Pázmány Péter Catholic University
Faculty of Information Technology and Bionics

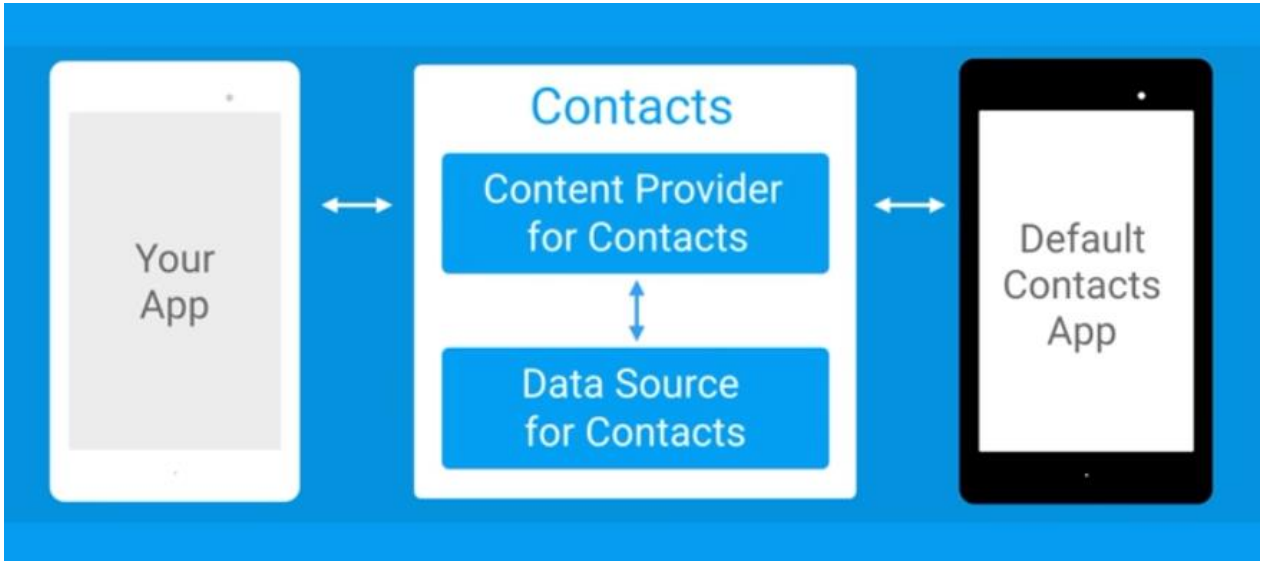
Android Development

Content Providers, Further details on UI

Content Providers

What are content providers

- Manage common data on the phone
  - Read the data
  - Edit the data
  - Add to the data
  - Delete from the data



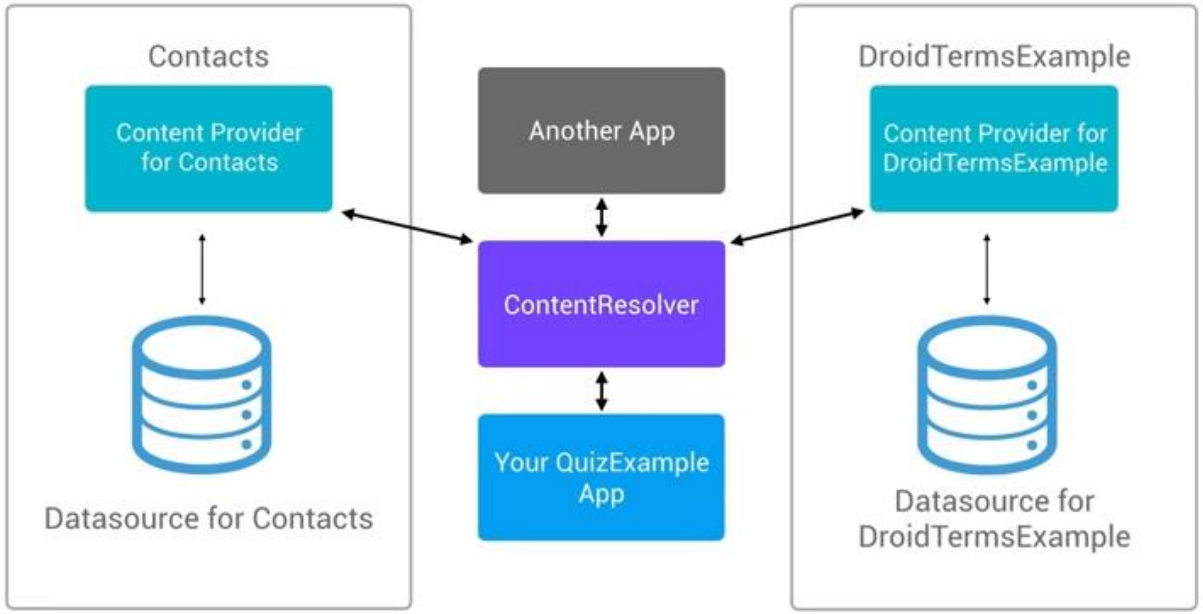
Why to use them?

- Is this just one more class to implement?
  - No!
- What can be the advantage of a „middle man”?
  - Easily change the underlying data source
  - Other built-in Android classes can use them (ex. Loaders)
  - Allow many apps to access, use and modify a single data source securely.
- Security
  - Permission needs to be added to manifest!

```
<uses-permission android:name="com.example.mycontentprovider.TERMS_READ" />
```

Content Resolver

- Middle man between all the content providers and the apps
- Manages the parallel requests to the Content Providers
- `ContentResolver resolver = context.getContentResolver();`



Possible data actions

- Read from the data
  - `query()`
- Add a row or rows to the data
  - `insert()`
- Update a row or rows in the data
  - `update()`
- Delete a row or rows from the data
  - `delete()`

- -> `resolver.selectedMethod()`

URI

- Uniform Resource Identifier
  - URL is a type of URI
- Can be
  - Web address (https://hu.wikipedia.org/wiki/URI)
  - File on a device (C:\Users\tornai\Documents\example.txt)
  - File on the web (https://hu.wikipedia.org/wiki/Port%C3%A1l:Informatika#/media/File:Us-nasa-columbia.jpg)
  - Book (urn:isbn:0-395-36341-1)
- URI in case of Content Providers
  - content://com.example.mycontentprovider/terms
    - content: Content Provider Prefix
    - com.example.mycontentprovider: Content Authority
      - Identifies the exact content provider
    - terms: Specific Data
      - Identifies the exact data in the Content Provider

Cursor

- Content Provider calls returns the same Cursor as SQLite queries.
- `Cursor cursor = resolver.query(MyContentProviderContract.CONTENT_URI, null, null, null, null );`
- Cursor first positioned **before** the first row
  - `moveToNext()`
    - Moves the cursor to the next row if possible
    - Returns true or false depending on if the move was successful or not
- Get data from the cursor
  - `getColumnIndex(String name):` returns the column index of the column with name
  - `get<type>(int index)`: methods return the value in from the index column
    - Chose one from the get methods depending on the type of the data in the column
  - `getCount():` number of rows in the data
- `close():` close the cursor when you are done!

