS)**
- SDK Manager in AS
- Android Virtual Device Manager (AVD)
- Android Emulator
- Android Device Monitor

Android System I

- **Android < 5.0 (Dalvik Virtual Machine)**
  - Virtual Machine executing byte-code
  - Just-in-time compiler since Android 2.2
  - Optimized for Low memory requirements
  - Compile during every launch of the application
- **Android ≥ 5.0 (Android Runtime)**
  - Ahead-Of-Time (AOT) – Compilation during installation
  - Backward compatible (same byte-code as Dalvik)

Android System II





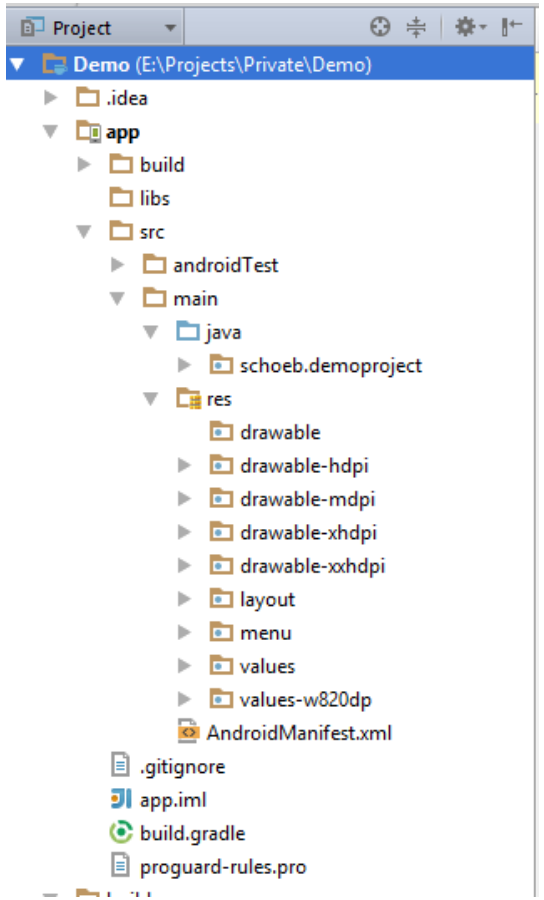
App fundamentals – general I

- Written in Java (C/C++, Kotlin)
- Packed in Android Package file «.apk» (It’s just a zip)
- Installed on the device the app runs in it’s own sandbox
  - Android is a multi-user-linux in which each app is a different user
  - Files are only accesable by this user
  - Each app runs in it’s own process (normally…)
  - Each process has it’s own VM
- Permissions give access to specific resources (e.g. SMS, Contact, Locations,…)

App fundamentals – general II

- A app consists of loosley coupled components
  - Activites, Services, Content Provider and Broadcast Receiver
- Components are hold together by «intents»
  - Some kind of messages to trigger another component
- Meta information about the app is stored in the «Manifest – File»
  - Permissions, available components, Broadcast Receivers, Services
- Application resources
  - Layouts, images, raw data , localizations, sounds,….

App fundamentals – Project structure



Folder	Description
java	Java source code
build	Generated files -> Never touch it !
libs	Include libraries (in addition to gradle dependencies)
res	Resources (layouts, colors, strings,….)

App fundamentals – Manifest

- Every app must have an «AndroidManifest.xml»
- Information about the app
  - Package name
  - Description of Components
  - Permissions (Careful: permission handling changed in Android 6.0)
  - API requirements (Will be overridden by the gradle build file)

App fundamentals – Manifest example

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="ch.schoeb.services“
          android:versionCode="1“
          android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8“ android:targetSdkVersion="17" />

    <application android:icon="@drawable/ic_launcher“
                           android:label="@string/app_name"
                 android:theme="@style/AppTheme" >
        <activity android:name="ch.schoeb.services.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>
    </application>
</manifest>
```

App fundamentals – Resources

- All data which is not code
- Stored in the «res»-folder
- Xml files
  - Layouts
  - Strings
  - Values (colors, dimensions,…)
  - Images (also vector graphics)
- Access these resources by the @-annotations

Declaration in Strings.xml:

```
<string name="app_name">Services</string>
```

Usage in any other XML-File:

```
android:label="@string/app_name"
```

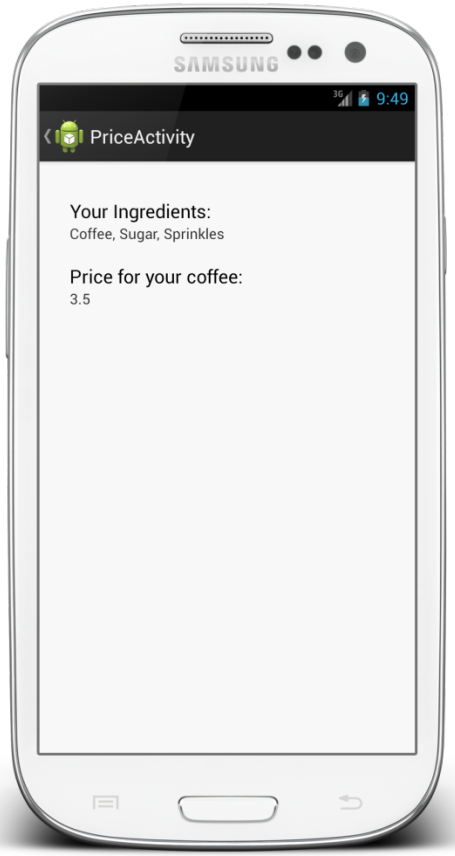
App fundamentals – R - Class

- Located in your «build/…» folder
- Automatically generated in each build
- Containts all resource Ids from your res-folder as public constants
- Subclasses for all Resources Types:
  - R.drawable, R.layouts, R.id, R.string,…
- Used to access resources in code

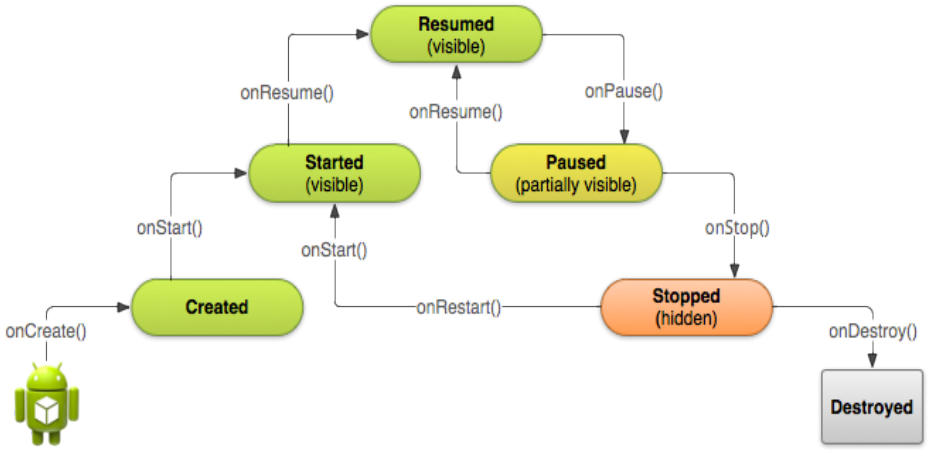
```
public final class R {
    public static final class string {
        public static final int action_settings=0x7f050001;
        public static final int app_name=0x7f050000;
}
```

Activity - Overview

- Represents a single screen a user can interact with
- The UI is either defined in xml or directly in the code
- Normally a application consists of multiple activities
- One activity can consists of multiple fragments
- Beware of the **Activity lifecycle**
- Your classes must extend **Activity** or even better **AppCombatActivity**



Activity - Lifecycle

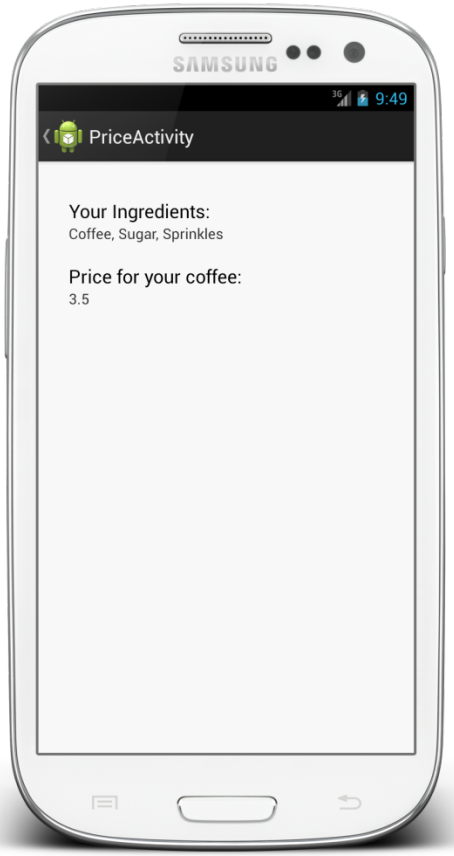


- **onCreate():** Called when the activity is first created
- **onStart():** Called when the activity becomes visible to the user
- **onResume():** Called when the activity starts interacting with the user
- **onPause():** Called when the current activity is being paused and the previos activity is being resumed
- **onStop():** Called when the activity is no longer visible to the user
- **onDestroy():** Called before the activity is destroyed by the system
- **onRestart():** Called when the activity has been stopped and is restarting again

Activity – Connect to the layout

PriceActivity.java
- Extend Activity class (better AppCompatActivity)
- Use lifecycle methods to connect to xml
- Use findViewById() to access view

activity_price.xml
- Declarative xml to define UI
- Define ID's for every view



Activity - Layout

activity_price.xml

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

    <Button
        android:id="@+id/my_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="MyButton" />

</RelativeLayout>
```

