FindViewById using Kotlin

FindViewById using Kotlin

- In Java, *`findViewById()`* is used to recover views.
- **Kotlin Android Extensions** are Kotlin plugin will allow recovering views from Activities, Fragments, and Views in an amazing way.
- The plugin will allow you to **access views in the layout XML**, **just as if they were properties** with the name of the id you used in the layout definition.
- It also builds a local view cache. So the first time a property is used, it will do a regular *findViewById*. But next times, the view will be recovered from the cache, so the access will be faster.

Example: recovering views from the XML

- Imagine you have a layout resource activity_main.xml containing this:

```xml
<TextView
   android:id="@+id/welcomeMessage"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_gravity="center"
   android:text="Hello World!"/>
```

- Just go to your MainActivity and write it:

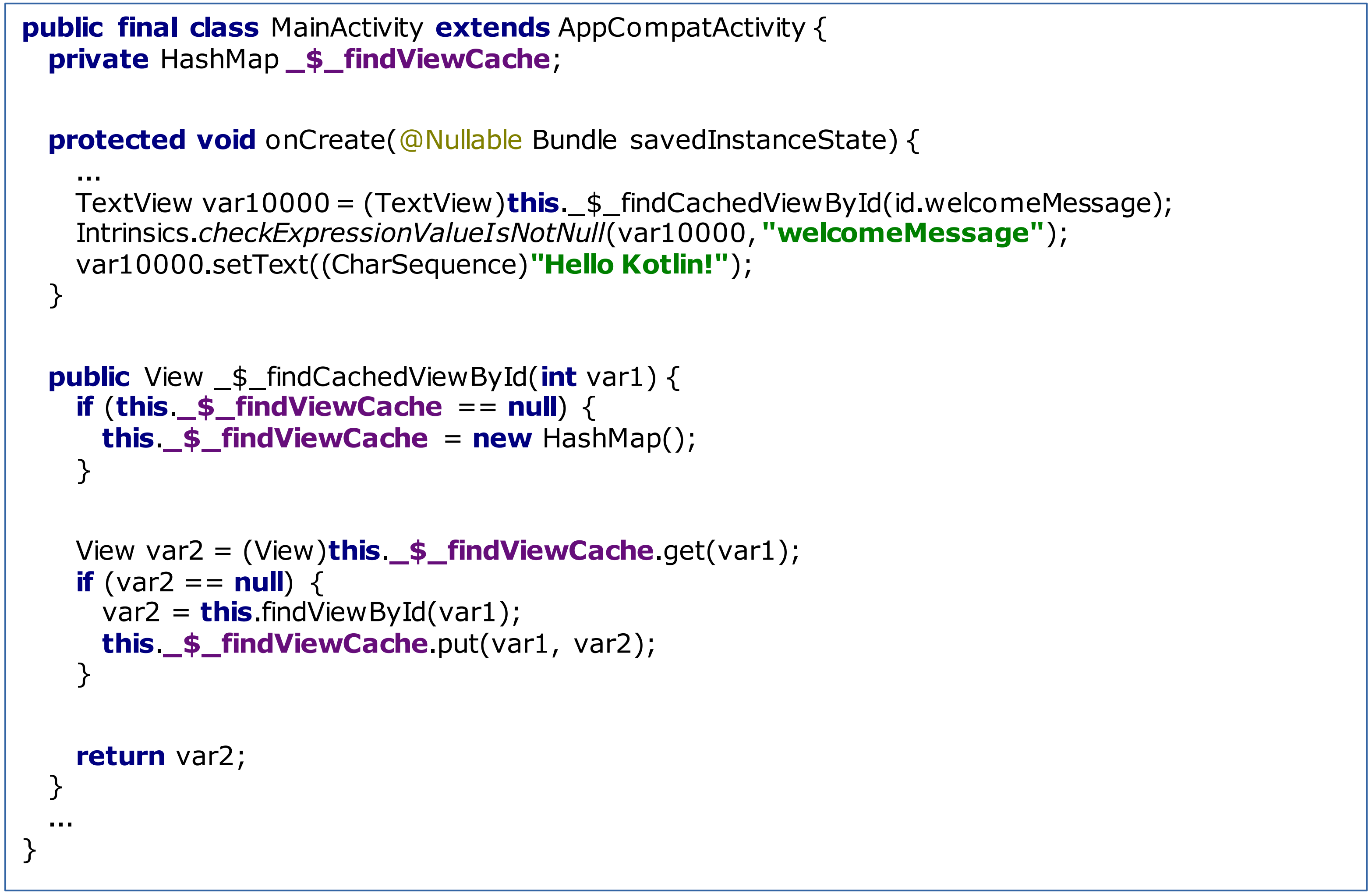
```kotlin
override fun onCreate(savedInstanceState: Bundle?) {
   super.onCreate(savedInstanceState)
   setContentView(R.layout.activity_main)
   welcomeMessage.text = "Hello Kotlin!"
}
```

- To be able to use it, you need a special import:

```kotlin
import kotlinx.android.synthetic.main.activity_main.*
```

The magic behind Kotlin Android Extensions

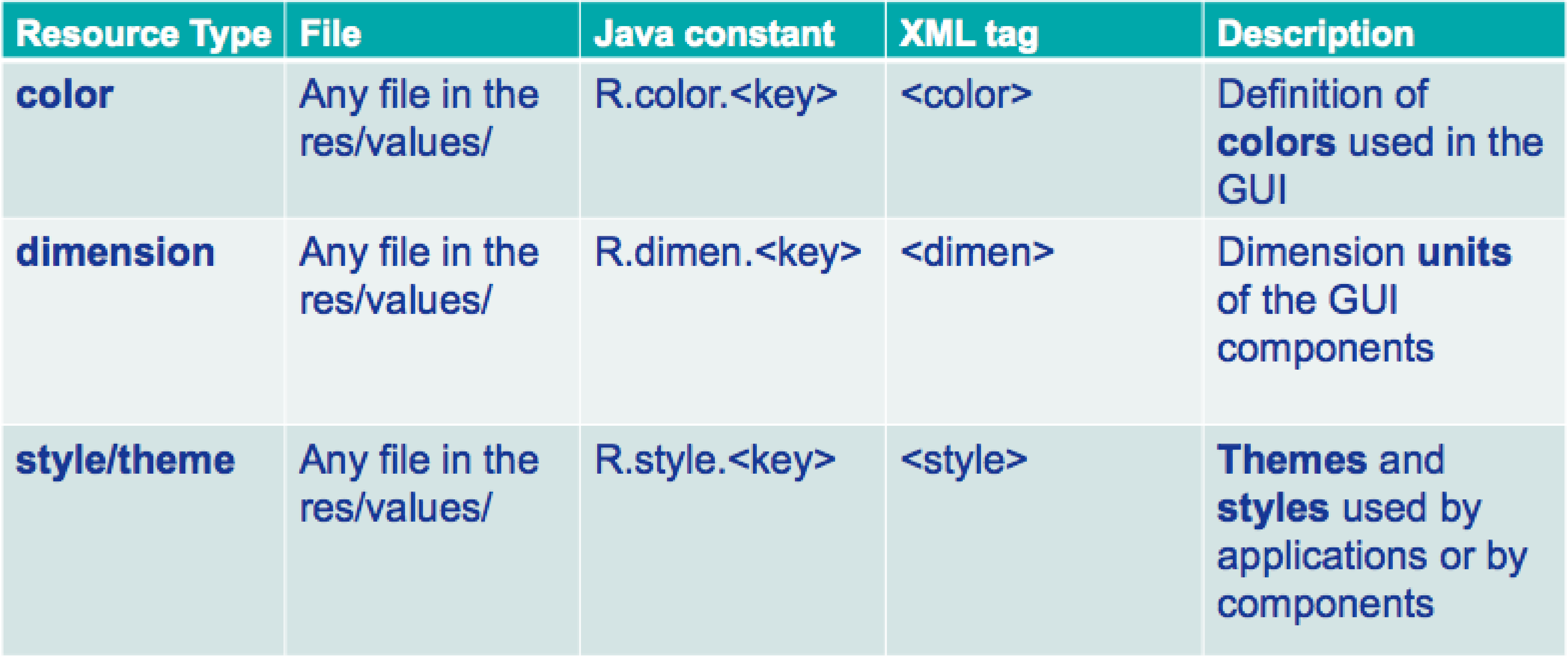
- it’s really interesting to understand the bytecode that is being generated when you use one feature or another.
- Tools –> Kotlin -> Kotlin Bytecode -> Decompile



```
public final class MainActivity extends AppCompatActivity {
   private HashMap _$_findViewCache;

   protected void onCreate(@Nullable Bundle savedInstanceState) {
      ...
      TextView var10000 = (TextView)this._$_findCachedViewById(id.welcomeMessage);
      Intrinsics.checkExpressionValueIsNotNull(var10000, "welcomeMessage");
      var10000.setText((CharSequence)"Hello Kotlin!");
   }

   public View _$_findCachedViewById(int var1) {
      if (this._$_findViewCache == null) {
         this._$_findViewCache = new HashMap();
      }

      View var2 = (View)this._$_findViewCache.get(var1);
      if (var2 == null) {
         var2 = this.findViewById(var1);
         this._$_findViewCache.put(var1, var2);
      }

      return var2;
   }
   ...
}
```

