Basics of Mobile Application Development

Android Basics

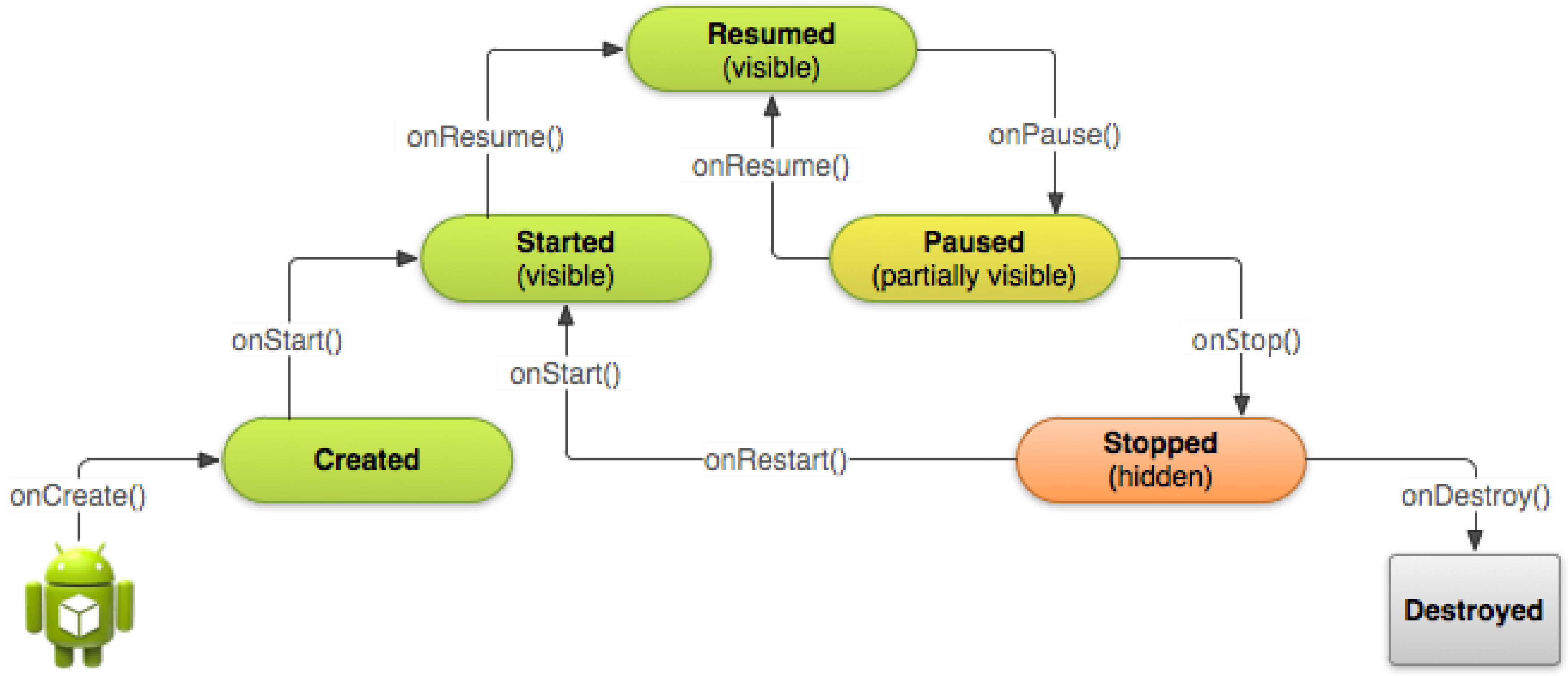
Activity

- There is no main() method!
- „An activity is a single, focused thing that the user can do. Almost all activities interact with the user, so the Activity class takes care of creating a window for you in which you can place your UI with setContentView(View)"
  - So Activities represents a page/window where the user can interact with the application.
  - Only one activity can be active at the time!
  - User can navigate between the Activities with system buttons
    - Back button destroys the current activity and returns to the previous one
    - Home button stops the current activity and returns to the launcher activity
    - Overview button stops(/pauses) current activity and can switch between recent activities

Activity

- Purposes
  - Communicate with the user
  - Handle GUI elements
  - Execute tasks
- An application can have multiple activities
- All activity is derived from `android.app.Activity` class

Activity life cycle



Activity life cycle – methods

- We are informed about the status changes of `Activity` with several different callback functions
  - We have to override these methods, and these methods are called by the system
  - Then we can execute tasks when events occur
- The life cycle functions are:
  - `onCreate:` when `Activity` starts newly (first start, or after disposal)
    - You may set the GUI and variables here
  - `onStart:` when the `Activity` is visible for the user
  - `onResume:` the `Activity` is in focus, now we can start working
  - `onPause:` when `Activity` is partially visible
    - Due to other `Activity`, or `Dialog`, …
    - In case of multi windows system (Android 7.x) when this is the inactive `Activity`
    - You may want to save the necessary information (state)
      - This have to be quick, as it blocks any other `Activity`.
      - If the `Activity` is being destroys this is the only function which execution is guaranteed!

Activity life cycle – methods

- `onStop:` when the `Activity` is invisible
  - It is totally invisible due to another `Activity`, or any other reason
    - Incoming call
    - Screen lock
- `onDestroy:` when `finish()` is called, or memory is needed
  - The `Activity` is destroyed (killed, deleted, …)
  - If the memory is needed instantly then this call may be discarded.
  - Do not save data here, only set the affected variables to `null`

- In all life cycle callback method you have to call the superclass' same method
  - Example: `super.onCreate`
  - The Android system check it
  - Runtime Exception is thrown if you violate this rule

Screen layouts

- You can define the screens two ways
  - Static method
    - Creating .xml files in the res/layout folder
  - Dynamic method
    - In the java source code
    - Creating new instances of View elements
- The layout defines the positions, sizes of elements in the screen
- A layout class is derived from the `View` class!