Activity – Connect to layout

Set layout for activity:

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

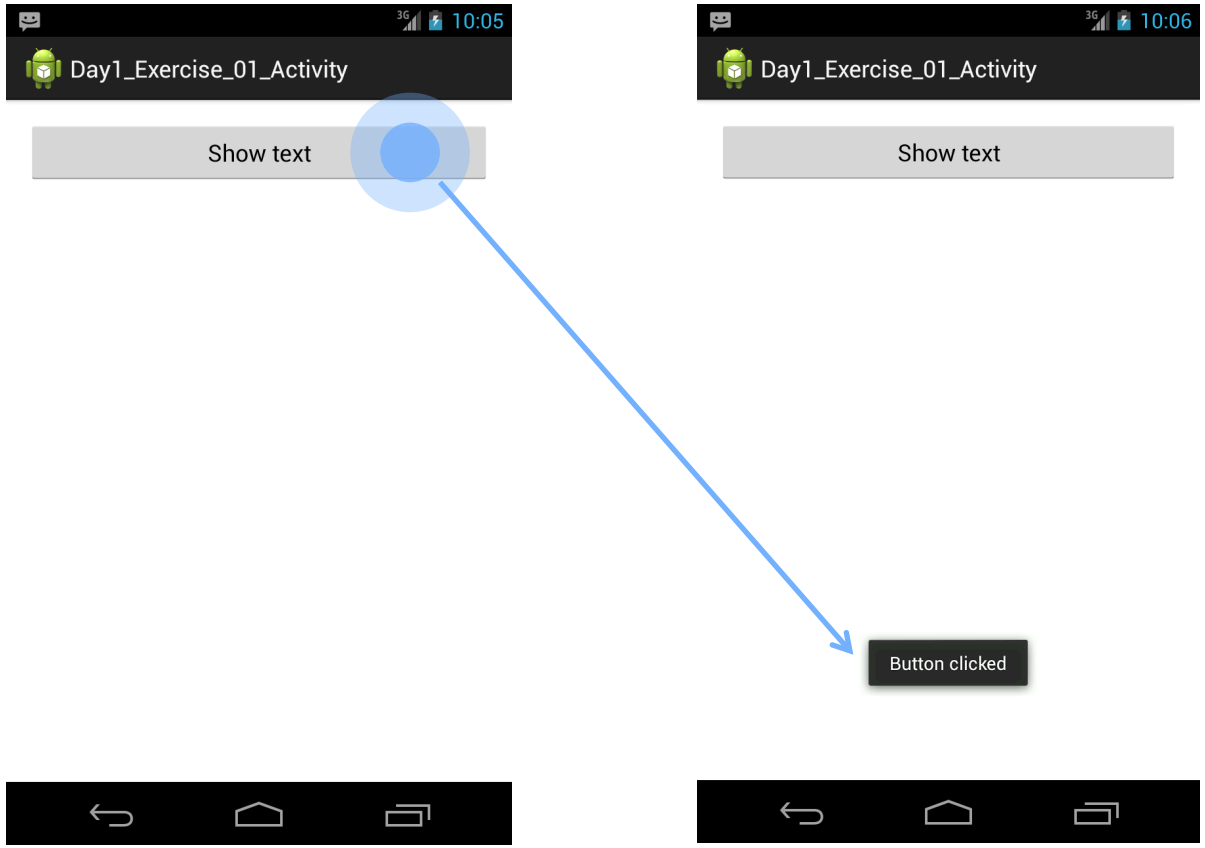
Access views in activity:

```
TextView textView= (TextView) findViewById(R.id.demoTextView);
```

There are easier ways to bind the views in the activities, which are not covered in this class:

- Android Databinding (bind views bi-directional to a Java model)
- Butterknive (external library)