Query parameters

- `Cursor cursor = resolver.query(MyContentProviderContract.CONTENT_URI, <projection>, <selection>, <selection args>, <sort order> );`
  - <projection>
    - This selects the columns
  - <selection> and <selection args>
    - Specifies the rows
  - <sort order>
    - Speaks for itself

Query parameters

```
Cursor cursor = resolver.query(DroidTermsExampleContract.CONTENT_URI,
        <projection>, <selection>, <selection args>, <sort order> );
```

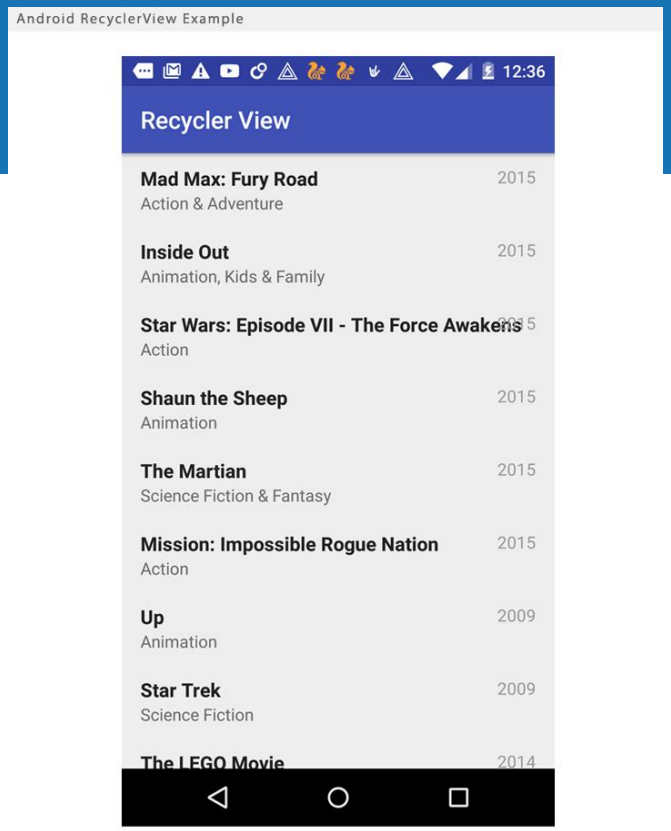
_ID	Word	Definition
1	Adapter	An object that acts as a bridge between an AdapterView and the underlying data for...
2	ListView	A view group that displays a list of scrollable items.
3	RecyclerView	A flexible view for providing a limited window into a large dataset. RecyclerViews require...
4	AsyncTask	A class that makes running asynchronous tasks easier by enabling proper and easy use of...

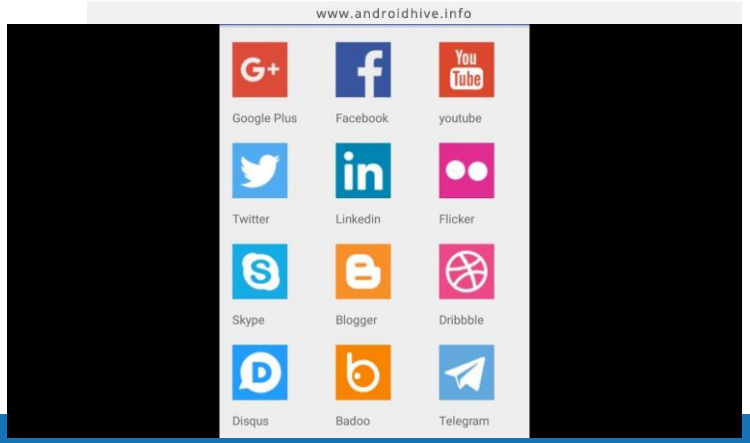
- projection - filters columns
- selection - statement for how to filter rows
- selection arguments - what to filter

Recycler View

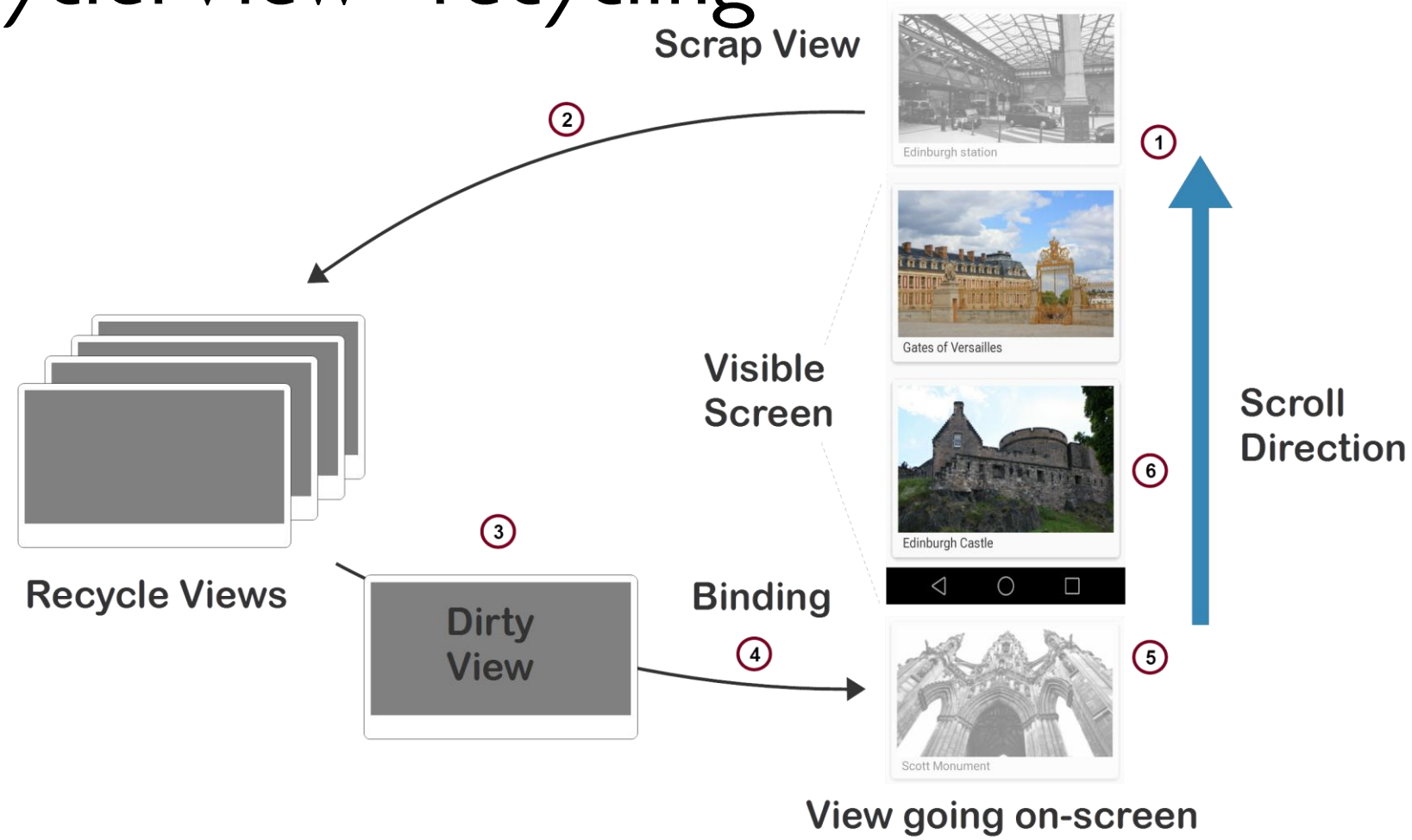
RecyclerView

- Viewgroup
  - Renders a group of views in a similar way
  - For example lists, grids of views
- View inflating is a computationally hard task
  - RecyclerView reuses the views which are not visible
    - faster scrolling
    - Less memory usage





RecyclerView - recycling



Components of RecyclerView

- RecyclerView class
  - This class needs to be placed on the layout and inflated with the other views if it have a LayoutManager and an Adapter defined
- LayoutManager
  - Positions the views inside a RecyclerView
  - Determines when to reuse the item views
- RecyclerView.Adapter
  - Fills the items with data
  - Needs a helper ViewHolder class
    - This stores the Views for one item
  - It inflates the proper ViewHolder
  - It binds the data to the given ViewHolder to display it
- ItemAnimator
  - It can animate the items like swipe or click animations

Usage of RecyclerView

1. Add RecyclerView support library to the gradle build file
2. Define a model class to use as the data source
3. Add a RecyclerView to your activity to display the items
4. Create a custom row layout XML file to visualize the item
5. Create a RecyclerView.Adapter and ViewHolder to render the item
6. Bind the adapter to the data source to populate the RecyclerView

More from Recycler View

- RecyclerView.Adapter types
  - LinearLayoutManager shows items in a vertical or horizontal scrolling list
  - GridLayoutManager shows items in a grid
  - StaggeredGridLayoutManager shows items in a staggered grid
  - Custom LayoutManager can be defined if you extend LayoutManager class
- Configuration
  - Optimization (if the items won't change)
    - recyclerView.setHasFixedSize(true);
- Decoration
  - Add divider or other decoration for items with ItemDecoration
- Animators
  - You can add animations for add, move, delete or more complex animations with ItemAnimators.