Resources

Resource types (2)



Resource Type	File	Java constant	XML tag	Description
color	Any file in the res/values/	R.color.<key>	<color>	Definition of **colors** used in the GUI
dimension	Any file in the res/values/	R.dimen.<key>	<dimen>	Dimension **units** of the GUI components
style/theme	Any file in the res/values/	R.style.<key>	<style>	**Themes** and **styles** used by applications or by components

Resource types: colors

Resources: colors

Color notation: #RGB #ARGB #RRGGBB #AARRGGBB

A,R,G,B - an hexadecimal digit
- R - red
- G - green
- B - blue
- A - transparency (alpha-channel)

colors can be represented by a 6-digit hexadecimal number: FFFFFF represents white, 000000 represents black, and so on.

Colors definition

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

      <color name=“red”> #FF0000 </color>
      <color name=“red_trasparent” > #66DDCCDD</color>

</resources>
```

Colors from Java code and XML resources

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

      <color name=“red”> #FF0000 </color>
      <color name=“red_trasparent” > #66DDCCDD</color>

</resources>
```

Colors definition

This application code retrieves the color resource:

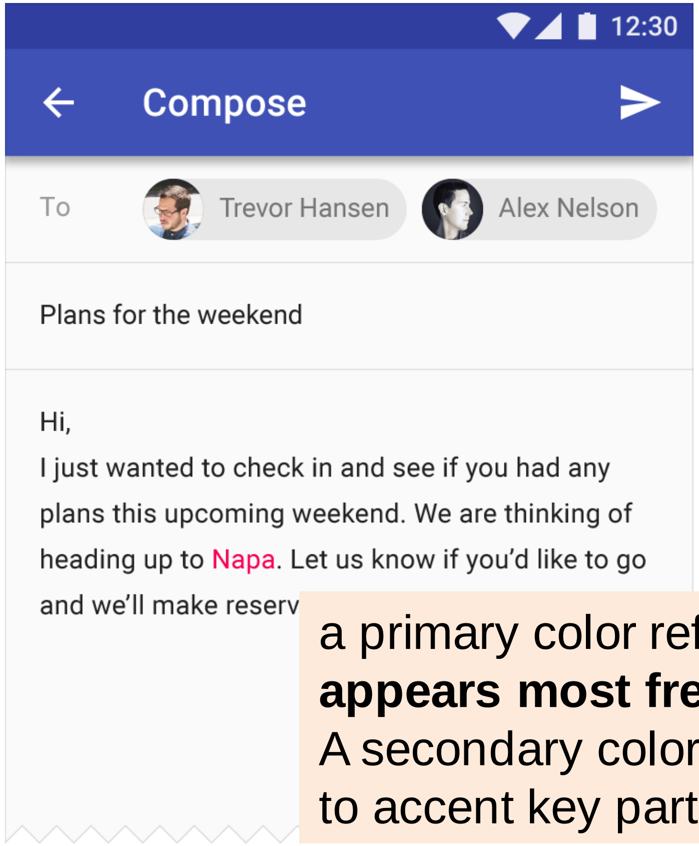
```
int redTransparent= getResources().getColor(R.color.red_transparent)
```

```
     <TextView
       android:layout_width="fill_parent"
       android:layout_height="wrap_content"
       android:textColor="@color/red_trasparent"
        android:text="Hello"/>
```

This layout XML applies the color to an attribute:

Choosing primary, secondary and accent colors

Primary text	#000000	87%
Secondary text	#000000	54%
Disabled / Hint text	#000000	38%
Primary color	#3E50B4	100%
Accent color	#FF3F80	100%



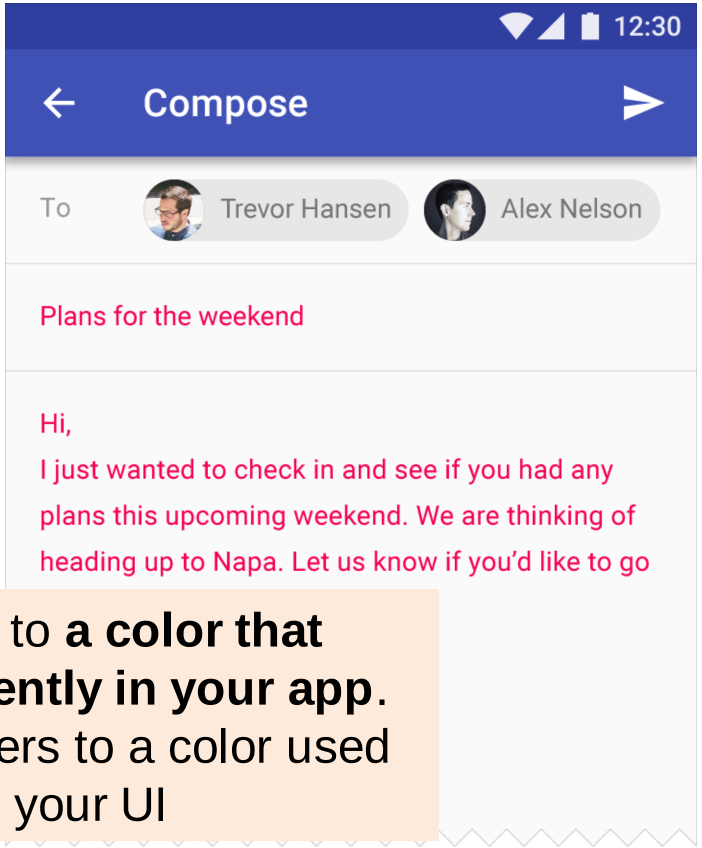
12:30

Compose

To Trevor Hansen Alex Nelson

Plans for the weekend

Hi,
I just wanted to check in and see if you had any plans this upcoming weekend. We are thinking of heading up to Napa. Let us know if you'd like to go and we'll make reserv



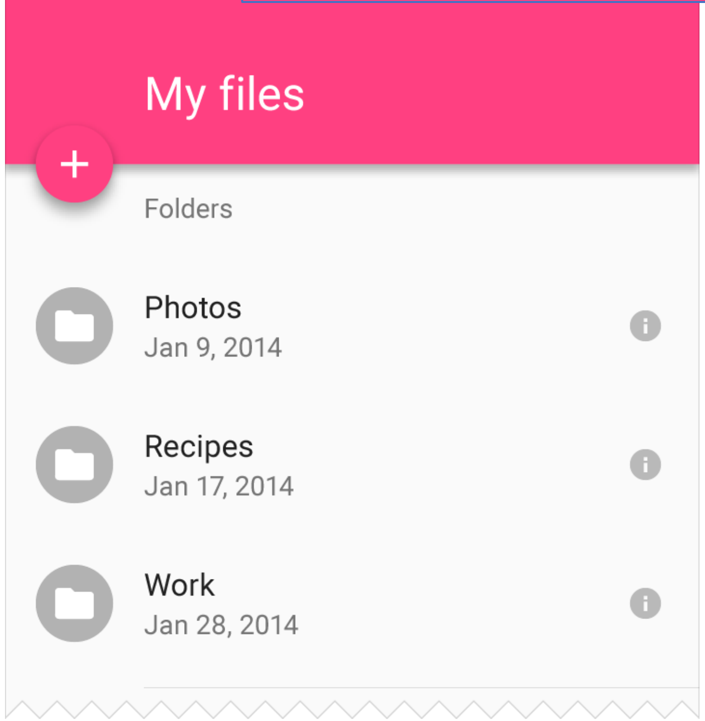
12:30

Compose

To Trevor Hansen Alex Nelson

Plans for the weekend

Hi,
I just wanted to check in and see if you had any plans this upcoming weekend. We are thinking of heading up to Napa. Let us know if you'd like to go



My files

Folders

Photos
Jan 9, 2014

Recipes
Jan 17, 2014

Work
Jan 28, 2014

a primary color refers to **a color that appears most frequently in your app**. A secondary color refers to a color used to accent key parts of your UI

Do.