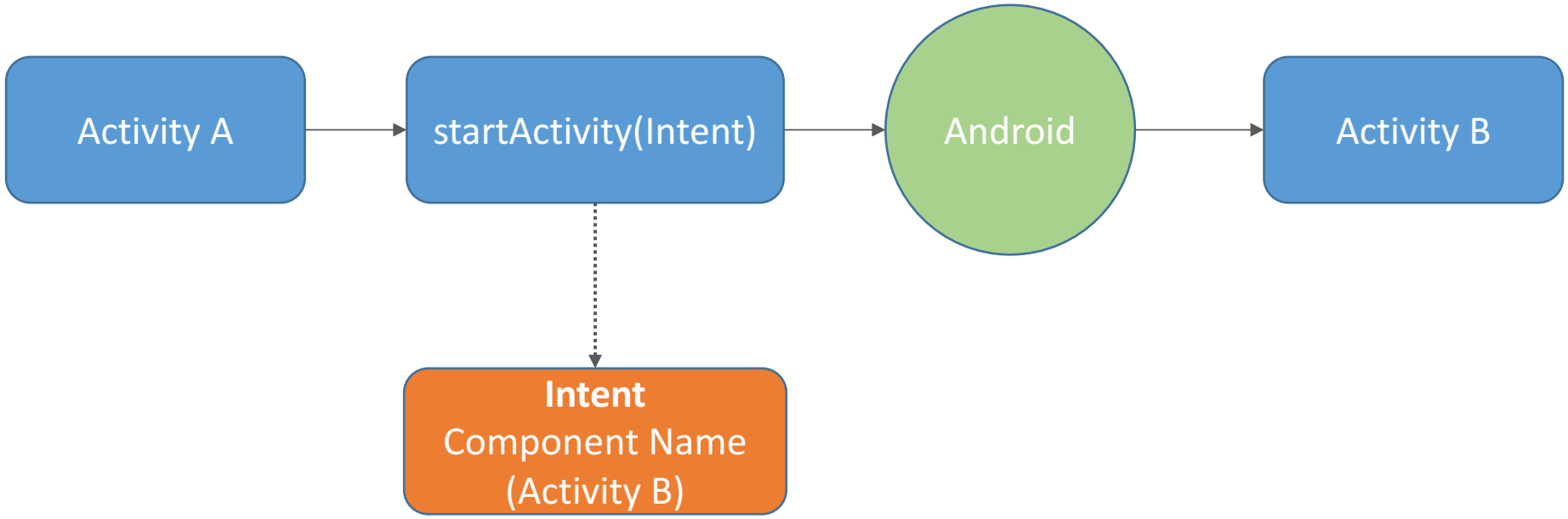
Activity – Exercise01_Activity



Intents - Overview

- An intent is a message to start another component (Not only activities)
- There are two types of Intents
  - **Explicit** (Specify the component to start by name (the fully qualified class name), typically used to start a component in your own app
  - **Implicit** (Do not name a specific component name, instead declare a general action to be performed, which allows a component from another app to handle it)
- An intent can contain data
  - Component names, actions, categories, extras, flags
- **The target component can be in another application!**

Intents - Explicit

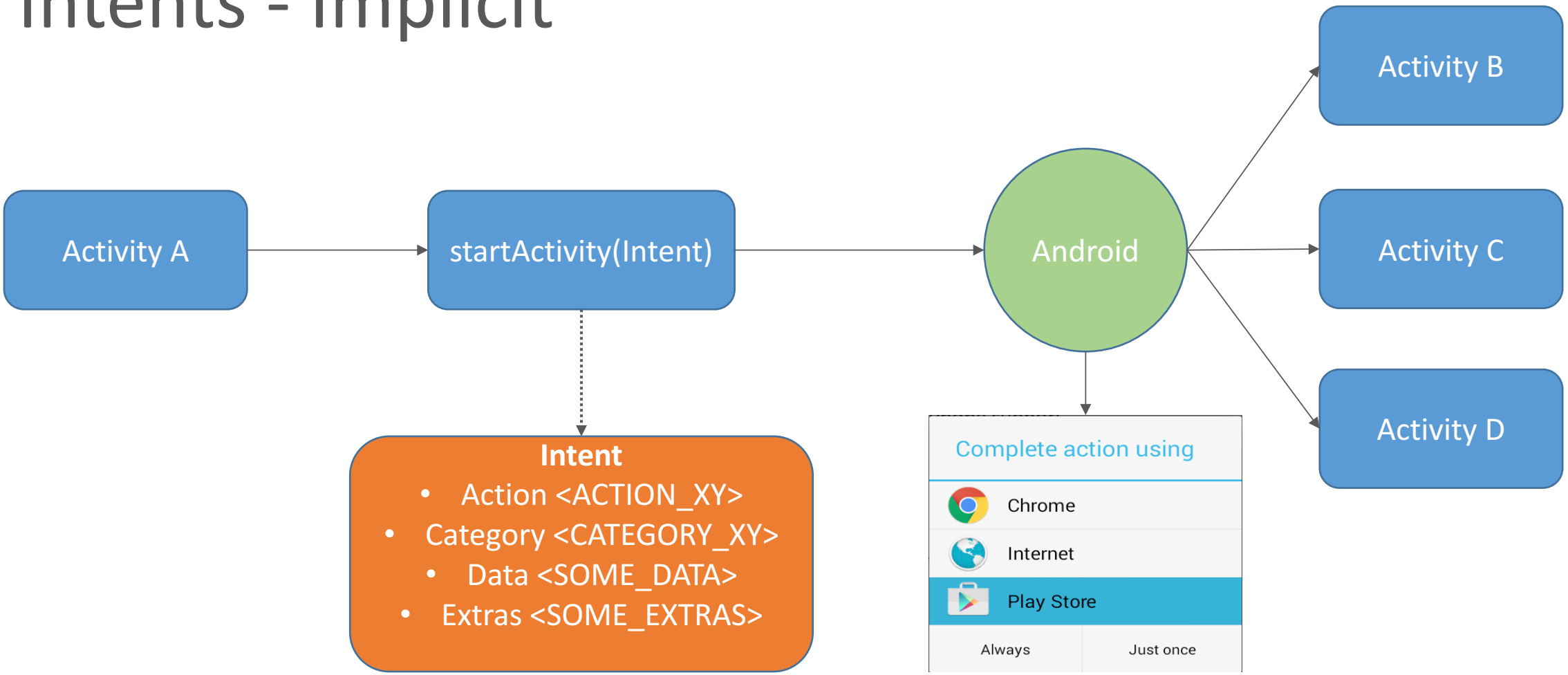


Intents – Explicit Example

```
// Intent erstellen (this = Context)
Intent intent = new Intent(this, TargetActivity.class);

// Daten in den Intent packen für die neue Activity
intent.putExtra("MyKey", "Die Daten");

// Neue Activity durch Android starten
startActivity(intent);
```

```
// In der onCreate()-Methode der TargetActivity:
// Intent in TargetActivity abfragen
Intent intent = getIntent();

// Extra Daten abfragen
String data = intent.getStringExtra("MyKey");
```