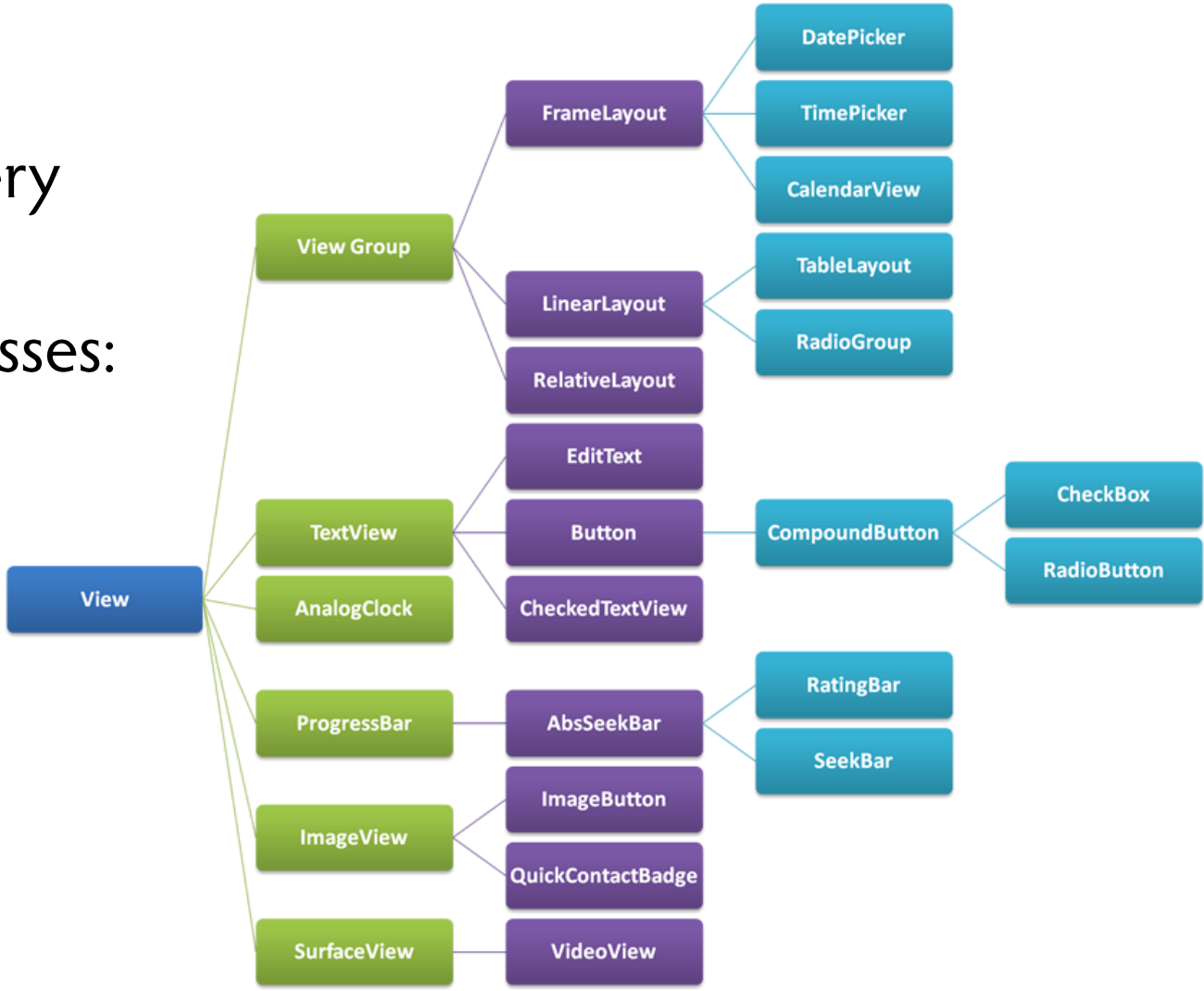
Handling touch events

- Interaction with the items (and every other view in Android) are according to the Observer pattern.
- Multiple ways. For simple click event you can use
  - Use a special ItemDecorator
  - Create an OnclickListener instance for every item
  - Make the ViewHolders implement OnclickListener (maybe the best)
- For any touch event
  - Use ItemTouchListener
- And so on…

Layouts – details

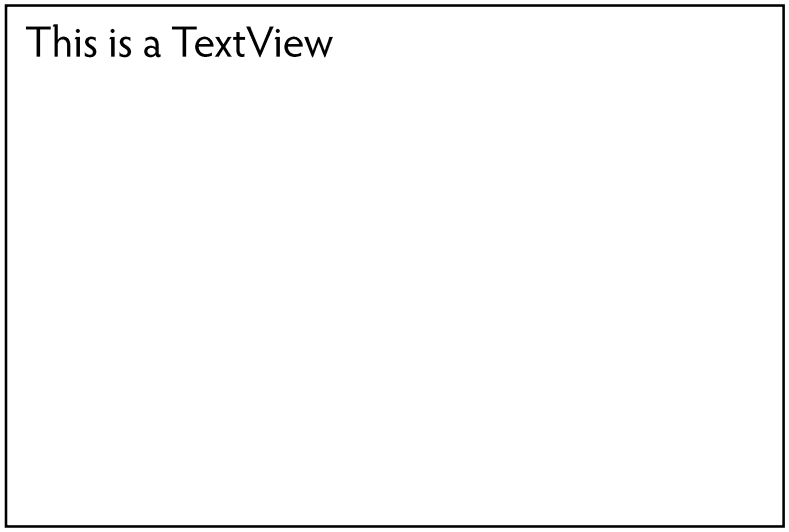
View

- Superclass of every displayed item
- Important subclasses:

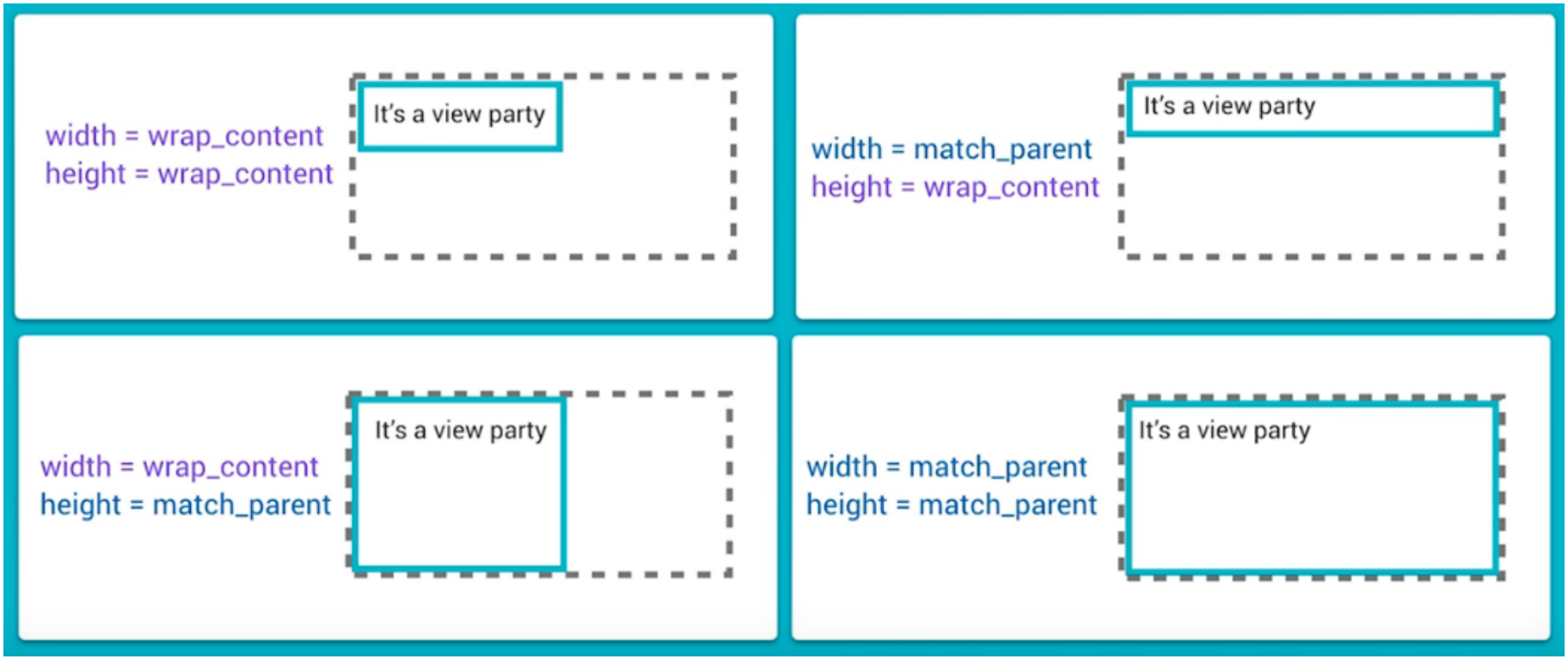


Which one is correct?

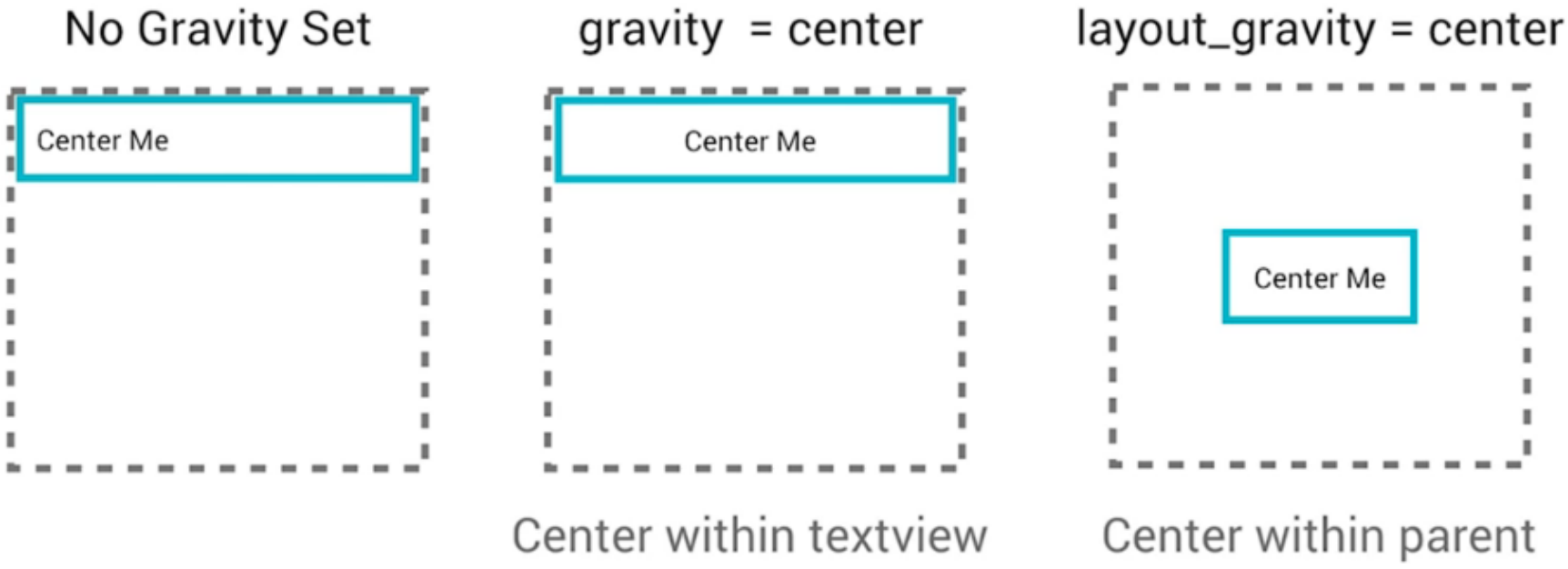
- 1. <TextView
  - layout_width = „wrap_content"
  - layout_height = „wrap_content" />
- 2. <TextView
  - layout_width = „wrap_content"
  - layout_height = „match_parent" />
- 3. <TextView
  - layout_width = „match_parent"
  - layout_height = „wrap_content" />
- 4. <TextView
  - layout_width = „match_parent"
  - layout_height = „match_parent" />