Only use the accent color for body text to accent a web link.

Don't.

Don't use the accent color for body text color.

Don't.

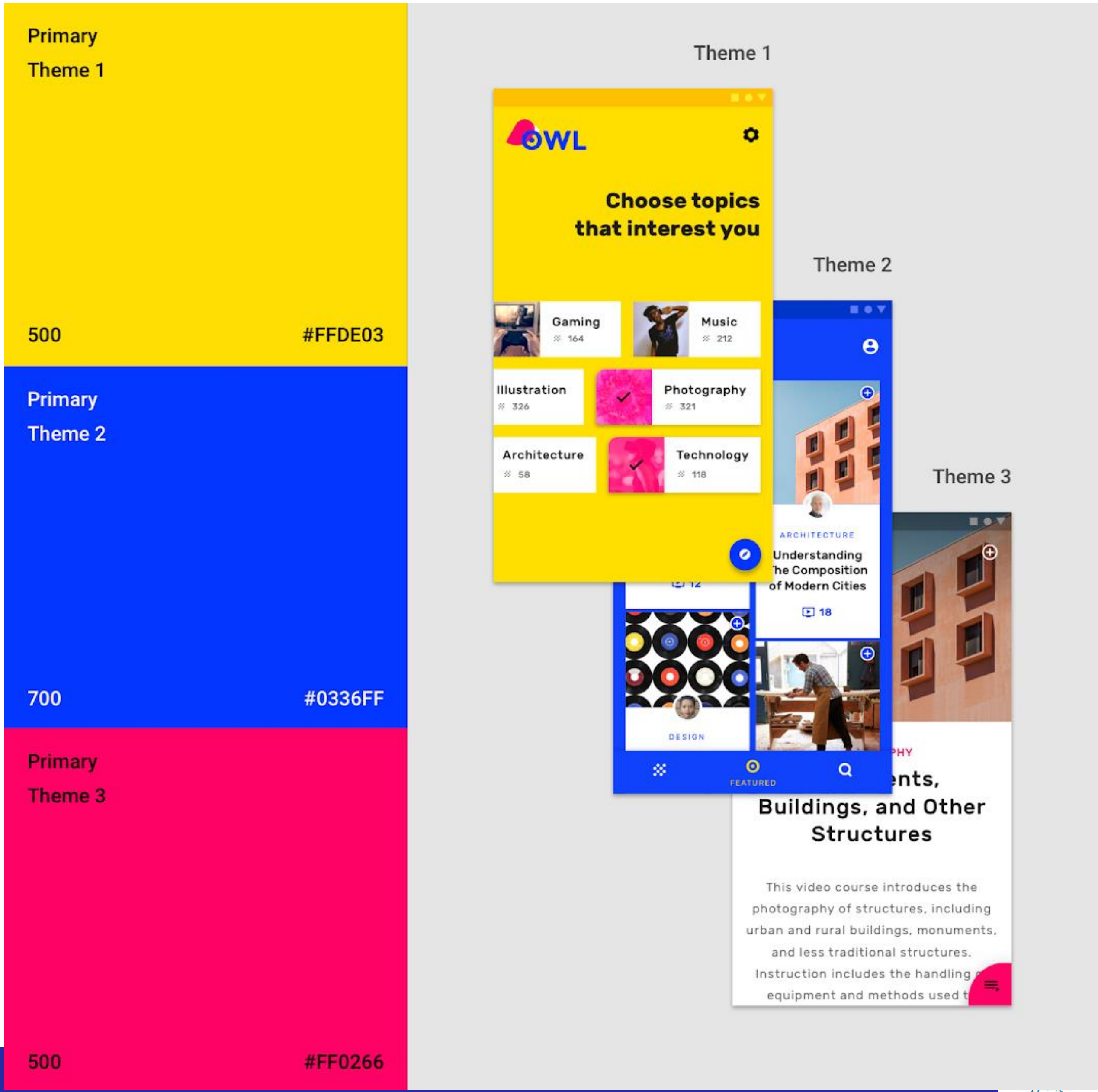
Don't use the accent color for app bars or larger areas of color. Avoid using the same color for the floating action button and the background.

https://material.io/guidelines/style/color.html#color-color-schemes

Alternative colors for section themes

Alternative colors can be used to theme different parts of an app.

Owl is an educational app that provides courses for people who want to...



Primary Theme 1
500 #FFDE03

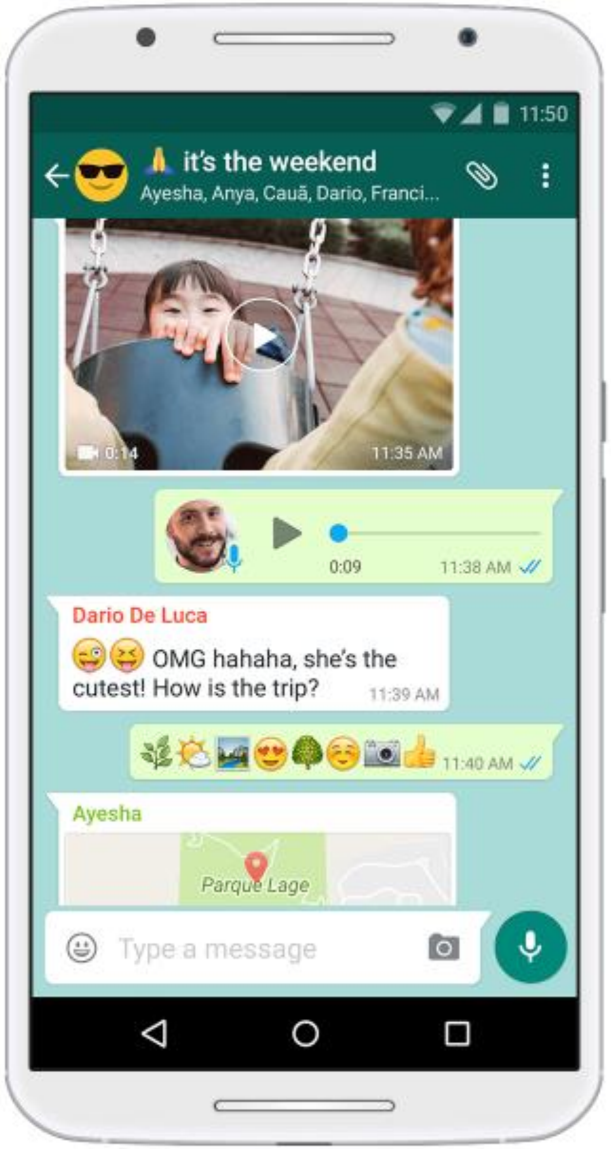
Primary Theme 2
700 #0336FF

Primary Theme 3
500 #FF0266

Theme 1
Theme 2
Theme 3
OWL
Choose topics that interest you
Gaming 164
Music 212
Illustration 326
Photography 321
Architecture 58
Technology 118
ARCHITECTURE
Understanding the Composition of Modern Cities
18
DESIGN
FEATURED
Buildings, and Other Structures
This video course introduces the photography of structures, including urban and rural buildings, monuments, and less traditional structures. Instruction includes the handling equipment and methods used t

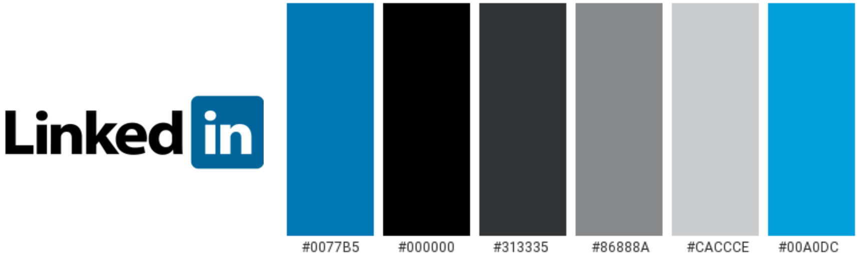
Material Design Style

- Material Design has a strict set of rules of how the user will navigate throughout the app:
- **how** items can be **added** or **removed**,
- what kind of **animations** are available,
- how much should be left for space between individual words
- and so on...

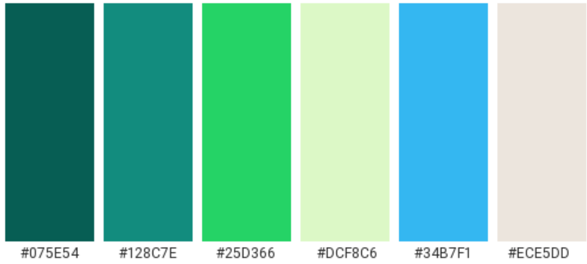


Dario De Luca
😜😝 OMG hahaha, she's the cutest! How is the trip?