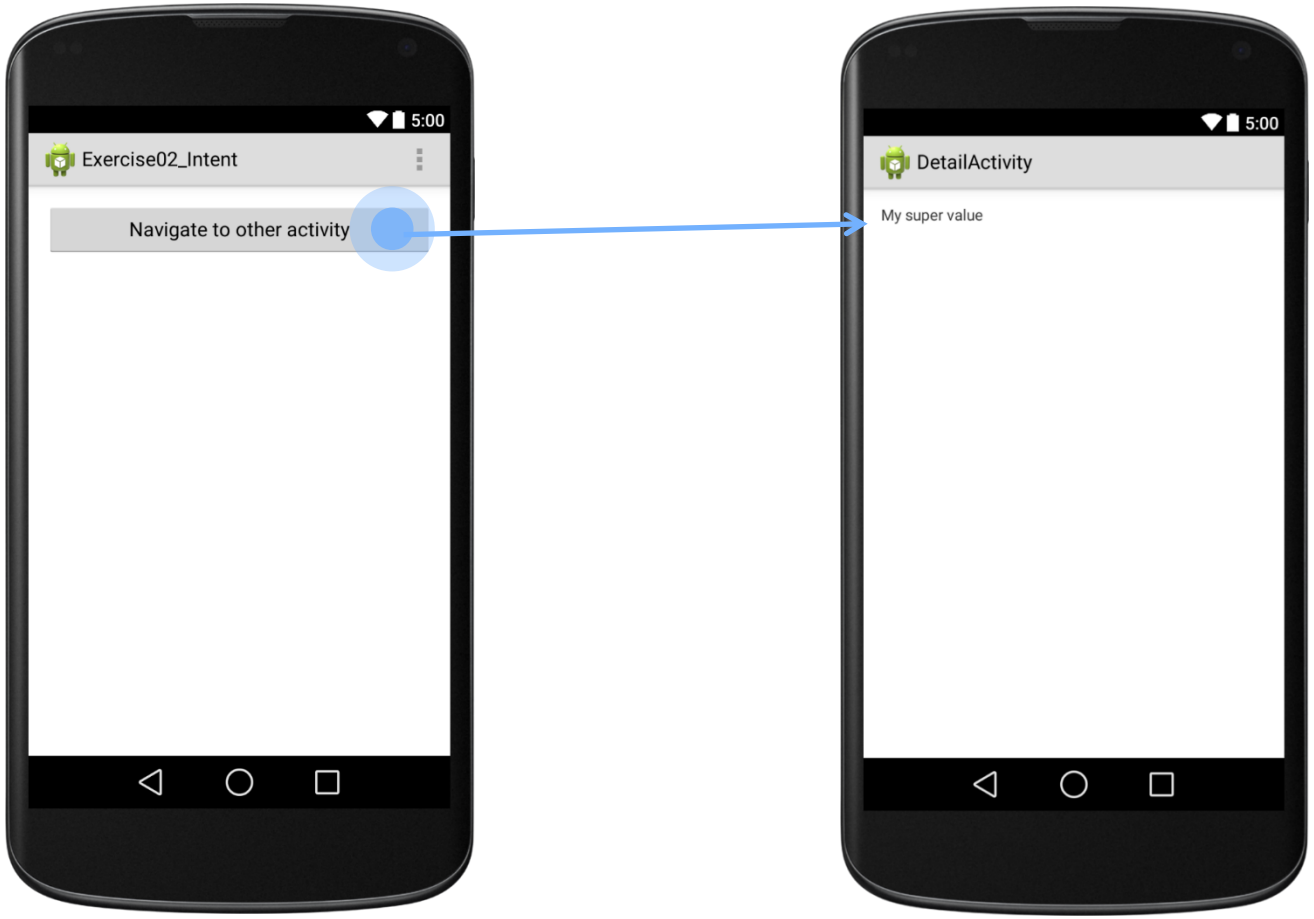
Intents - Implicit



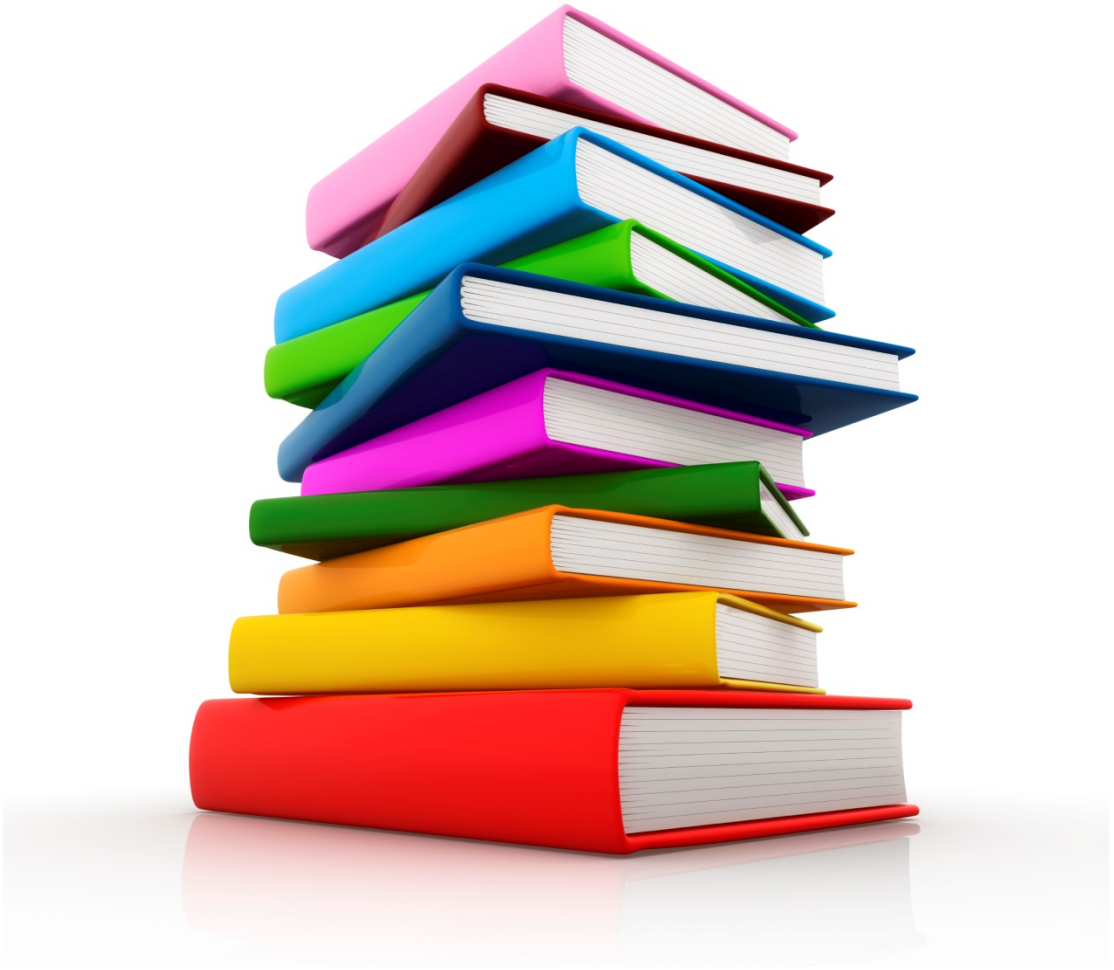
Intents – Implicit Example

```
// Create a text message with a string
Intent intent = new Intent();
intent.setAction(Intent.ACTION_SEND);
intent.putExtea(Intent.EXTRA_TEXT,"My awesome Text message");
intent.setType ("text/plain");
// Verify that the intent will resolve to a activity
if(intent.resolveActivity(getPackageManager()) != null){
        startActivity(intent);
}
```