View Width/Height

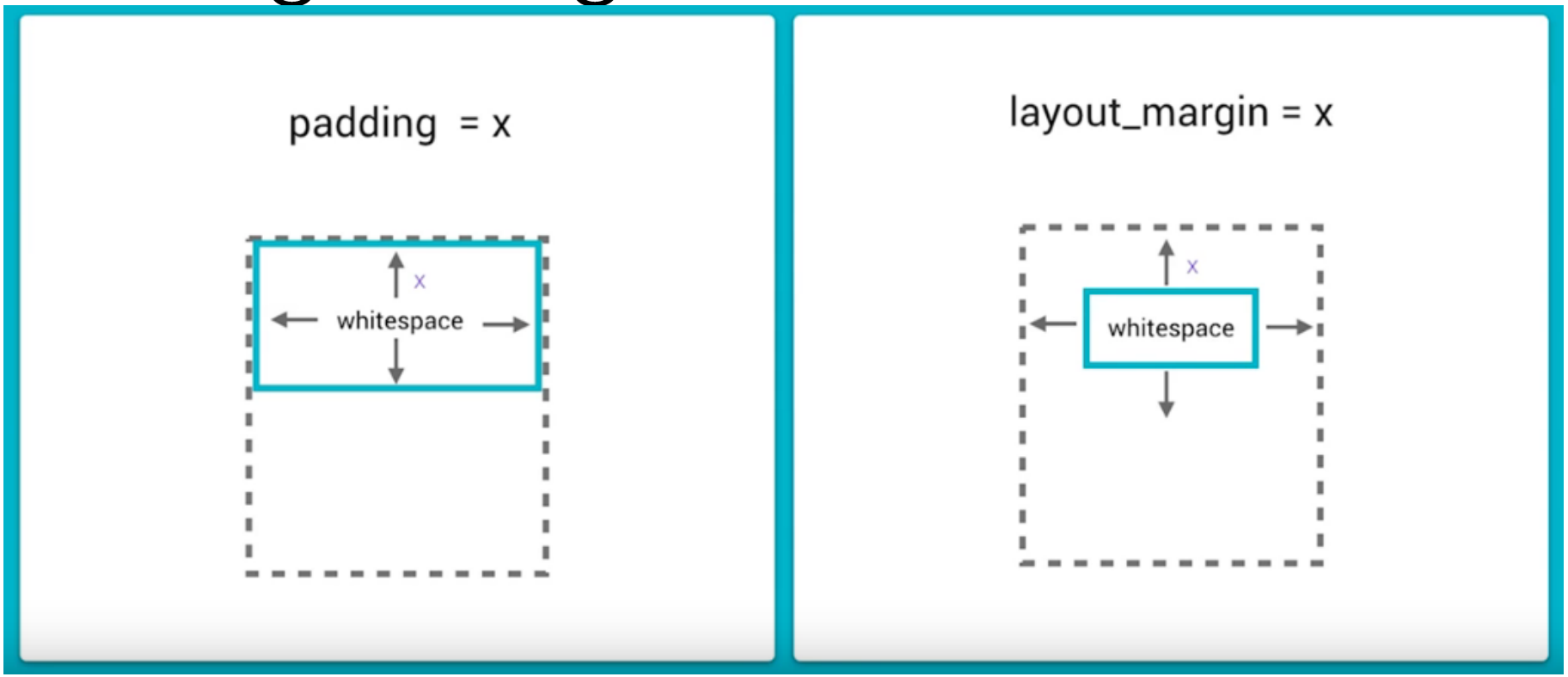


Gravity



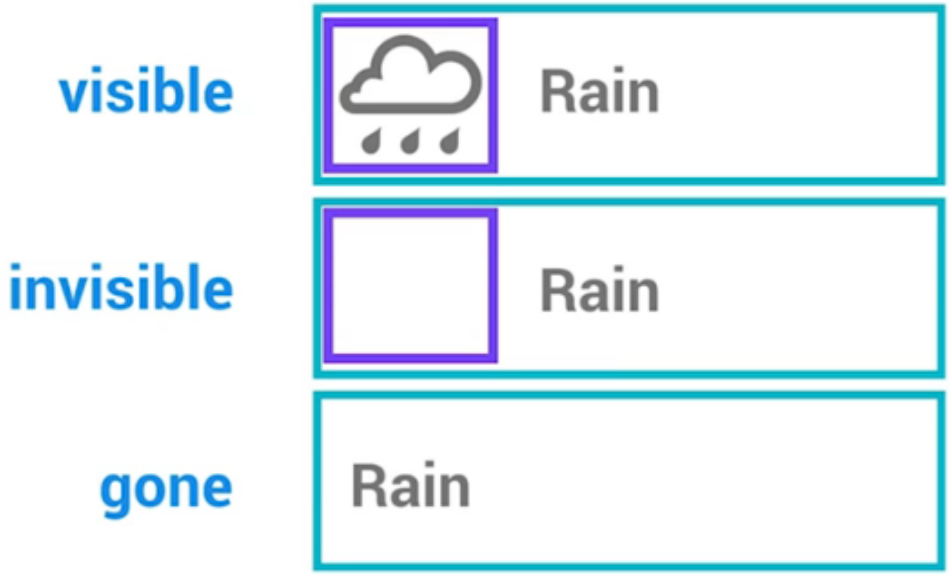
- What is the view width in each cases?

Padding vs Margin



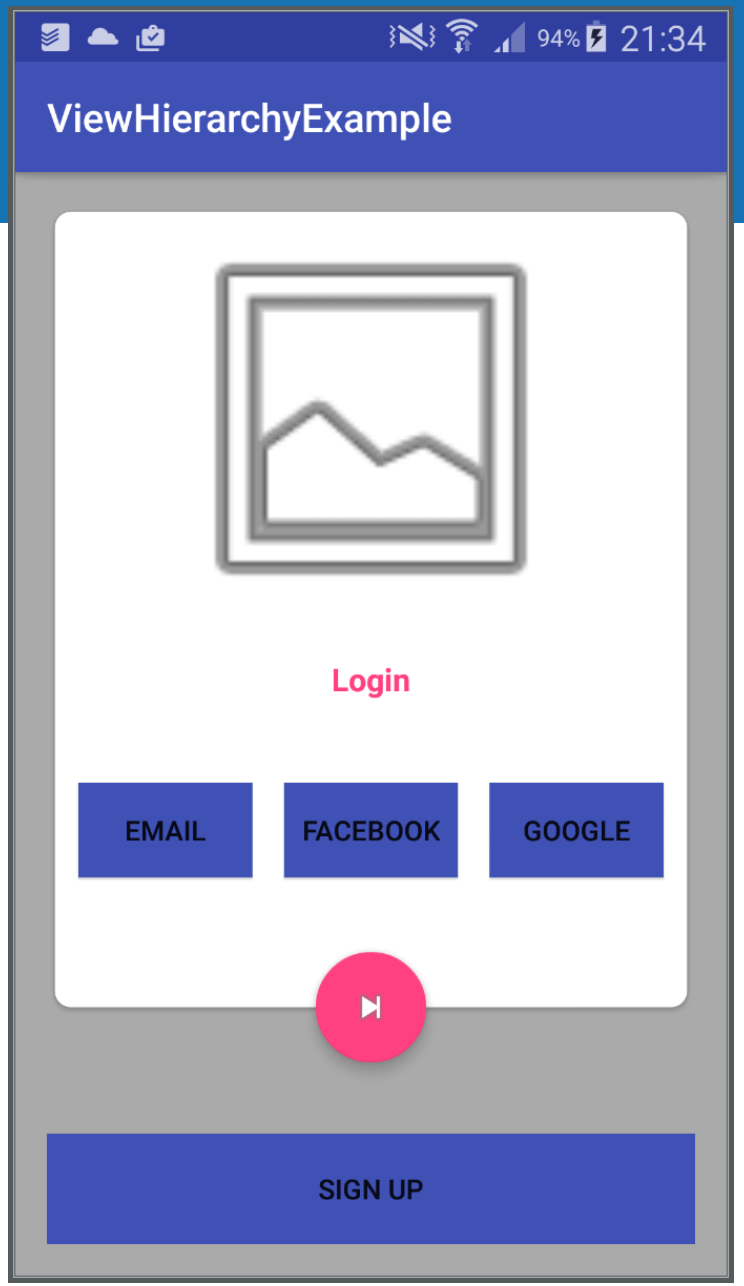
Will the two look different?