Attributes of GUI elements

- `layout_width` and `layout_height`
  - Specify the width and height of the view element or layout
    - It is required to specify
    - Runtime exception is thrown if it is missing
  - The actual size is calculated (based on this value and other elements)
- Possible values
  - `wrap_content` – as the content requires
  - `match_parent` – the size of this element is specified by the parent
  - fix size – the unit is dp, which is the devices independent pixel
- `id`: optional (you have to specify if you wish to access it from `Activity`)
- `gravity`: the view is aligned
  - `left`, `right`, `bottom`
  - `center` – vertical and horizontal
    - `horizontal`, `vertical`
    - You can mix: `android:gravity="center|bottom"`

Attributes of GUI elements

- `layout_weight="2"`
  - The „importance" of the element can be set
  - More important element can „push" aside the other elements
    - There are three views but the middle should be larger
- `visibility`:
  - visible – you can see it, visible
  - invisible – cannot be seen, but its size is considered
  - gone – cannot be seen, and no space is occupied
- `padding`
  - Space between the elements
- `background`
  - Could be a color or drawing
- There are attributes which are depending on the actual class of the parent `ViewGroup`
  - For example: the column of a table can be interpreted only in a table

GUI elements

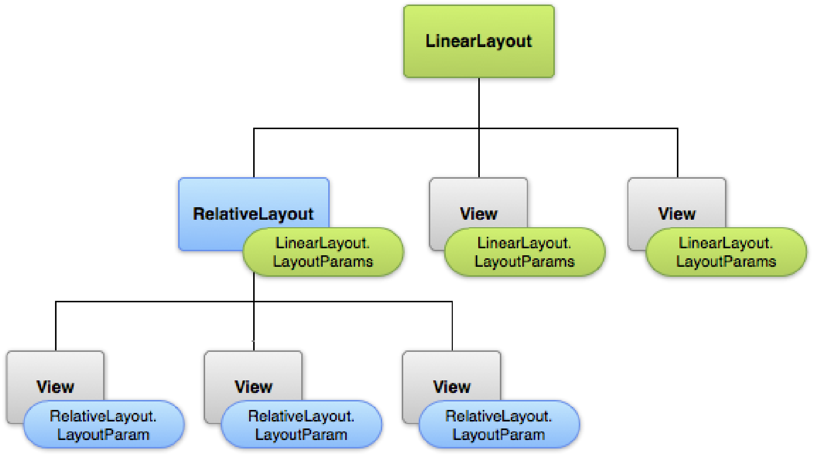
- Layouts
  - Linear Layout
  - Relative Layout
  - Constraint Layout
  - Coordinator Layout
  - Recycler View
  - Frame Layout
  - Web View
- Widgets and other Views
  - Text View
  - Edit Text
  - Auto Complete Text View
  - Button
  - Image View
  - Scroll View
  - View Pager
  - Map View
  - etc

GUI structure

- The GUI is built from Widgets which are `View` and `ViewGroup` elements arranged in a tree structure
  - The [`ViewGroup`]() is extended from the [`View`]() class also
  - The `ViewGroup` is a special `View`, which can have children, so it can contain other elements
- It is possible the define own Views or View groups, but there are a lot of predefined one.
  - If you need to create an own view Extend from the proper class

View hierarchy

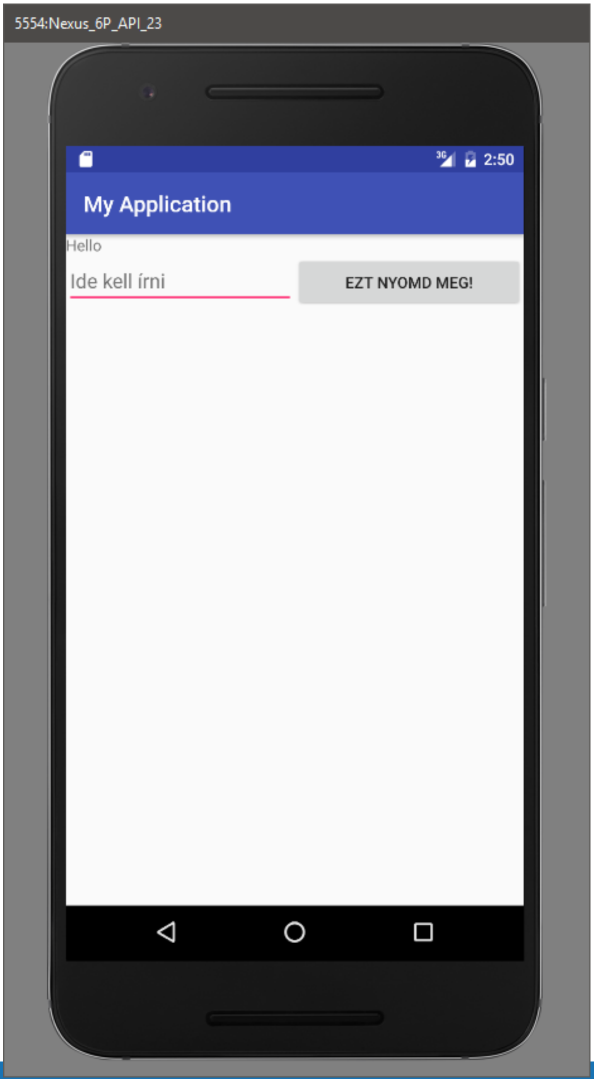
- There is one root element
- Set the root element with the `setContentView` function of the Activity class.
  - In the onCreate() function
- `Every ViewGroup` responsible for the drawing of it's children
- Views are drawn on the top of root.
- We can add child to a ViewGroup dynamically with the `addView(View)` `function`