Ayesha



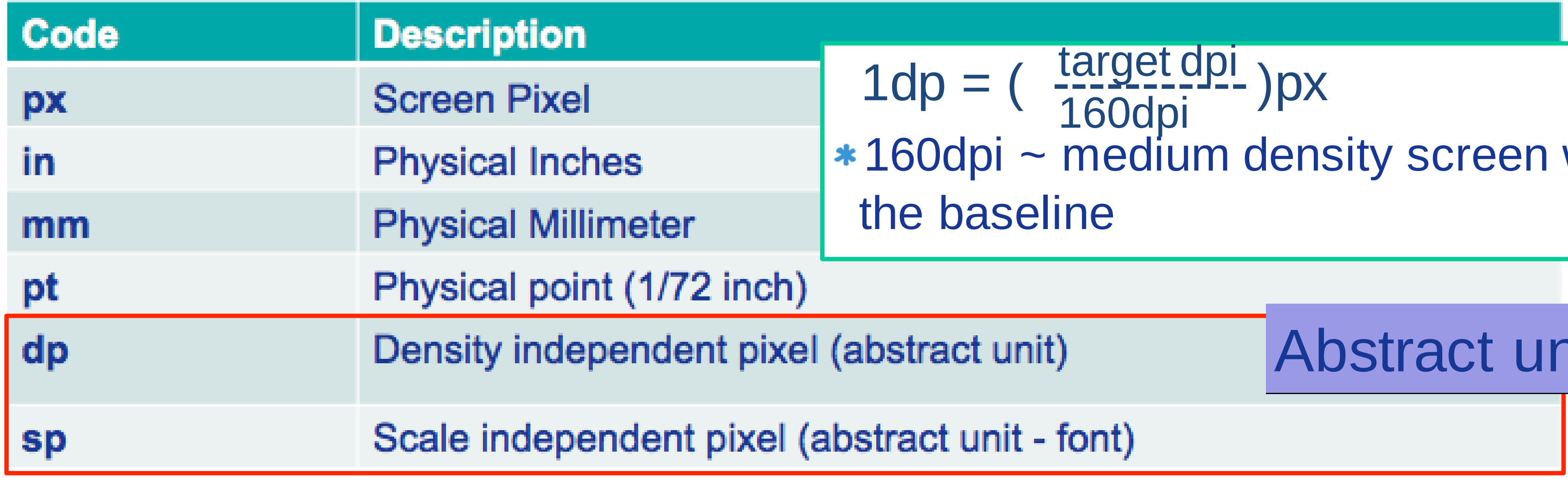
LinkedIn Website Color Scheme - by SchemeColor.com



WhatsApp Background Color Scheme - by SchemeColor.com

Resource types: dimension

Dimensions



Code	Description
px	Screen Pixel
in	Physical Inches
mm	Physical Millimeter
pt	Physical point (1/72 inch)
dp	Density independent pixel (abstract unit)
sp	Scale independent pixel (abstract unit - font)

$$1dp = \left(\frac{\text{target dpi}}{\text{160dpi}} \right) px$$

- 160dpi ~ medium density screen which is the baseline

Abstract units

Dp (Density Pixel):

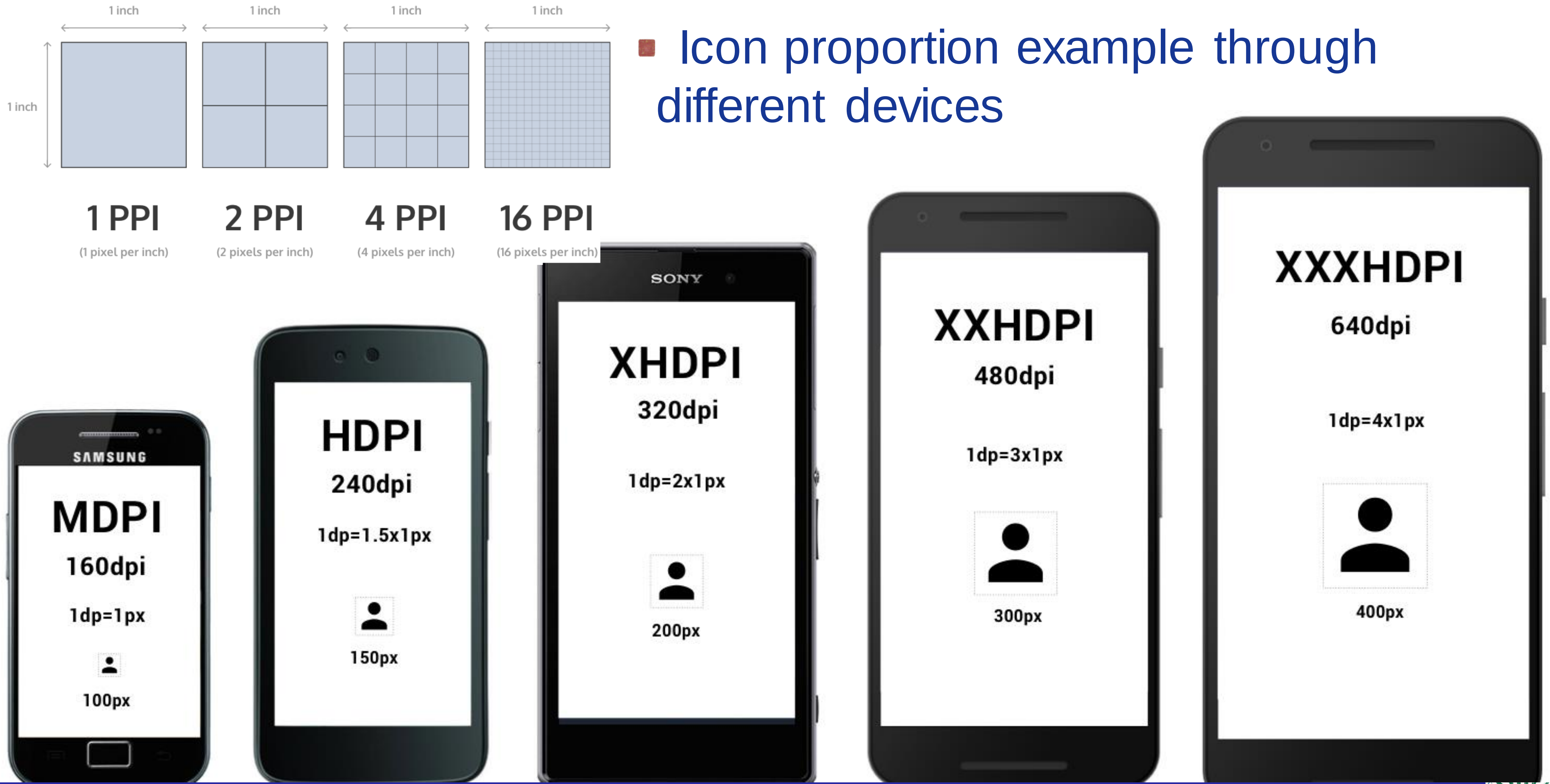
- Relative to a 160 dpi (dots per inch) screen, on which 1dp≈1px.
- Higher density screen: the number of pixels used to draw 1dp is scaled up by a factor appropriate for the screen's dpi.
- On a lower density screen, the number of pixels used for 1dp is scaled down

Sp (Scale Pixel):

- This is like the dp unit, but it is also scaled by the user's font size preference.