Intents – Excerise02_Intent



Resources



Resources

Overview

- Split resources from code
  - Layout/strings/images/…
- Resources are all kept in a “res”-folder
- Different resources for different languages or screen sizes
- Default-Resources (Fallback)
  - Used when no special case is defined
  - When not found → App crash (runtime)
- Alternative-Resources
- Used for special case
  - for example landscape is different than portrait

Resources

Providing **Default** resources

- Specific folders in “res” for specific resources
- No other folder-names supported

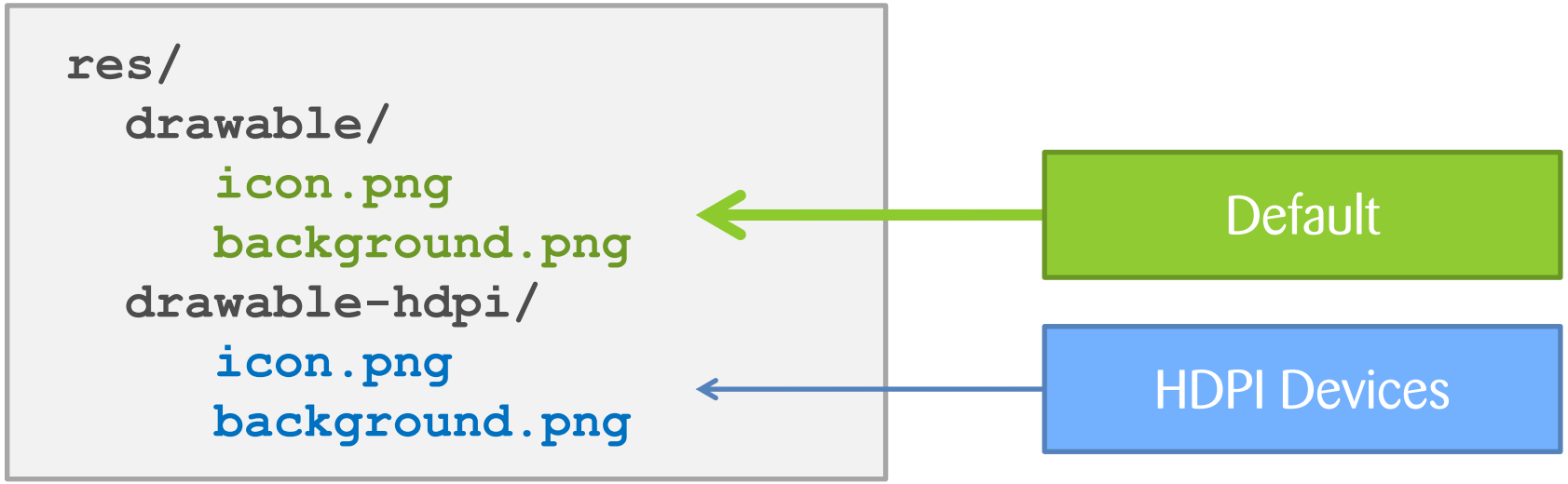
Foldername	Resources
animator	XML files for property animations
anim	XML files for tween animations (View animations)
color	XML files that define state list of colors
drawable	**Bitmap-files or XML**
layout	**XML Layout files**
menu	**XML files defining Option/Context/Sub-Menu**
raw	Other files
values	**XML files with simple values like strings, integers**
xml	Other XML-files that can be read with Resources.getXML()

Resources

Providing **Alternative** Resources

- Same folder structure as default resources
- Additional qualifiers

```
<resourcesfolder-name>-<qualifier>.
```



Resources

Available qualifiers

Qualifier	Example
Language / Region	values-en ; values-de-CH
Screen Size	layout-small ; layout-large
Screen Orientation	layout-land ; layout-port
Screen pixel density (dpi)	layout-mdpi ; layout-hdpi ; drawable-hdpi
Platform version (API Level)	values-v10 ; layout-v8

Other qualifiers:
Mobile Country Code (MCC); Layout Direction; Smallest Width; Available Width; Available Height; Screen aspect; UI Mode; Night Mode; Touchscreen type; Keyboard availability; Primary text input method; Navigation key availability; Primary non touch navigation method

Resources
Providing Alternative Resources

Qualifier Rules

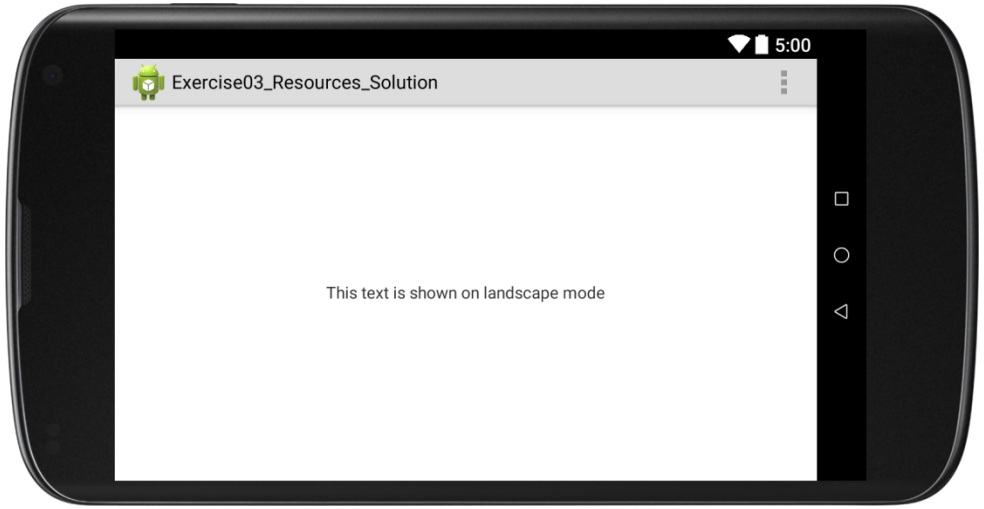
Multiple qualifiers for single resource possible

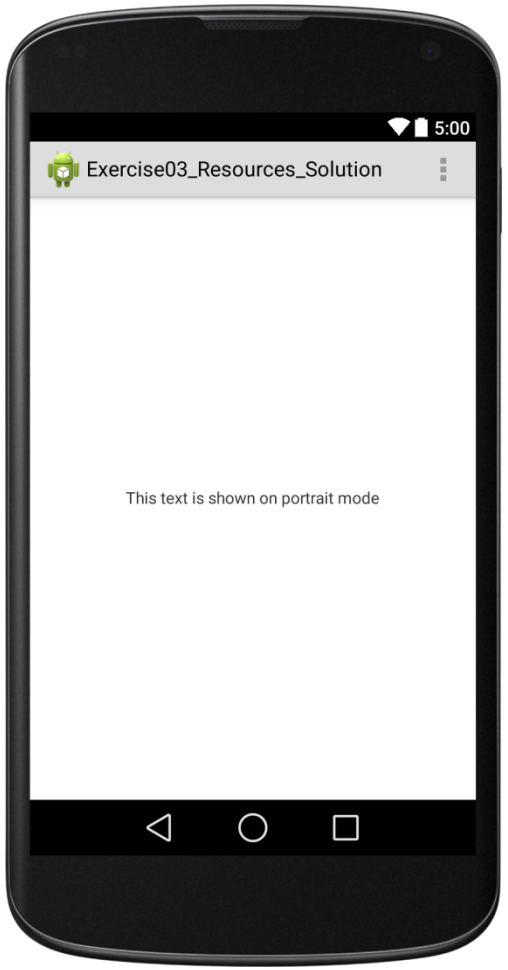
- For example: layout-de-CH-hdpi
- Order of chained qualifiers is important
  - http://developer.android.com/guide/topics/resources/providing-resources.html#AlternativeResources
- Case insensitive
- Multiple qualifier of same type not supported
- For example: “layout-hdpi-mdpi” is not allowed