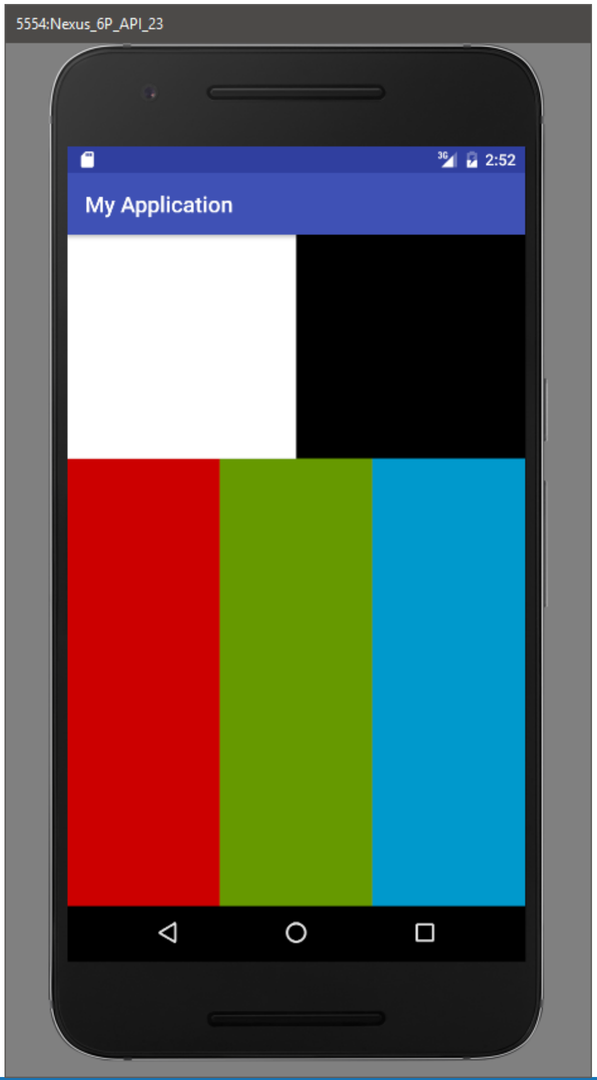
Inflation

- The hierarchy can be derived in xml files as well
  - In that case the parameter of the `setContentView` is not a View, but an int
    - This is an id for the layout file
    - The id and the xml are connected in the `R.java` file
    - The connection is automatically created
  - First, the system creates the view hierarchy based on the layout
  - Then the it calls the `setContentView(View)` function
  - Example:
    - Last week's hello worlds
    - `setContentView(R.layout.activity_main);`