View Visibility



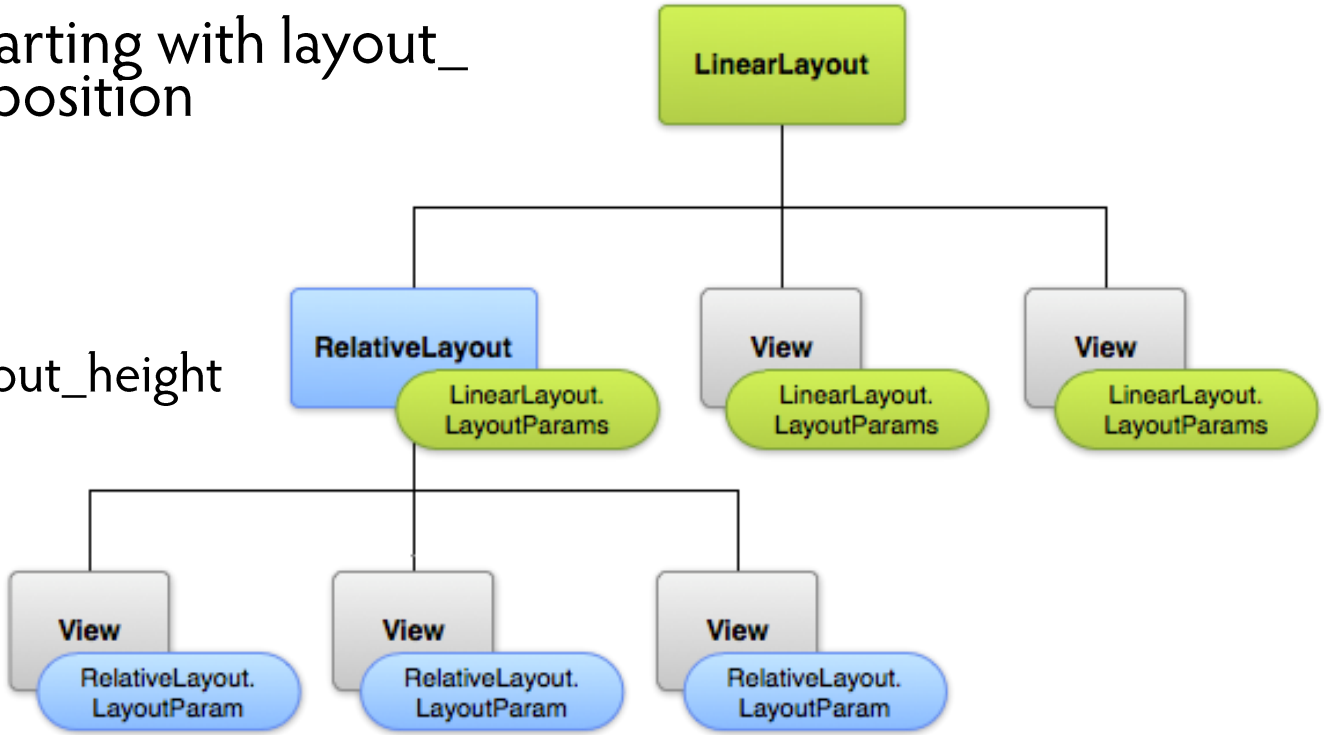
Create this layout

- Use only:
- CoordinatorLayout
- RelativeLayout
- LinearLayout
- CardView
- FloatingActionButton
- Spinner
- Button
- ImageView
- TextView
- FrameLayout
- RadioButton



Nesting Layouts in each other

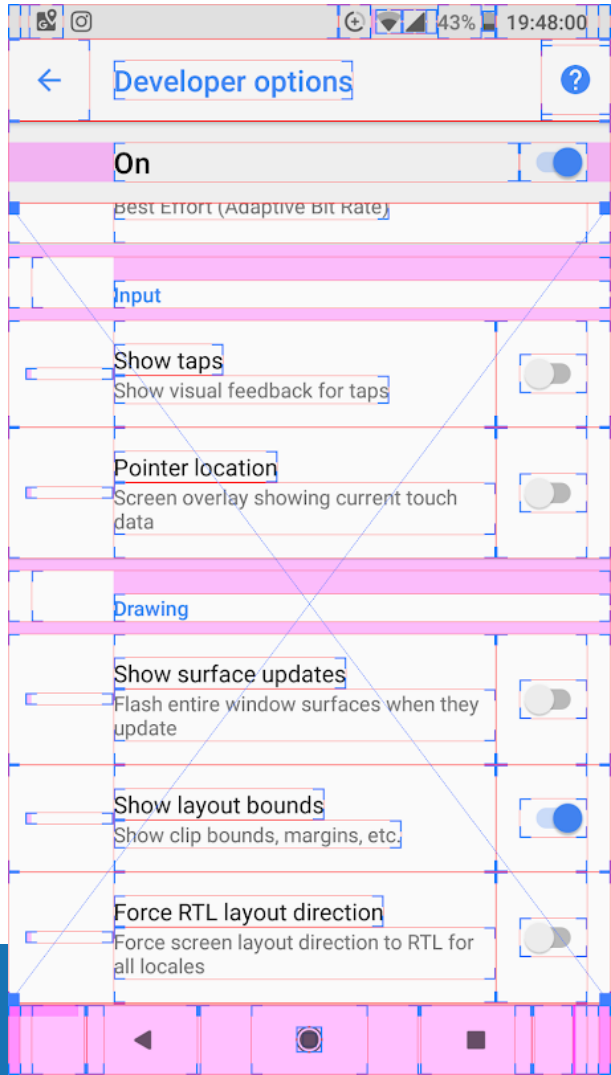
- The parameters starting with layout_ are for the layout position inside the parent



- Every layout:
  - layout_width, layout_height
- LinearLayout:
  - layout_weight
  - layout_gravity
    - But != gravity
- ReltaiveLayout:
  - layout_below
  - layout_alignParentBottom

Layout plan, inspection and profiling

- What makes a layout better?
  - Compatible with different screen sizes and orientations
  - Compatible with different languages
  - Render fast
  - Build upon reusable elements
  - Beautiful
- Tools for creating better layouts
  - Hierarchy Viewer (deprecated) -> dump view hierarchy
  - Layout Inspector
  - Android studio design view
  - Developer options on phone
    - Force RTL layout
    - Show layout boundaries
    - Show screen updates
    - Debug GPU overlay
    - GPU rendering profile
    - Designer Tools



Analyze your view hierarchy

- Try out the following tools!
  - Check your layout boundaries
  - Check View Overlay areas
  - Check nested layout tree size
  - Check layout rendering times (if you have Hierarchy Viewer)
  - Check if FAB is exactly at the center of the bottom of the CardView
    - FAB: Floation Action Button

Data Binding Library

- Built in android support library
- At least gradle plugin 1.5 is required
- Replace your root in the layout xml to a layout tag
- Add the data tag within your layout with the type of the Dataholder class
- Create the Dataholder class
- Bind the Activity in the onCreate
- More info

```
android {
    ....
    dataBinding {
        enabled = true
    }
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<layout
xmlns:android="http://schemas.android.com/apk/res/android
">
   <data>
       <variable name="user" type="com.example.User"/>
   </data>
   <LinearLayout
       android:orientation="vertical"
       android:layout_width="match_parent"
       android:layout_height="match_parent">
            …
            android:text="@{user.firstName}"
            …
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   MainActivityBinding binding =
            DataBindingUtil.setContentView(this,
            R.layout.main_activity);
   User user = new User("Test", "User");
   binding.setUser(user);
}
```

Accessibility

- Making an app International and usable more widely
- Examples
  - **TalkBack** which is a pre-installed screen reader service provided by Google.
  - **Explore by Touch** which is a system feature that works with TalkBack, allowing you to touch your device's screen and hear what's under your finger via spoken feedback. This feature is helpful to users with low vision.
  - **Accessibility** settings that let you modify your device's display and sound options, such as increasing the text size, changing the speed at which text is spoken, and more. and more.
- What can you do?
  - Enable focus-based navigation
  - No audio-only feedback
  - Use sp for text sizes
  - Use the contentDescription attribute for buttons and images without text
  - Use start-end instead of left-right
- More info

More about dp

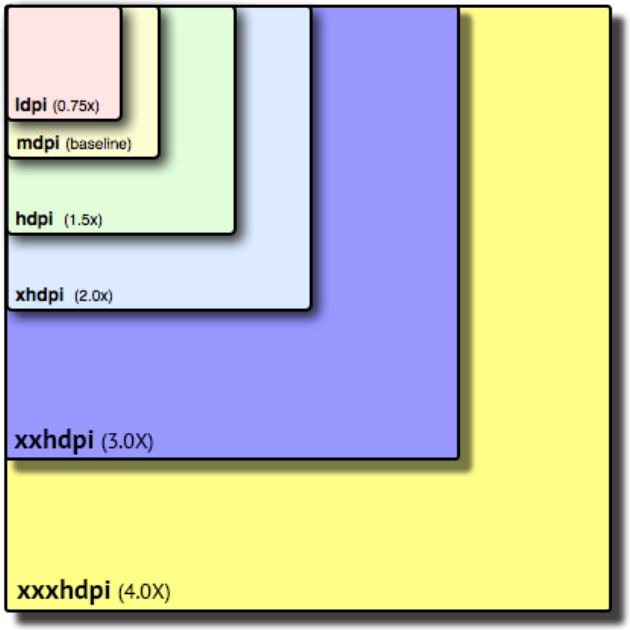
- Resolution: The number of pixels in each dimension that can be displayed.
- Screen size: The size of the screen (usually in inch) can be very different than the dimension!
- Screen Density: The number of pixels in a fixed physical area.
- Dots per inch (dpi): A measure unit of screen density. The number of pixels under one inch.
- Density independent pixels (dp): A physical unit of measurement which represents an abstraction of a pixel for use of an application.