Exercise 3

Exercise03_Resources

- Show different layout in landscape
  - Use CTRL + F12 to change emulator
  - orientation
- Show a different text on landscape mode



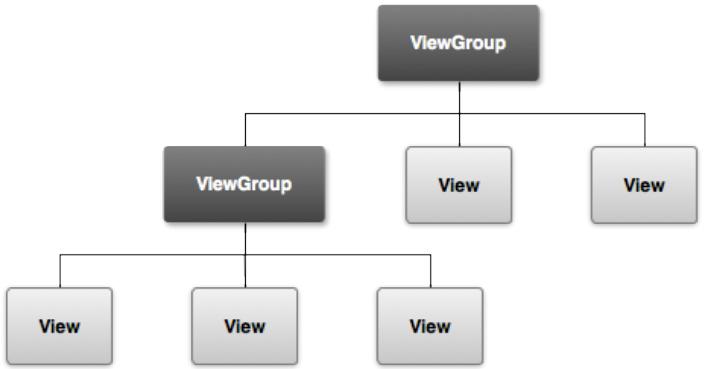


User Interface

User interface

«layouts»

- Layouts are created in XML or in Layouteditor (prefer XML at the beginning to get a better understanding)
  - Placed in res/layouts-folder
- Everything is a View
  - TextView, EditText, Button, Spinner, etc.
- ViewGroup is a container for multiple Views



User interface

XML Elements and Attributes for layouts

- Layout_width / layout_height mandatory
- Id-attribute to link (find) the view

```
<TextView
        android:id="@+id/textViewDemo"
        android:textColor= "#F00"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Activity lifecycle demo" />
```

User interface

ID-Attribute

Create a new Id with “@+id/NameOfTheId”

```
<TextView
        android:id="@+id/textViewDemo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
```

Use an existing Id with “@id/NameOfTheId”

```
<TextView
        android:layout_above="@id/theOtherTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
```

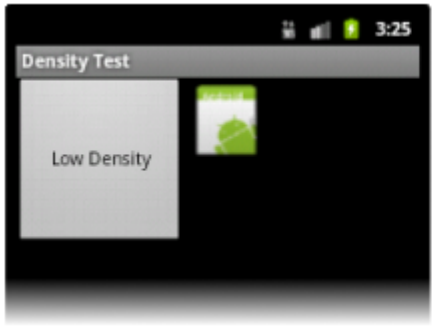
Layout Parameter

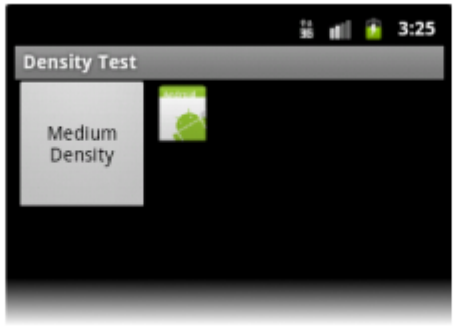
- Define how views are measured
- Depend on the ViewGroup the view does belong to
- layout_width / layout_height are mandatory to every view
- Multiple measurement units available

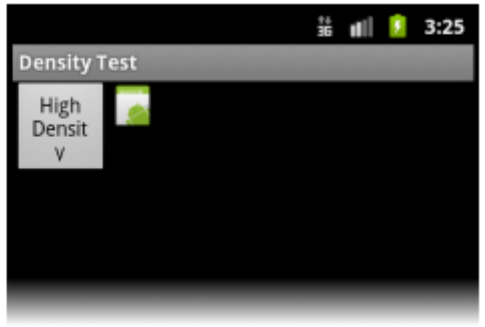
Layout Parameter

Measurement units

Shortcut	Name	Description
px	Pixels	Real pixel on screen
in	Inches	Size in inches
mm	Millimeters	Size in millimeters
pt	Points	1/72 in
dp/dip	Density – independent pixel	Abstract unit based on the current screen density
sp	Scale – independent pixel	Same as dp but scaling with font preference





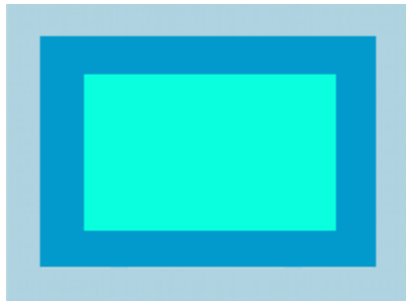


View Groups

Bild Quelle: http://www.kunocreative.com/blog/bid/86368/Top-Tips-and-Tools-to-Measure-Social-Media-ROI

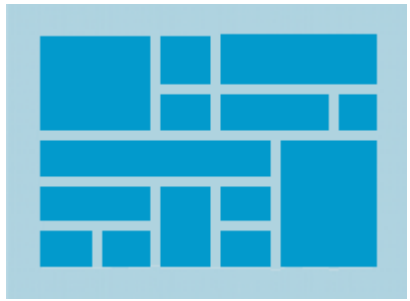
View Groups

FrameLayout



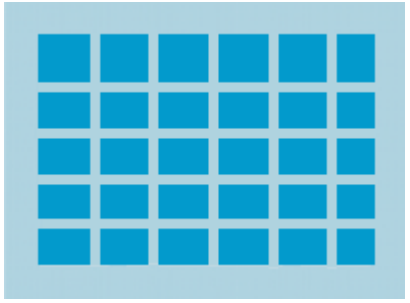
RelativeLayout

GridLayout



LinearLayout

TableLayout



View Groups

Linear Layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

View Groups

Relative Layout

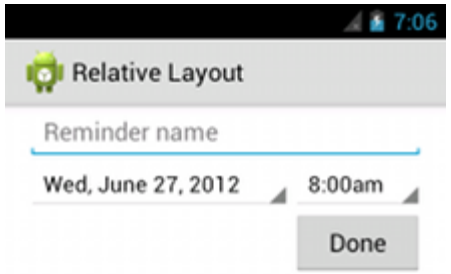
- Align views relative to each other
  - android:layout_toLeftOf="@id/otherView"
  - android:layout_below="@id/otherView"
- Align views relative to parent
  - android:layout_alignParentLeft="true"
  - android:layout_alignParentBottom="true"