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text= "Hello" />
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >
        <EditText
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:hint="Ide kell írni" />
        <Button
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Ezt nyomd meg!" />
    </LinearLayout>
</LinearLayout>
```



Example

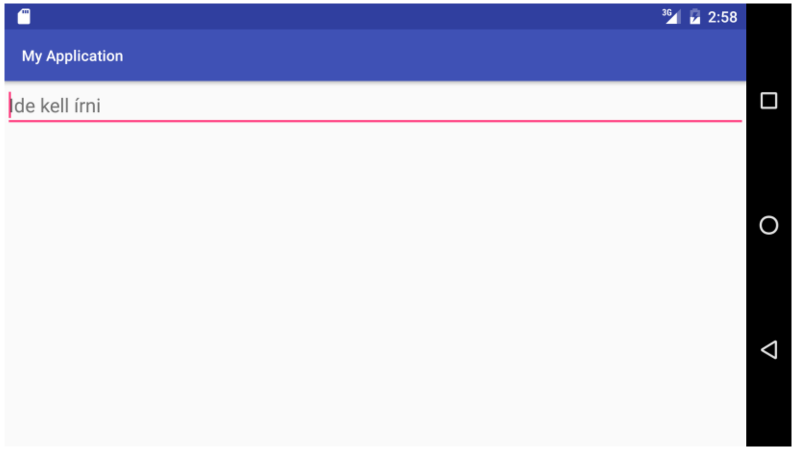


Widgets

- **`TextView`**
  - To display text
  - Main attributes
    - `text` – text given
    - `textColor` – color of text
    - `textSize` – size of the text
    - `typeface` – font of the text

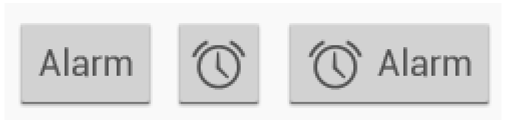
Widgets

- **`EditText`**
  - Derived from `TextView`
  - To input text
  - The keyboard is shown automatically when this view gains the focus
  - Important attributes
    - `inputType` – text, number, email address, etc.
    - `hint` – hint is shown before any text is added



Widgets

- **`Button`**
  - Derived from `TextView`
  - Differs only the default background
  - Represents a button which can be pressed

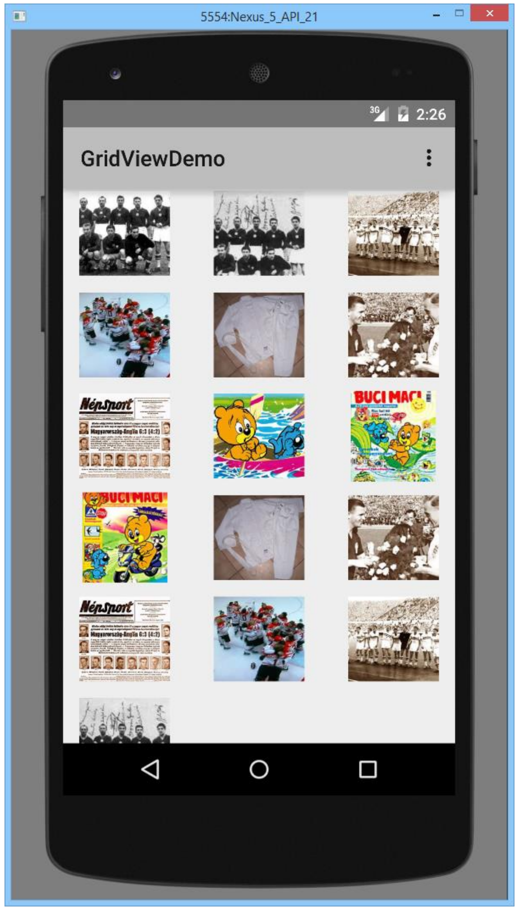


Widgets

- **`ImageView`**
  - To display a picture
    - Set the `src`, not the background
  - Important attributes
    - `scaleType`: how the image is scaled if the aspect ration of the `View` differs from the image
  - `src`: the image to be displayed

GridView

- Views in a grid (matrix)
- A list adapter has to be defined
  - Adapter to get the actual View
  - For optimal usage of the resources
  - A View is instantiated if and only if it is about to be displayed or visible
  - In lists and similar Views this technique is used
- Larger content handled automatically



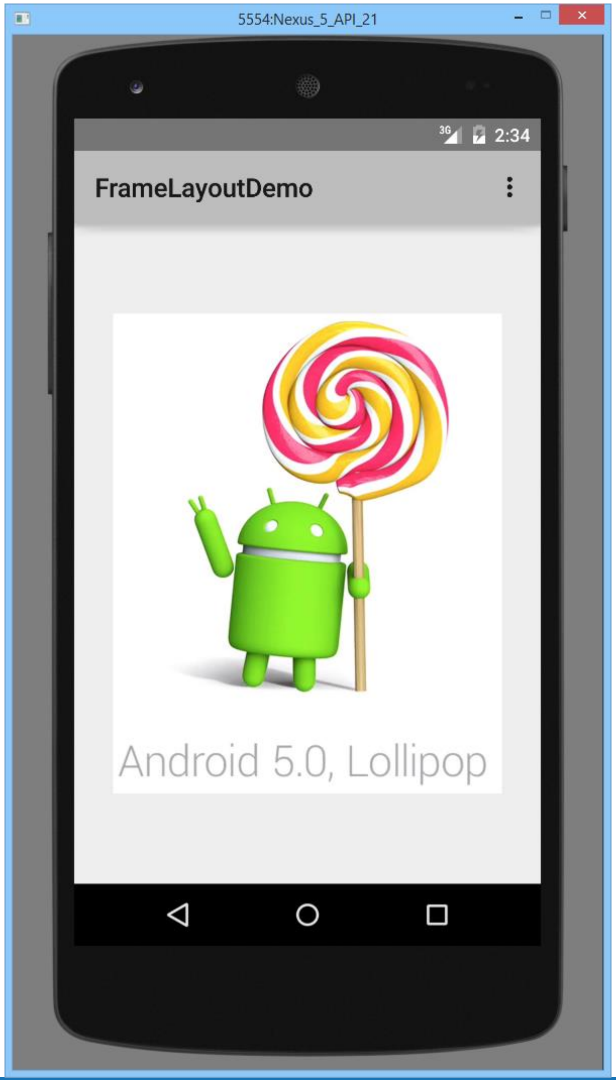
WebView

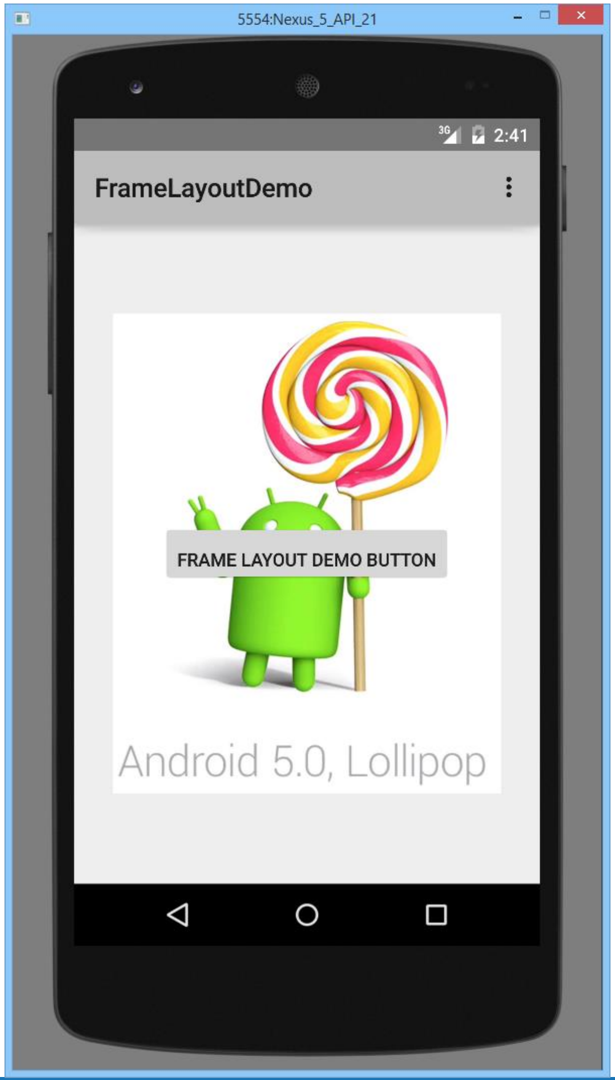
- Built-in web browser
  - To display web content

FrameLayout

- To display a single element
  - When you put more than one element the overlap each other
  - However you can specify
    - Gravity
    - Padding
    - Etc.

Example


5554:Nexus_5_API_21
2:34
FrameLayoutDemo
Android 5.0, Lollipop


5554:Nexus_5_API_21
2:41
FrameLayoutDemo
FRAME LAYOUT DEMO BUTTON
Android 5.0, Lollipop

TableLayout

- Table-like arrangement
- TableRow
  - These are the rows of the tables
  - The columns are created from the elements of the rows (horizontal layout)
- TableLayout
  - Contains TableRow elements, under each other
- TableRow
  - Contains View elements, next to each other
- The number of the columns
  - Is defined by the maximal number of Views of all TableRows
  - This maximal value is used for all rows

TableLayout

- Width of the column
  - The widest View of the column
- `TableRows` are always
  - `layout_width` = `MATCH_PARENT`
  - `layout_height` = arbitrary chosen

TableLayout

- What is the parent of the TableRow?
  - LinearLayout
- Example
  - `android:stretchColumns`: to fill the screen:
    - `android:stretchColumns="0"` , 0. column
    - `android:stretchColumns="1, 2"` , 1. and 2.
    - `android:stretchColumns="*"`, all

Defining a listener

- The most basic method is to implement the Listener by the Activity