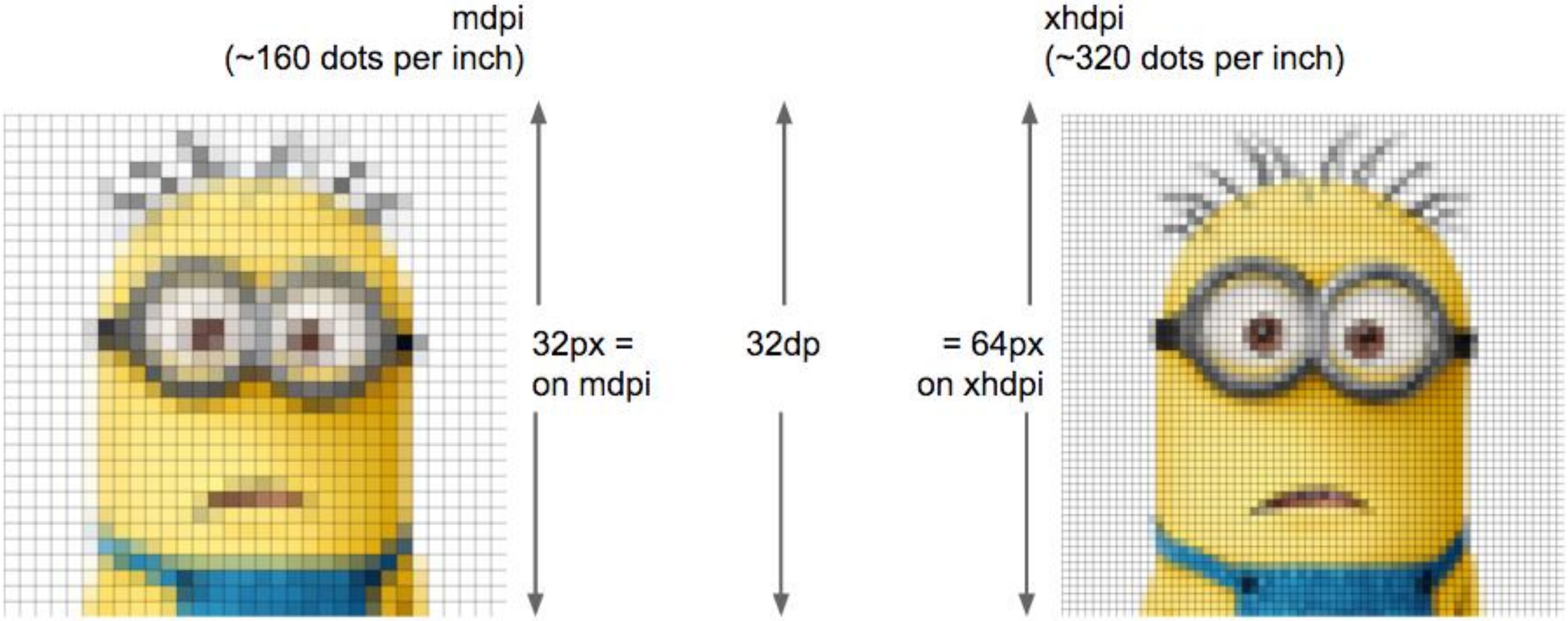
Android Screen Density

- Icon proportion example through different devices



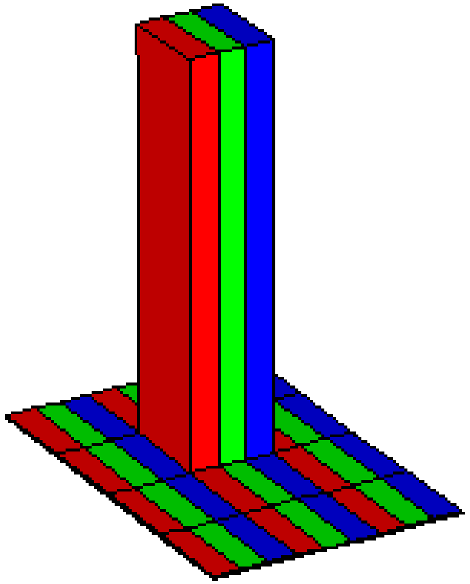
Android Screen Density

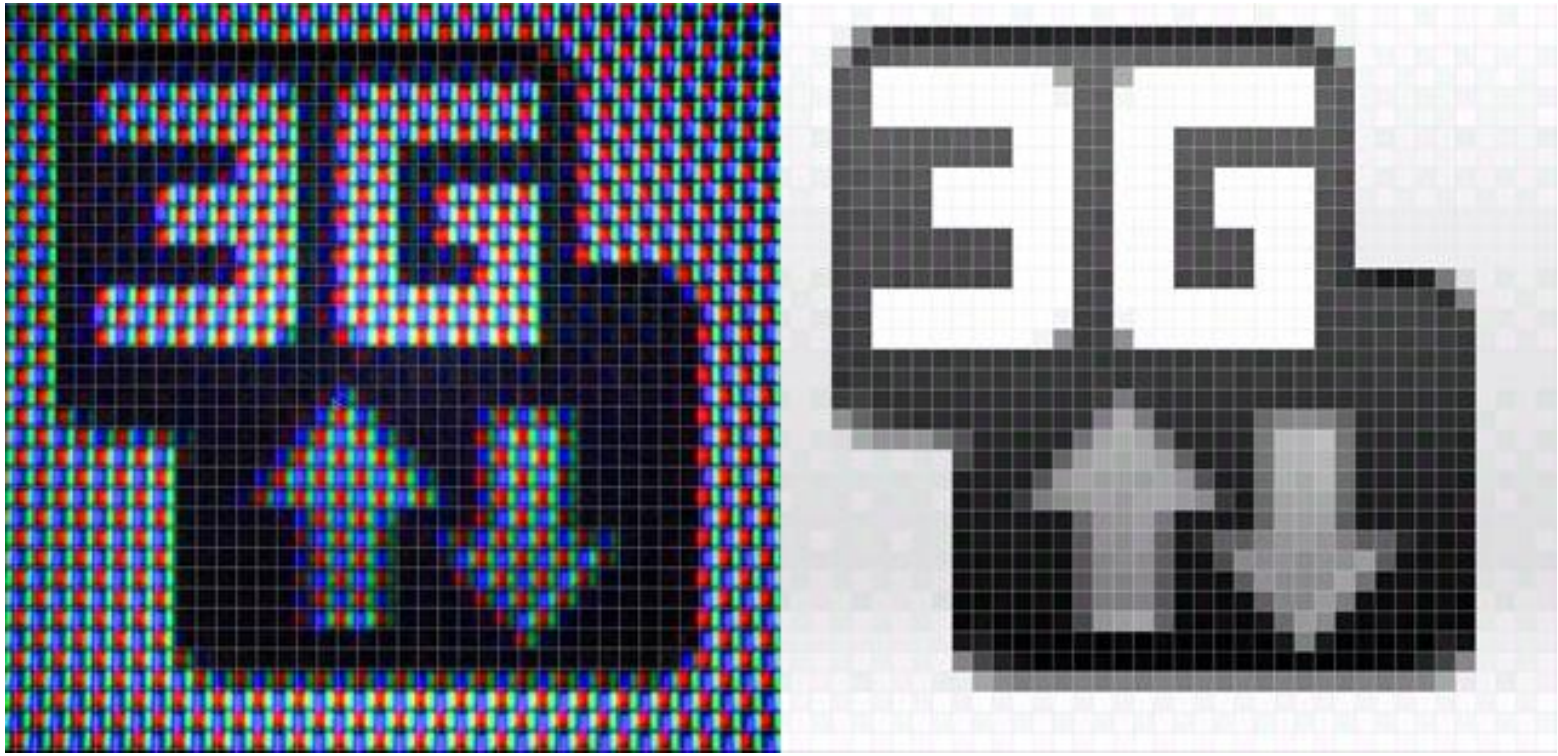
- Icon proportion example through different devices



Pixel

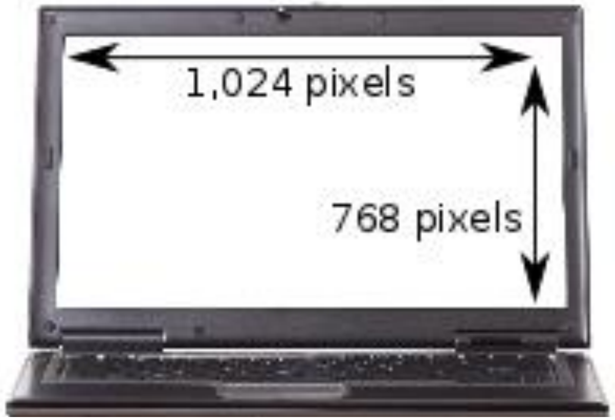
- A pixel is the smallest element that can be displayed on a screen.
- A pixel is just a single **dot** on a display or a screen, or the smallest addressable element in an all points addressable display device; so it is the smallest controllable element of a picture represented on the screen