- Alignment parameter to override widht / height

View Groups

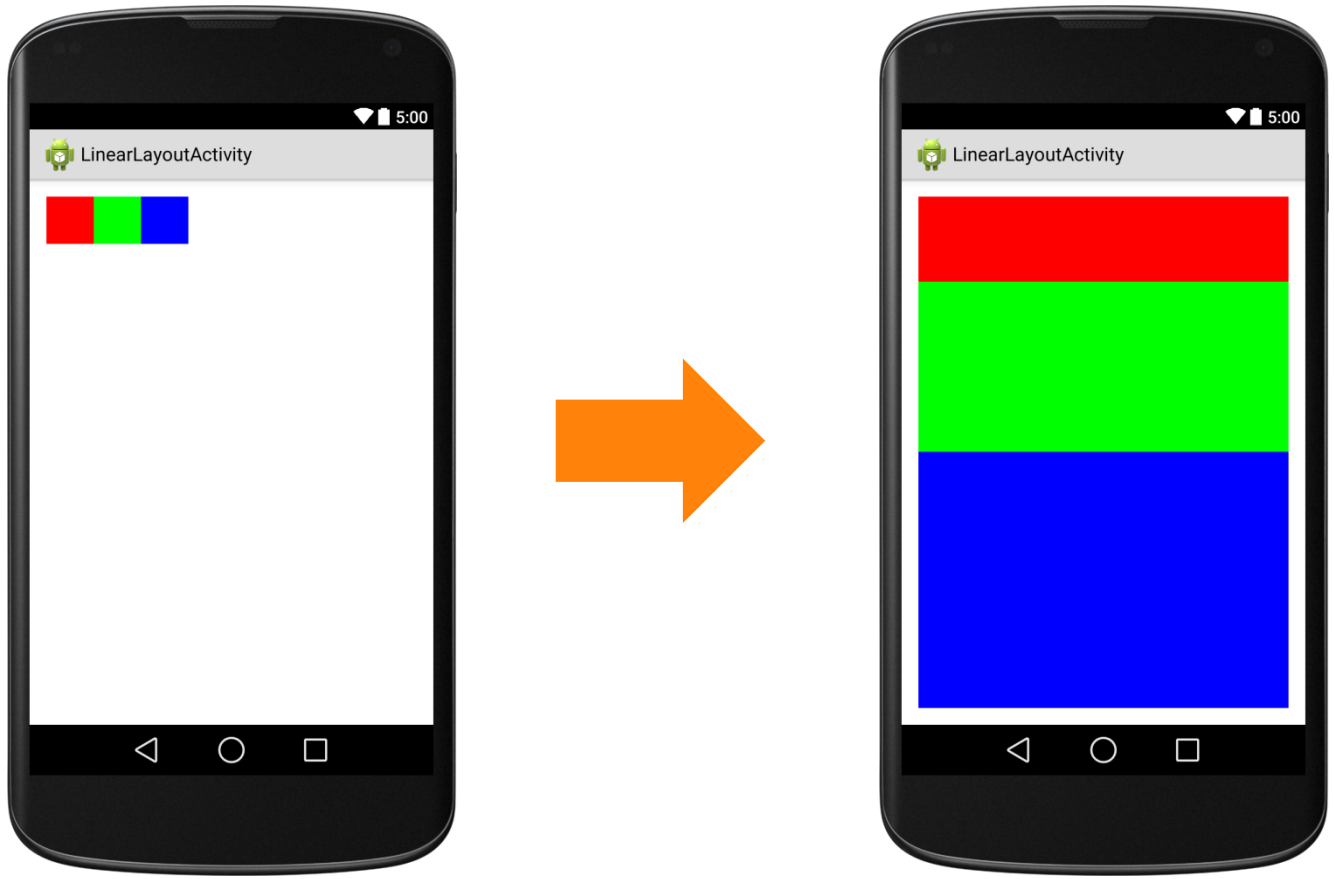
Relative Layout

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp" >
    <EditText
        android:id="@+id/name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/reminder" />
    <Spinner
        android:id="@+id/dates"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentLeft="true"
        android:layout_toLeftOf="@+id/times" />
    <Spinner
        android:id="@id/times"
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentRight="true" />
    <Button
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/times"
        android:layout_alignParentRight="true"
        android:text="@string/done" />
</RelativeLayout>
```



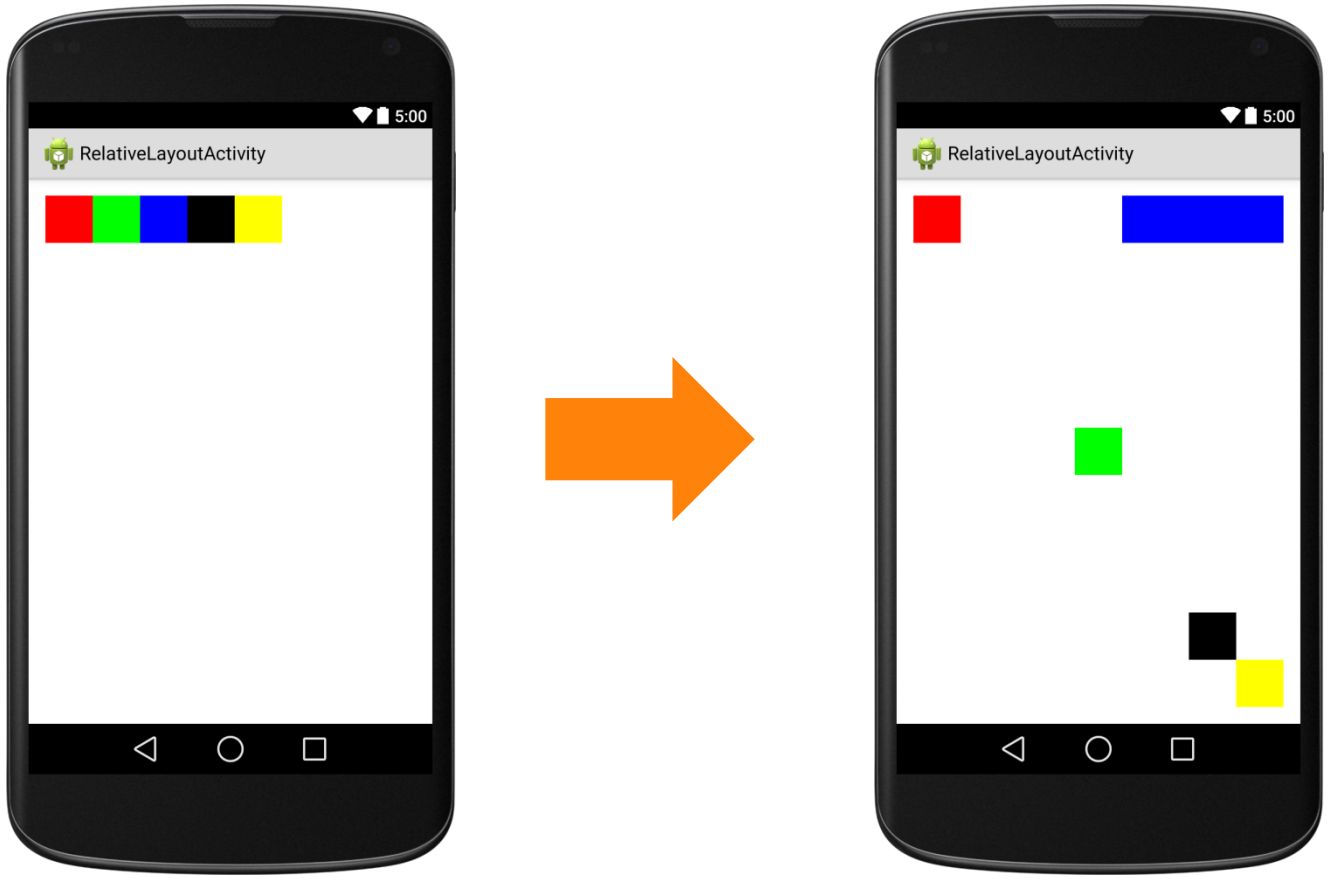
Exercise 4

Exercise04_Layouts → LinearLayout



Exercise 4

Exercise04_Layouts → Relative Layout



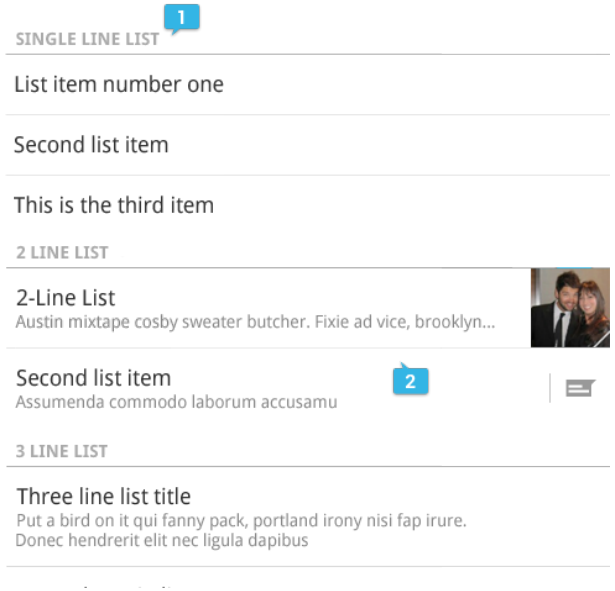
AdapterView

AdapterView

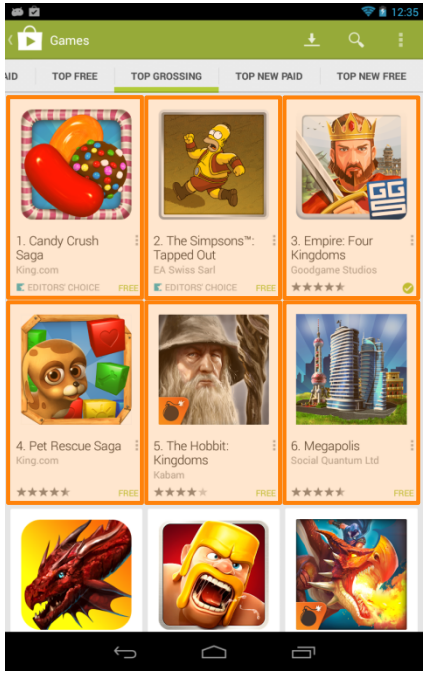
Types

- Display data in a List/Grid
- Dynamic views

ListView

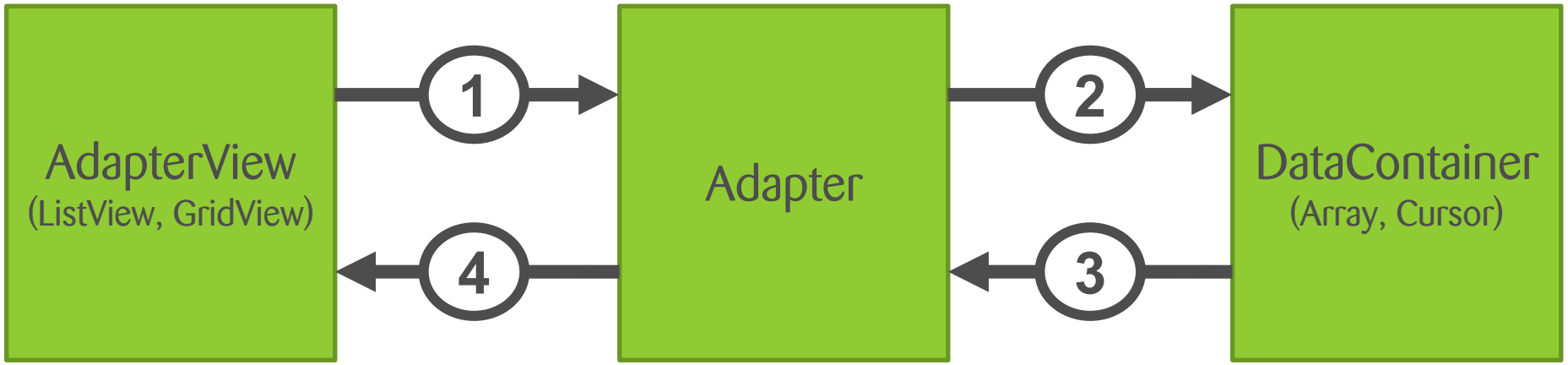


GridView



Adapter View

Connect data to AdapterView



1. AdapterView requests Views to be shown
2. Adapter requests data to be shown
3. DataContainer delivers data entries
4. Adapter creates views for every data entry

Adapter View

Connect data to AdapterView

- Implement interface Adapter
  - ArrayAdapter, BaseAdapter, SimpleAdapter, SpinnerAdapter, …
- Methods to implement
  - getCount()
  - getItem(int position)
  - getItemId(int position)