```
public class MainActivity extends Activity implements View.OnClickListener {
  @Override
  public void onClick(View v) {
    // runs on Click event
  }
}
```

- Thus the instances passed to the View is the instance of the current `Activity`
  - `this`

Adding a listener

- The Listener can be set in the XML
  - It is not recommended as in case the code is changed you have to modify the XML as well
    - If you forget it you will get runtime exception
- The listener also can be set in the code
  - By calling the set<ListenerName>(listenerInstance) function
- You should used the `Activity.findViewById` method to retrieve the instance of a view by id
  - The parameter is the id of the `View` which is set in the XML (Example: `R.id.myButton`)
  - This method iterates recursively the `View` hierarchy, and returns with the first occurrence
- It returns a `View`
  - Thus the type must be casted explicitly
  - `Button button = (Button) findViewById(R.id.myButton);`
  - Remember `ClassCastException` is thrown if the type mismatches
- Finally: `button.setOnClickListener(this);`

Listener – anonymus class

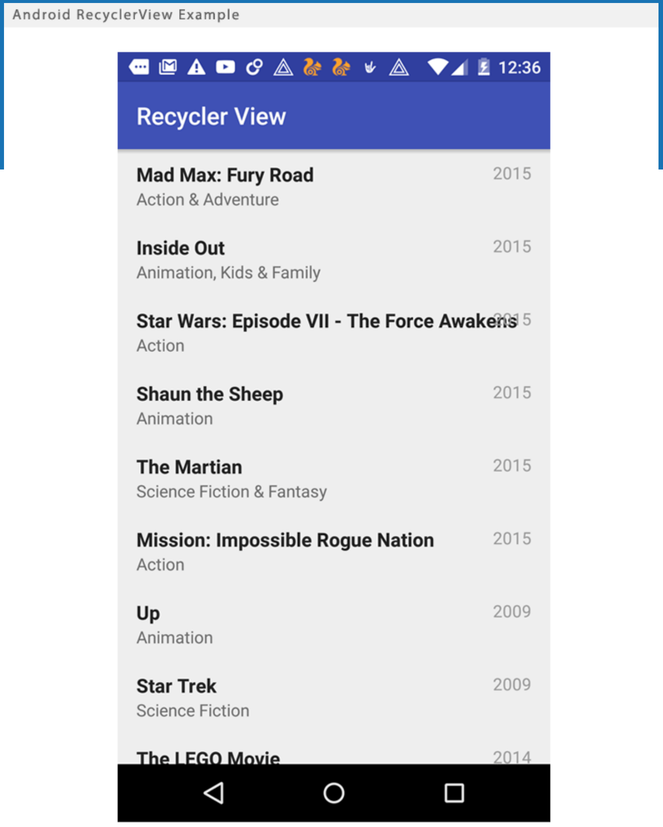
```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  Button button1 = (Button) findViewById(R.id.button1);
  button1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
      Log.i("MainActivity", "button1 pressed");
    }
  });

  final String string = "hello";