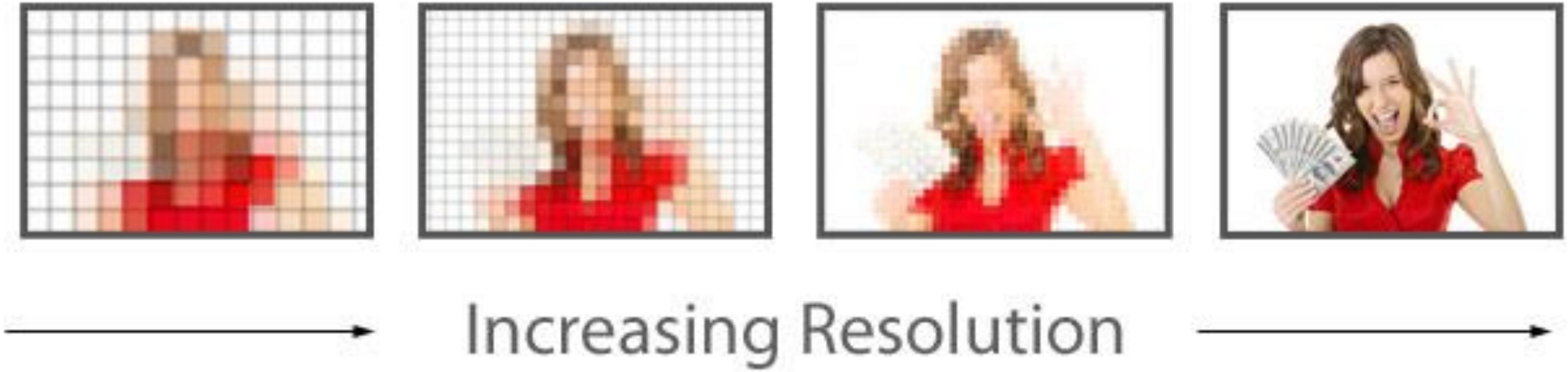




Resolution

- **Screen resolution**: width × height, with the units in pixels.
- E.g. Resolution of 1024×768 pixels means that the there are 1024 pixels horizontally and 768 pixels vertically. So a total of 786432 pixel in the display.

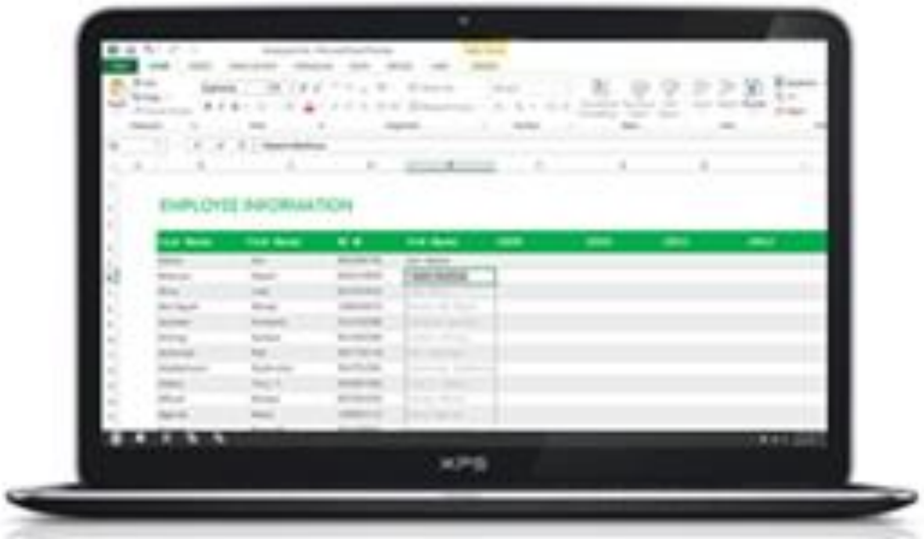




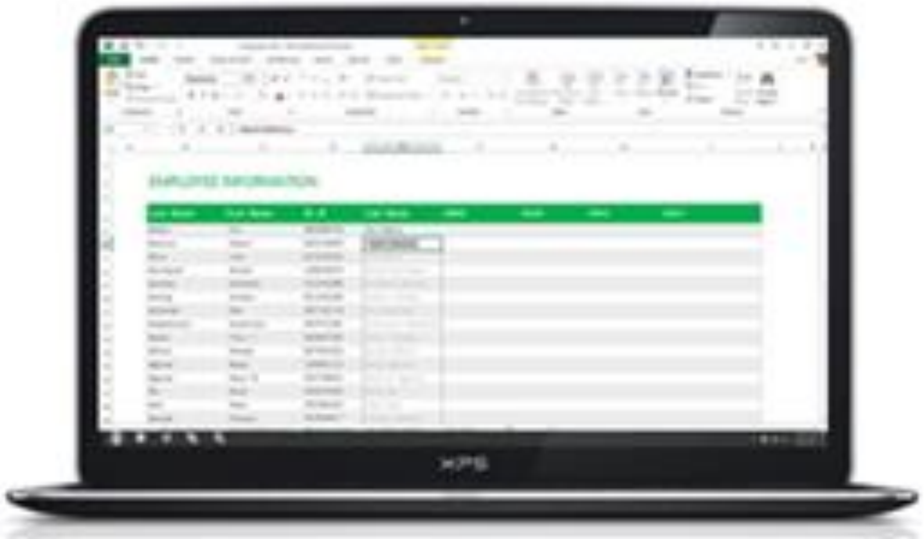
What higher resolution means

- Comparing two screens of the same size but with different resolutions,
  higher resolution –-> will be able to show you more
- Higher resolution also means that elements on the screen - like icons and text - will look smaller.

XPS 13 (720p)



XPS 13 (1080p)

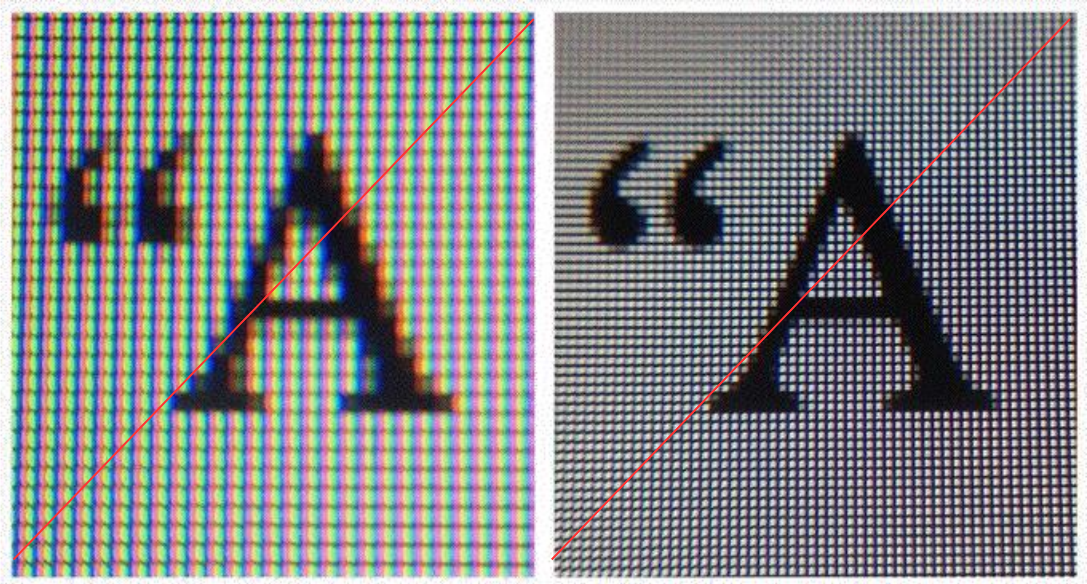


Screen Density

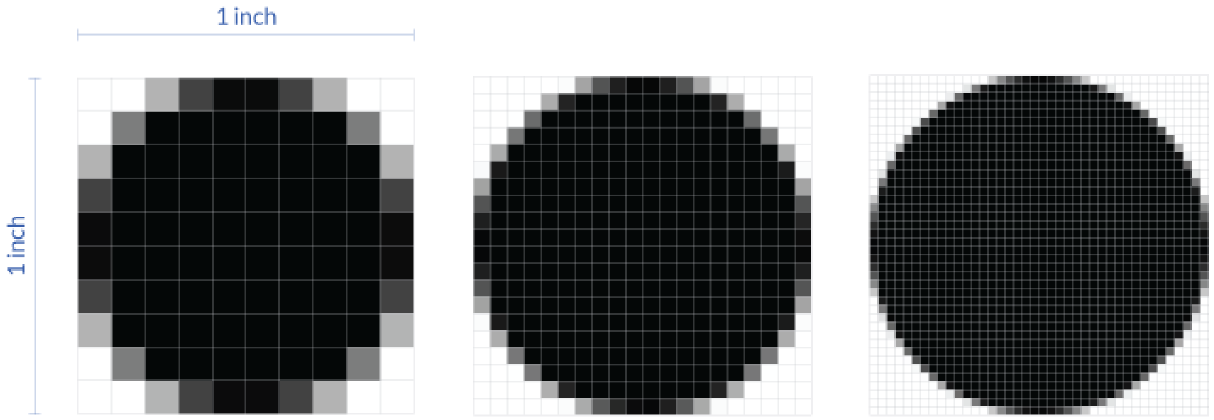
- **Screen density:** is a ratio of resolution and display size, which can be quantified as **Dots Per Inch (DPI)** or Pixels Per Inch **(PPI)**.
  - The higher the dpi, the smaller each individual pixel is, and the greater clarity.
  - A higher dpi means
    more detail is displayed per inch