$$\frac{1\ px}{1\ dp} = \frac{160\ dpi}{160\ dpi} \qquad \frac{2\ px}{1\ dp} = \frac{320\ dpi}{160\ dpi} \qquad \frac{?\ px}{12\ dp} = \frac{120\ dpi}{160\ dpi}$$

```
MyProject/
    src/
        MyActivity.java
    res/
        drawable/
            graphic.png
        layout/
            main.xml
            info.xml
        mipmap/
            icon.png
        values/
            strings.xml
```

Resource folder Qualifiers

- Every android resource is stored in the **res** folder
- Providing alternative resources for different configurations is easy with res qualifiers!
- Only the proper ones will be used by the phone
- Possible qualifiers
  - Language
  - Layout direction
  - Screen size
  - Aspect ratio
  - Screen pixel density
  - Etc.
- Full list



Localization

- Create the strings.xml in every supported language
- Use the res qualifiers to separate the different files
- Refer the texts in the strings.xml from your layout and code!
- More info

Vectors vs. Bitmaps

- Because of the different dpi-s of the devices, we need to specify each image in at least 3 sizes (mdpi, hdpi, xxhdpi) but more is better.
  - The res qualifiers make the organization easy, but they still consume space on the device!
- It is possible to define icons & drawings in (scalable vector graphics) **svg** format, which stores only vectors instead of images.
  - They consume more CPU power to draw, but they can lower the APK size dramatically!
  - In the AS you can generate and import them easily
  - Easier to modify them also (change color / size)

State Lists

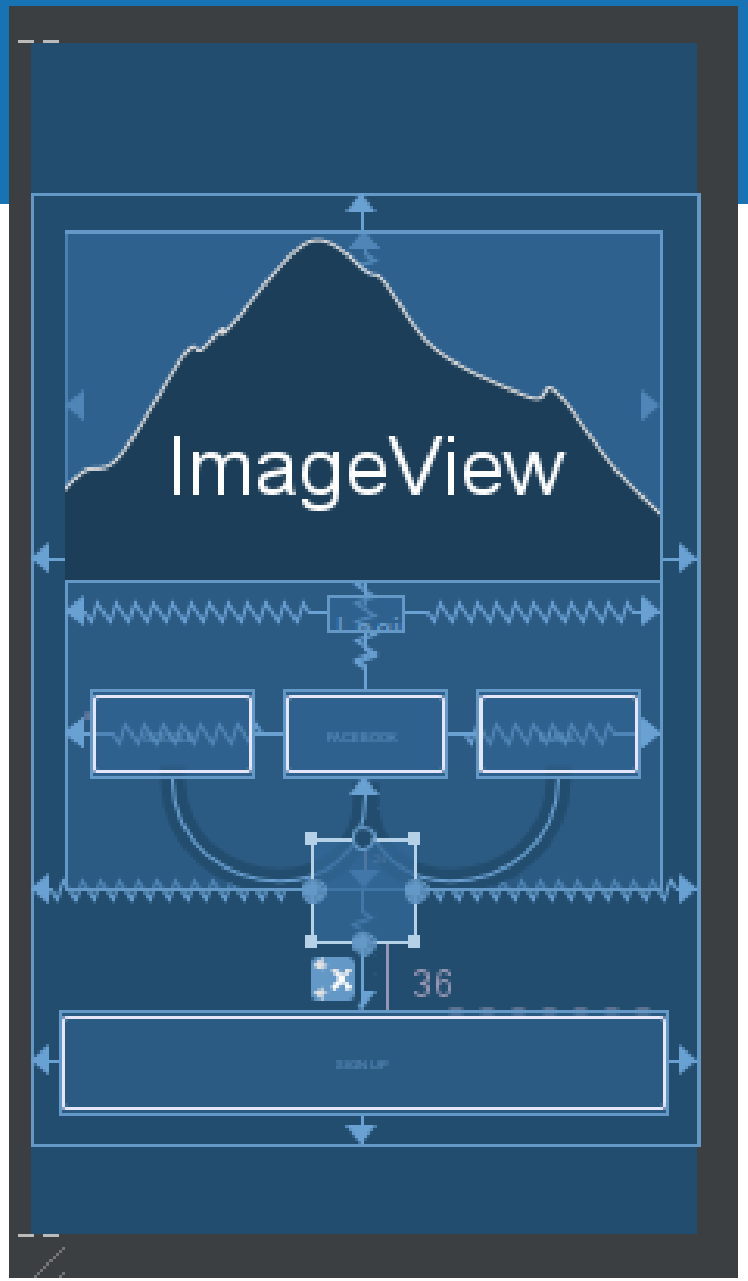
- Possibility to react view state changes without any java code!
- You only need to define the states and the corresponding images/layouts in xml
- Good for
  - Touch-feedback
  - Focus / selection feedback
  - Enabled / disabled view change
  - Checked / unchecked
- Button.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true"
          android:drawable="@drawable/button_pressed" /> <!-- pressed -->
    <item android:state_focused="true"
          android:drawable="@drawable/button_focused" /> <!-- focused -->
    <item android:state_hovered="true"
          android:drawable="@drawable/button_focused" /> <!-- hovered -->
    <item android:drawable="@drawable/button_normal" /> <!-- default -->
</selector>
```