  Button button2 = (Button) findViewById(R.id.button2);
  button2.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
      Log.i("MainActivity", string);
    }
  });
}
```

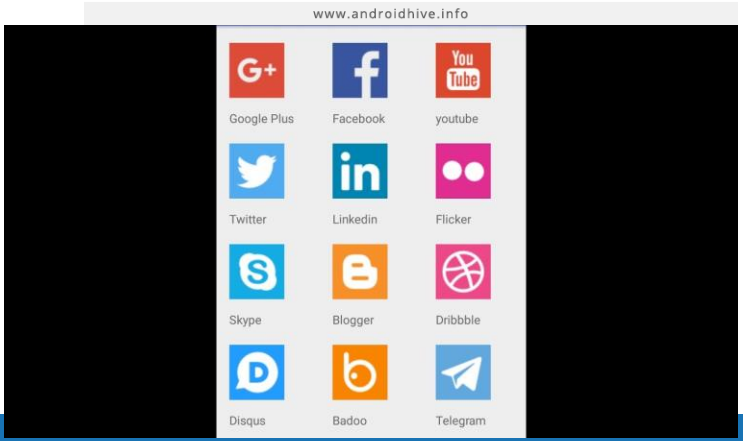
RecyclerView

- 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

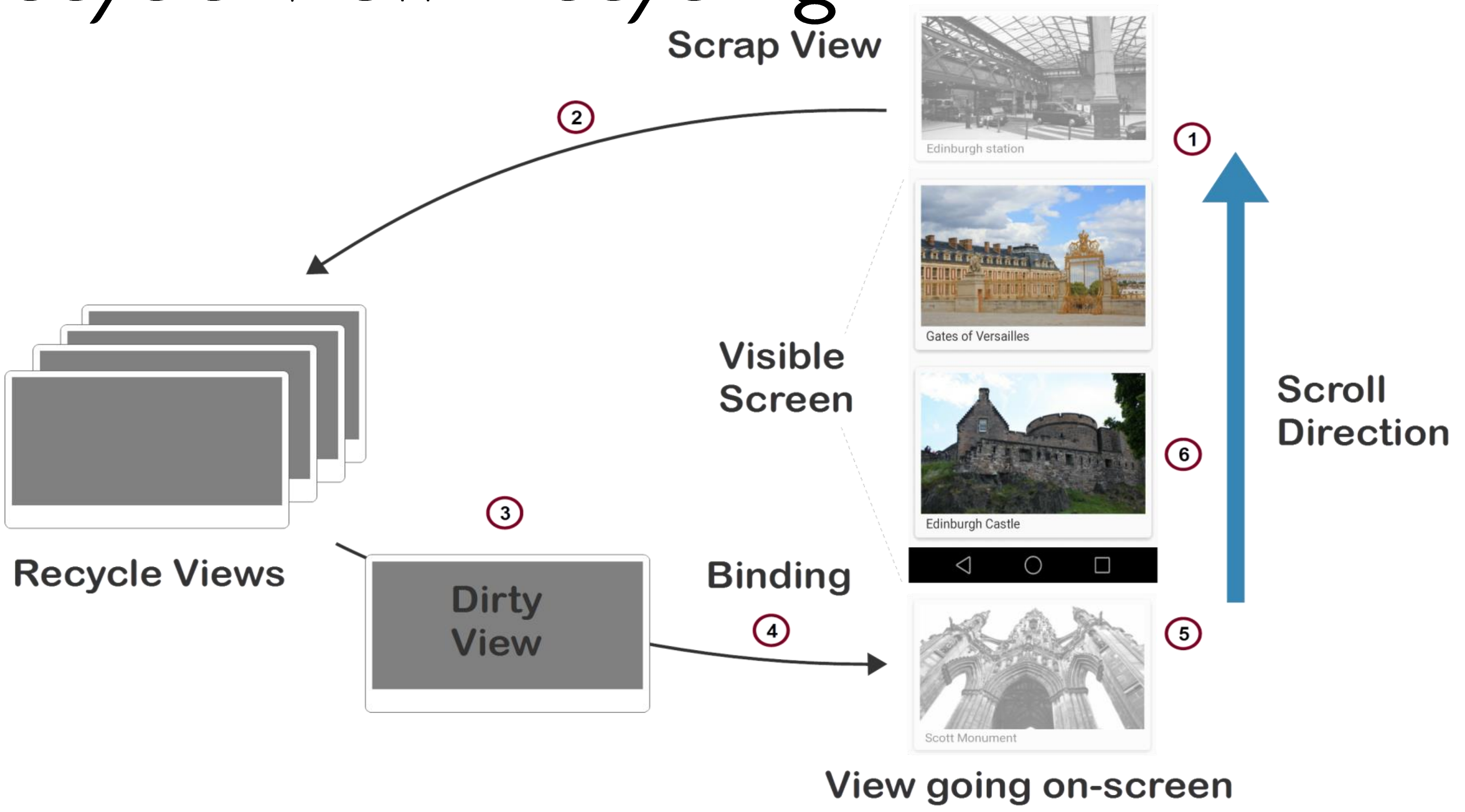

Android RecyclerView Example

Recycler View

Mad Max: Fury Road Action & Adventure	2015
Inside Out Animation, Kids & Family	2015
Star Wars: Episode VII - The Force Awakens Action	2015
Shaun the Sheep Animation	2015
The Martian Science Fiction & Fantasy	2015
Mission: Impossible Rogue Nation Action	2015
Up Animation	2009
Star Trek Science Fiction	2009
The LEGO Movie	2014



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…

User events, interaction

- The UI event handling is like the Observer pattern
- You have to implement an interface, which dedicated method will be called in case of the event occurs
  - In the Observer design patter that was the notify/update function
  - In Android this function name indicates the type of the event
    - `onClick`
    - `onLongClick`
- After the class is ready, you have to instantiate
- And this instance have to be passed to the View
  - The `View` is the subject
  - You have to call the `set<ListenerName>` method
    - Similar to the `registerObserver`

Resources

res/anim - Animations

- Basic animations can be described
  - Sliding picture, animated buttons, …

```
<set
  android:ordering=["together" | "sequentially"]>

    <objectAnimator
        android:propertyName="string"
        android:duration="int"
        android:valueFrom="float | int | color"
        android:valueTo="float | int | color"
        android:startOffset="int"
        android:repeatCount="int"
        android:repeatMode=["repeat" | "reverse"]
        android:valueType=["intType" | "floatType"]/>

    <animator
        android:duration="int"
        android:valueFrom="float | int | color"
        android:valueTo="float | int | color"
        android:startOffset="int"
        android:repeatCount="int"
        android:repeatMode=["repeat" | "reverse"]
        android:valueType=["intType" | "floatType"]/>

    <set>
        ...
    </set>
</set>
```

res/values

- Values
  - To implement application with multi lingual support
  - values/string.xml   values-fr/string.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="string_name" >text_string</string>
</resources>
```

- The strings can be referred from
  - layout.xml-s
  - Activity-s

Styles

- To define custom styles
  - Rounded button
  - Custom text with coloring, format, etc. …
- It can be applied on a single `View` as well as on entire `Activitys`