- **DPI =**

$$\frac{\textbf{Diagonal resolution in Pixel}}{\textbf{Diagonal size in Inch}}$$



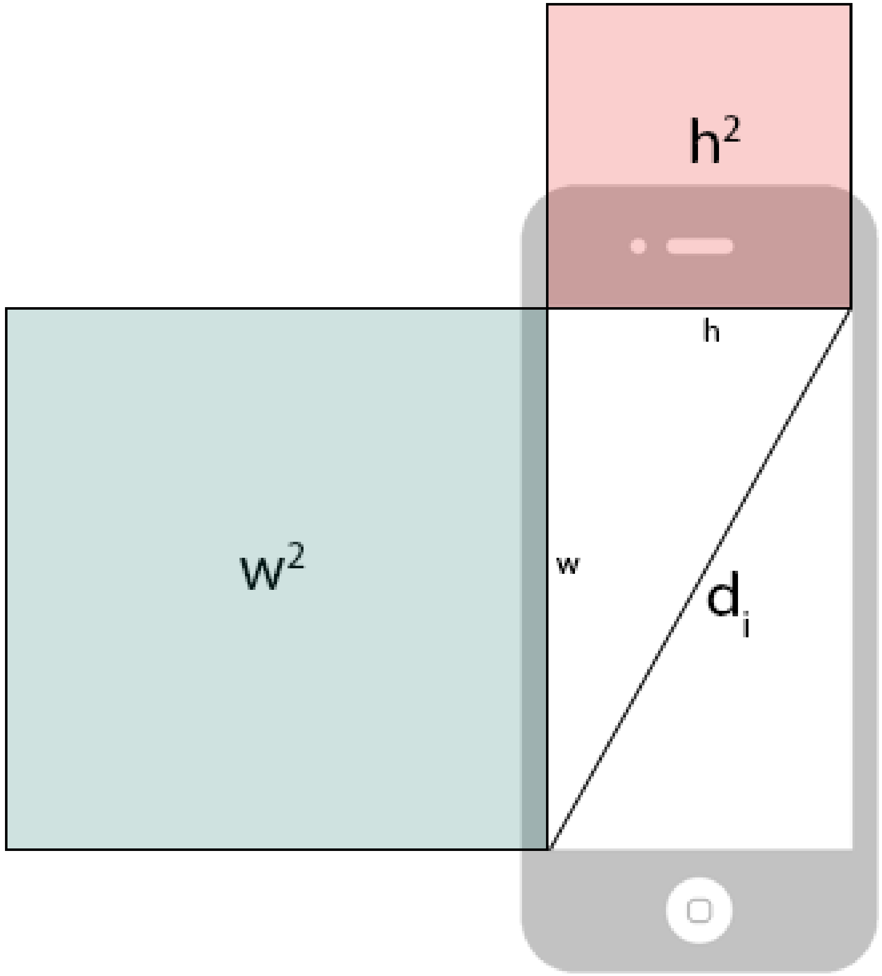
Calculating Pixels Per Inch (PPI)



10PPI

20PPI

40PPI



$$d_p = \sqrt{w^2 + h^2}$$

$$PPI = \frac{d_p}{d_i} \quad \text{where}$$

w is width resolution in pixels

h is height resolution in pixels

d_p is diagonal resolution in pixels

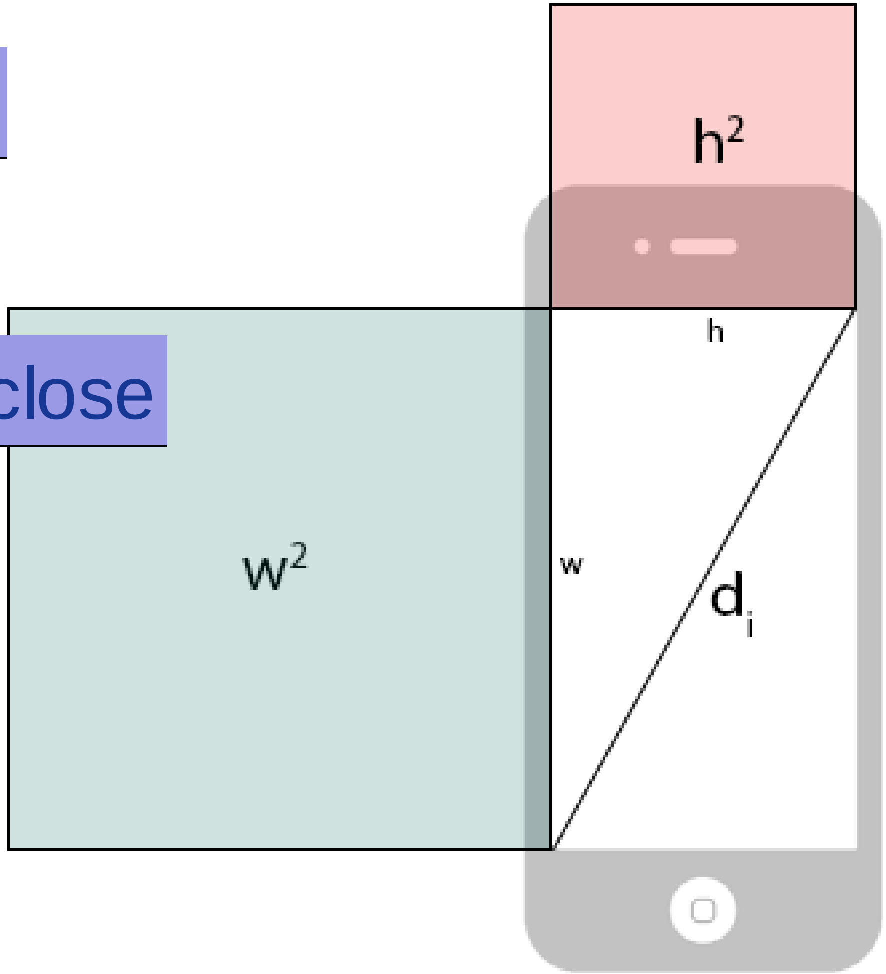
d_i is diagonal size in inches (this is the number advertised as the size of the display)

Resolution vs density

- Galaxy Nexus (4.65” diagonal)
  - resolution: **720x1280** px
  - density: **316dpi**
- Nexus 7 (7” diagonal)
  - resolution: **800x1280** px
  - density: **216 dpi**

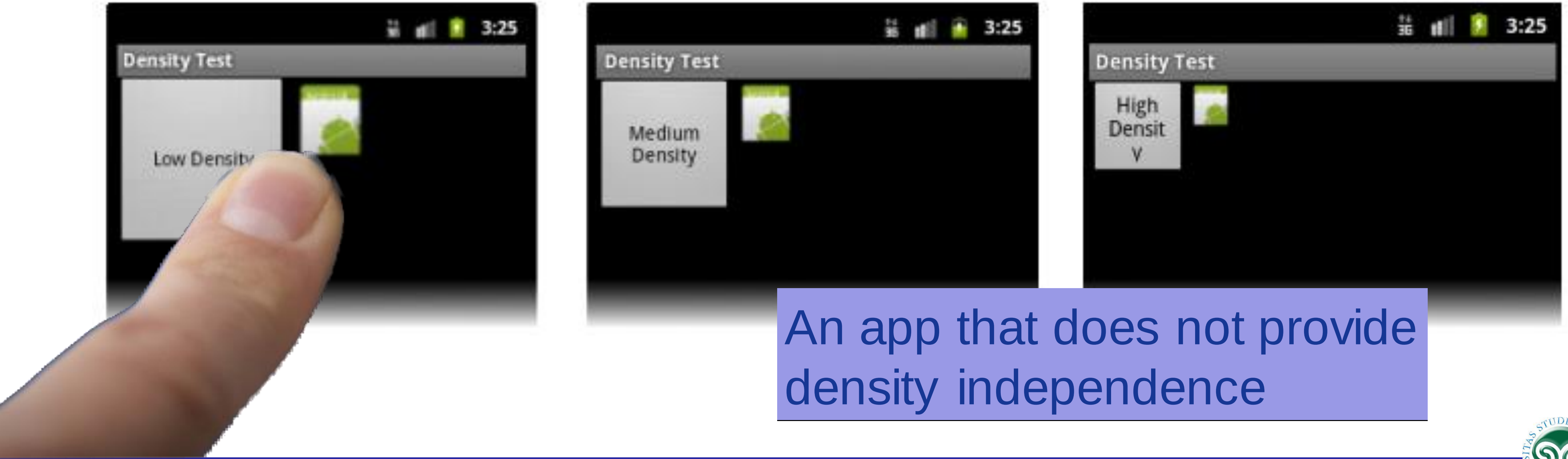
Similar

Not even close



Android Density independence

- Your application achieves "density independence" when it preserves the physical size of user interface elements when displayed on screens with different densities (dpi).
- Why? Simple, a user’s **finger** is the same physical size no matter what the screen density is.