  - getView(int position, View convertView, ViewGroup parent)
  - getViewType(int position)
  - getViewTypeCount()

Adapter View

Signature of getView(…)

```
public abstract View getView (int position, View convertView, ViewGroup parent)
```

position: Position of the item within the adapters data-set
convertView: Cached view to be reused
parent: Reference to the parent (AdapterView)

- Create view when “convertView” is null
- Set data to the given view
- When multiple types of views
- → Make sure getViewType() / getViewTypeCount() is implemented

Adapter View

Create Views in getView(…)

How to create Views?

- Use a LayoutInflater (System-Service)
- Use inflate(int layout, ViewGroup root, bool attachToRoot)

```
String name = Context.LAYOUT_INFLATER_SERVICE;
LayoutInflater infl = (LayoutInflater) context.getSystemService(name);
View view  = infl.inflate(R.layout.spaced_list_item, parent, false);
```

Exercise: Exercise05_AdapterView

Aufgaben:

1. Sämtliche Klassen analysieren und Zusammenhang zwischen MainActivity (main_activity.xml), PersonAdapter und der ListView verstehen
2. Bug beheben, dass der Name und die Adresse nicht angezeigt werden

Verwendede API Klassen:

- Activity
  - Wird benötigt um einen einzelnen Screen anzuzeigen
- ListView
  - Eine AdapterView welche in main_activitx.xml definiert wurde und sämtliche Personen aus “PersonRepository” anzeigen soll
- BaseAdapter
  - Basis implementation des Adapter-Interface
- LayoutInflater
  - Wird benötigt um in der getView()-Methode des Adapters die benötigten Views zu erstellen