```
<Button
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:background="@drawable/button" />
```

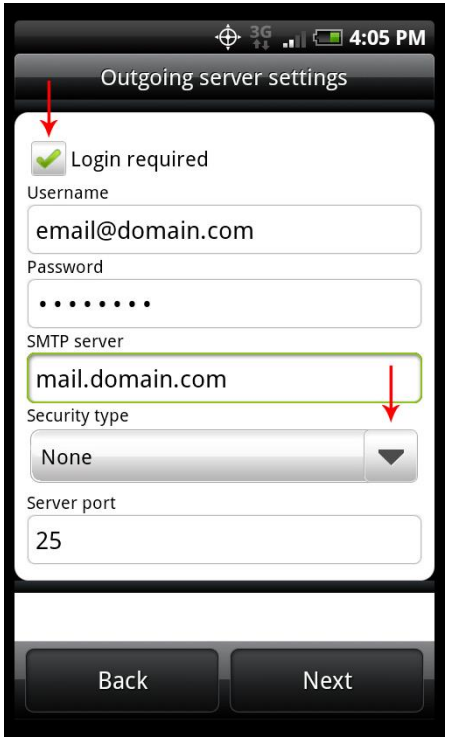
ConstraintLayout

- For avoiding the nested Views Coordinator layout was introduced with Android Studio 2.0
- It is similar to RelativeLayout but have more complex relations between views and it have a graphical interface!
- Create the same layout with the use of ConstraintLayout and try out the tools we used before on this layout also!

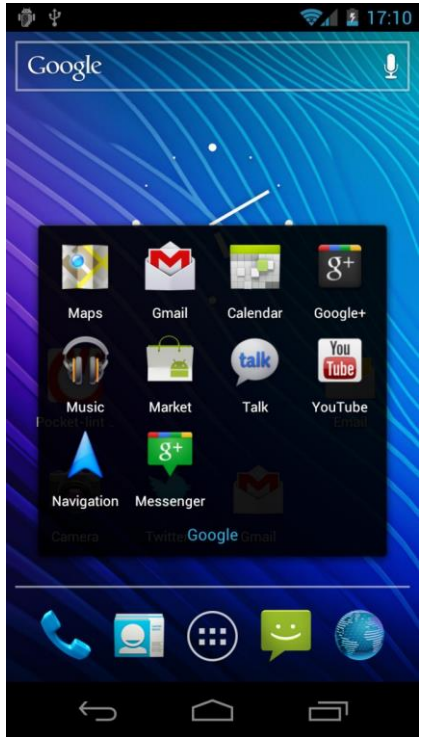


Material design

Old Layouts


4:05 PM
Outgoing server settings
Login required
Username
email@domain.com
Password
SMTP server
mail.domain.com
Security type
None
Server port
25
Back
Next


Google
Posted 4 hours ago
CLASSIC
HTC EVO 4G review
Messaging
Market
Phone
Contacts
Browser
Maps


17:10
Google
Maps
Gmail
Calendar
Google+
Music
Market
Talk
YouTube
Navigation
Messenger

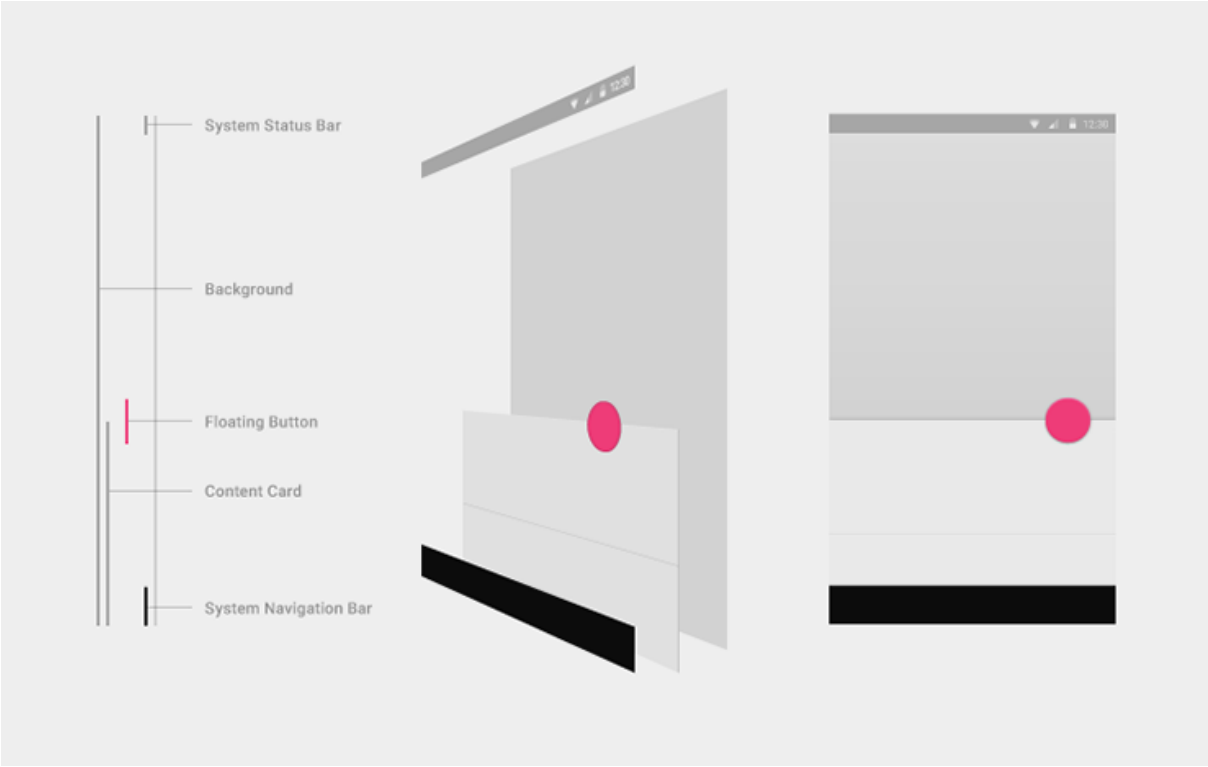

8:43
0
MS
M+
M-
MR
MC
%
√
x²
xⁿ
AC
7
8
9
DEL
C
4
5
6
×
÷
1
2
3
+
-
0
.
±
ANS
=

Material Design

- Make a unified look for the android phone applications
- Why is it important from the perspective of an application?
  - Because the users judge the quality of an app in the first 30 sec!
  - If the usage of the app is more straightforward the users will more probably like it

Material Design Principles

- 3D word and surfaces from digital paper!
- Playing with light and shadow allows us the visualize the third dimension
- More about the dos and don'ts



Useful rules

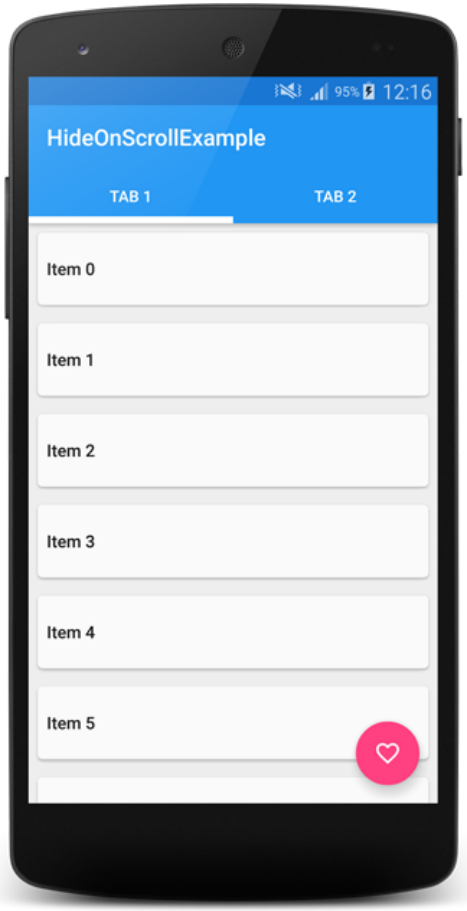
- Touch & interact with the objects itself instead of buttons and menus
- Images instead of words and sentences
- Allow users to customize the app and make it theirs
- Never ask information twice (use content providers)
- Touch targets 40dp at least!
- Shortcuts for complex tasks
- Make the app responsive to touches (Use touch selectors for example)

Material Design Patterns

- Toolbar
  - ?actionBarSize
- App bar
  - Toolbar on the top of the screen
- Tabs
  - Top
- Navigation Drawer
  - Associated with the hamburger icon
- Scrolling & Paging
- List to detail

Elements

- Accent color
  - Use as few color as possible, but have an accent color which differs from your main color palette to direct the users attention
- Floating Action Button (FAB)
  - Floats on the top with accent color
  - Easily accessible
  - The most important action on the screen
- CardView
  - A commonly used surface with rounded corners



Style and Themes

- Reusable sets of attributes to apply to UI
- A style changes the UI of a single element
- Theme uses the same syntax of a style, but applies a collection of styles to an application
- Theme derives to it's children, but style don't
- You can define and override them in the res/values/styles.xml
- More info
- Where did you met with themes already?
  - In the AndroidManifest.xml

Homework

- Create a Contact application:
  - Add a new activity, which opens the clicked contacts and displays detailed information about them
    - The contact photo in bigger
    - The contact name and display name
    - The contact address
    - The contact number
    - The contact email address
    - Hint: https://developer.android.com/guide/topics/providers/contacts-provider.html
  - Make a menu item in this details activity, with which the user can edit the contact (either inside your app or outside)
  - Pay attention to make database calls only in the background thread!

Maps, Location

Next week