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style
        name="style_name"
        parent="@[package:]style/style_to_inherit">
        <item
            name="[package:]style_property_name"
            >style_value</item>
    </style>
</resources>
```

  - `<EditText style=“@style/numbers” …/>`

res/layout

- To define GUI layouts

```
<?xml version="1.0" encoding="utf-8"?>
<ViewGroup xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@[+][package:]id/resource_name"
    android:layout_height=["dimension" | "match_parent" | "wrap_content"]
    android:layout_width=["dimension" | "match_parent" | "wrap_content"]
    [ViewGroup-specific attributes] >
    <View
        android:id="@[+][package:]id/resource_name"
        android:layout_height=["dimension" | "match_parent" | "wrap_content"]
        android:layout_width=["dimension" | "match_parent" | "wrap_content"]
        [View-specific attributes] >
        <requestFocus/>
    </View>
    <ViewGroup >
        <View />
    </ViewGroup>
    <include layout="@layout/layout_resource"/>
</ViewGroup>
```

res/color

- Example
  - To define colors for a button
  - Depends on the state (normal state, pressed, released)

  -
    ```
    <?xml version="1.0" encoding="utf-8"?>
    <selector xmlns:android="http://schemas.android.com/apk/res/android" >
        <item
            android:color="hex_color"
            android:state_pressed=["true" | "false"]
            android:state_focused=["true" | "false"]
            android:state_selected=["true" | "false"]
            android:state_checkable=["true" | "false"]
            android:state_checked=["true" | "false"]
            android:state_enabled=["true" | "false"]
            android:state_window_focused=["true" | "false"] />
    </selector>
    ```

res/color

- Similarly different images also can be defined for buttons
  - Depending on actual state
  - If we are not working on colors, the res/drawable should be used
- An XML have to be defined

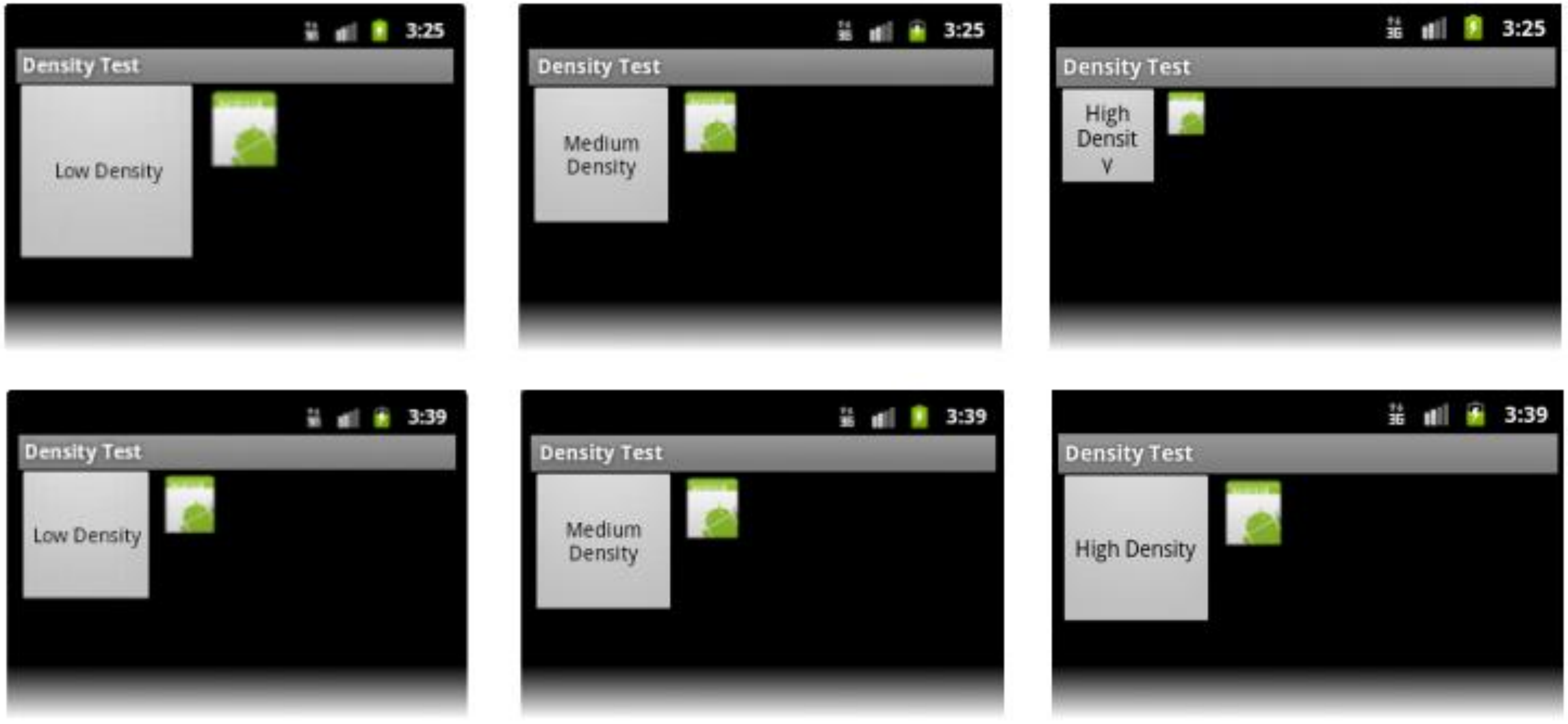
- `android:text_color= "@color/color_name"`

Screen sizes

Definitions

- Screen size: the physical size of the screen
- Orientation
  - Landscape or portrait
- Number of pixels
- Dpi
  - dots per inch
  - Defines the screen density as well
- Dp or Dip
  - Device independent pixel (or density independent pixel)
  - Virtual pixel
    - We can achieve that the physical size of the objects are the same on different devices
      - 1 dp ~ 0,16mm ± 0,02mm on screen
  - You have to used it in GUI XML
- Resolution
  - Number of pixels on the screen

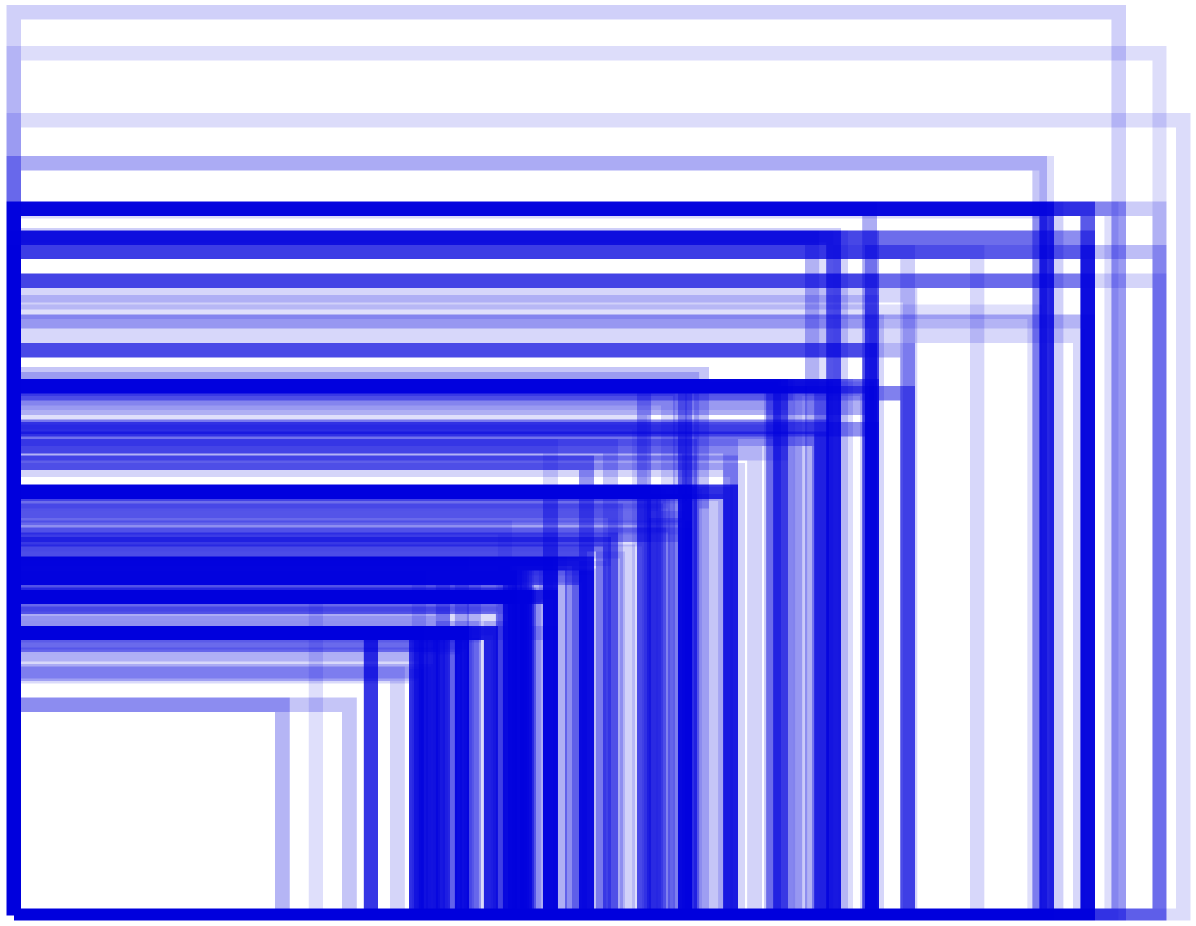
Comparison


3:25
Density Test
Low Density
3:25
Density Test
Medium Density
3:25
Density Test
High Densit y
3:39
Density Test
Low Density
3:39
Density Test
Medium Density
3:39
Density Test
High Density

Some example

	Low density (120)	Medium density (160)	High density (240)	Extra high density (320)
Small	240x320		480x640	
Normal	240x400 240x432	320,480	480,800 480,854 600x1024	640x960
Large	480x800 400x854	480x800 480x854 600x1024		
Extra large	1024x600	1280x800 1024x768 1280x768	1536x1152 1920x1152 1920x1200	2048x1536 2560x1536 2560x1600

Screen sizes



How to create application supporting different screens?

- Preparations: declare sizes in the AndroidMainfest.xml
- Create different layouts for different screen sizes
  - res\layout-ldpi\main.xml
  - res\layout-mdpi\main.xml
- Naming convention

• smallestWidth	- sw<N>dp
• Available screen width	- w<N>dp
• Available screen height	- h<N>dp

- Example
  - Sw600dp
- How to use?
  - 320dp: typical (240x320 ldpi, 320x480 mdpi, 480x800 hdpi).
  - 600dp: 7” tablet (600x1024 mdpi).
  - 720dp: 10” tablet (720x1280 mdpi, 800x1280 mdpi).

Homework – Deadline 11/19 10.15 am

- Create a multilingual (English, Hungarian, …) application
  - A list of several items (such as music playlist)
    - A content of the list is arbitrary, it must contain as many items as they do not fit into the screen.
  - When the button has pressed some action have to happen
    - Log, display, etc.
  - The application must support
    - Landscape mode: the components are next to each other
    - Portrait mode: the components are vertically arranged

Kotlin

Next week