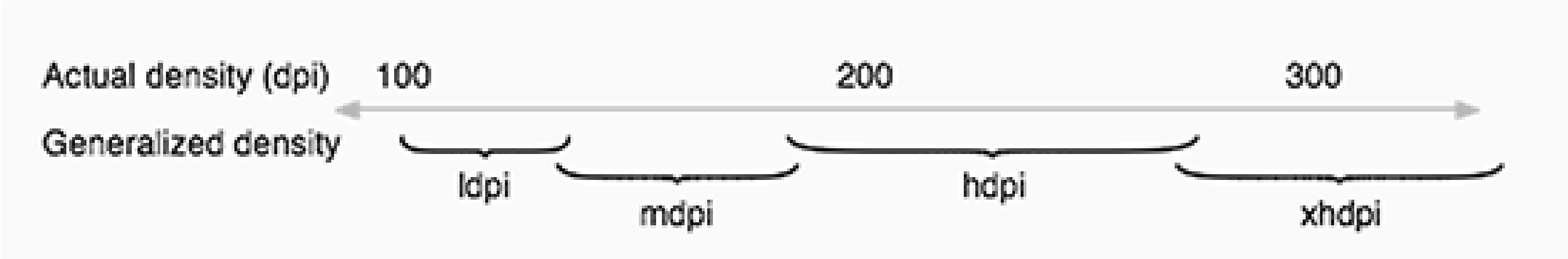


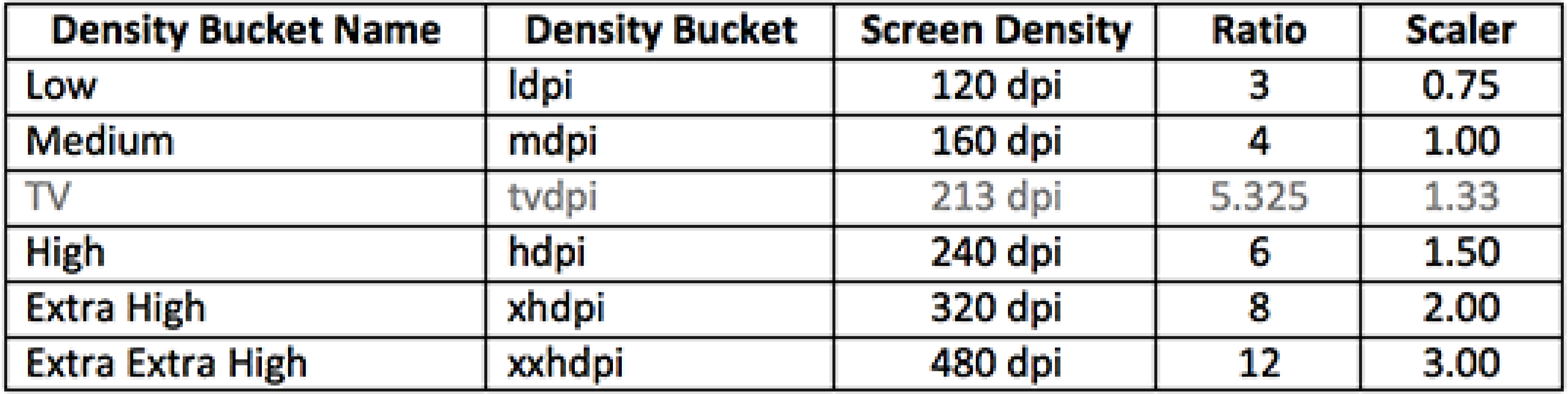
An app that does not provide density independence

Density bucket

- There is a myriad of Android devices with varying screen densities, which can range from 100 dpi to over 480 dpi.
- In order to optimize images for all these screen densities, images need to be created at different resolutions.
- However, trying to optimize every image resource for every possible density would be incredibly tedious, cause app sizes to be enormous, and simply is not a feasible solution.
- As a compromise, Android uses density **buckets** that are used to group devices together within certain screen density ranges.
- This way, apps are only required to optimize images for each density bucket, instead of every possible density.

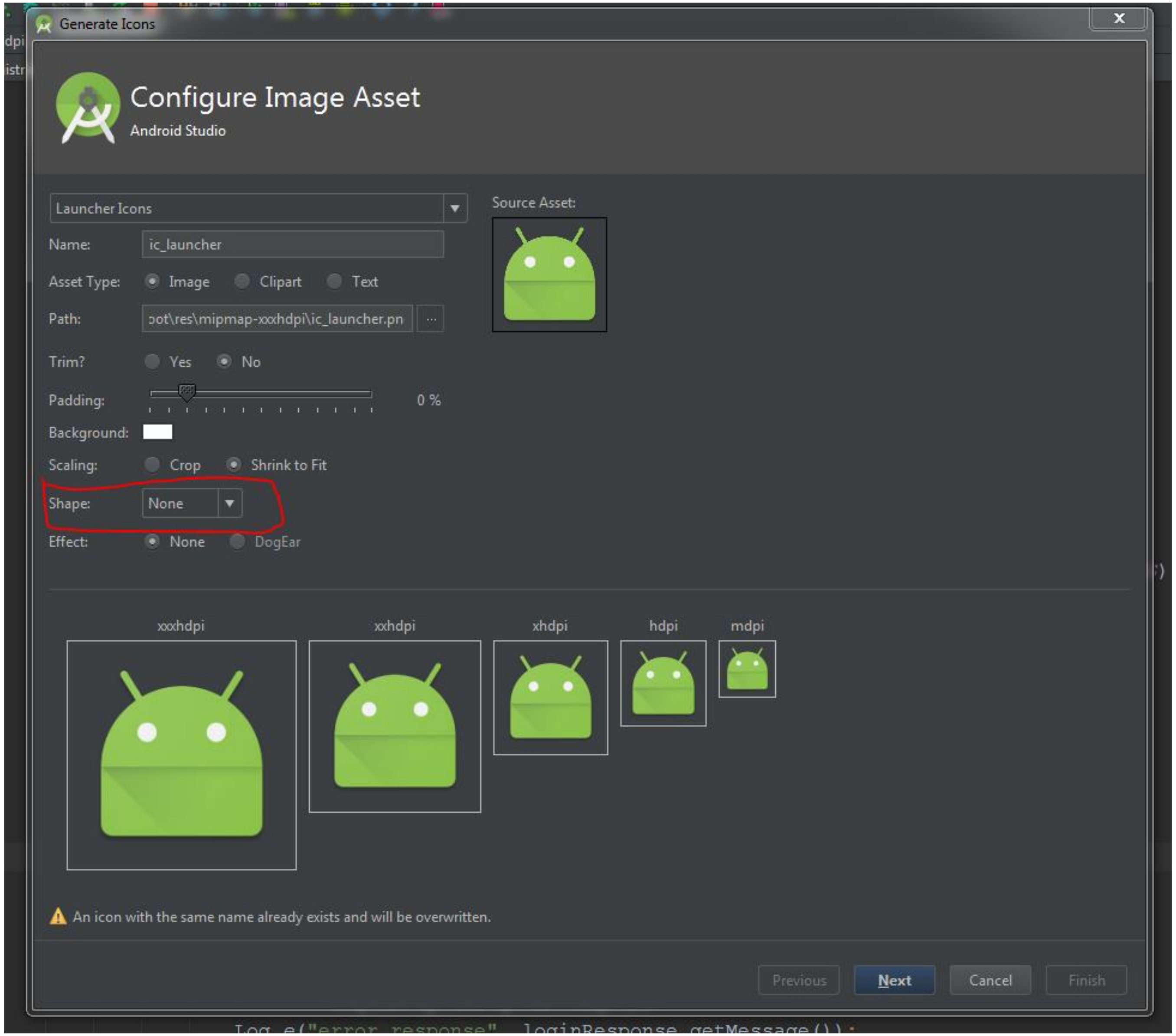
Density Bucket





Density Bucket Name	Density Bucket	Screen Density	Ratio	Scaler
Low	ldpi	120 dpi	3	0.75
Medium	mdpi	160 dpi	4	1.00
TV	tvdpi	213 dpi	5.325	1.33
High	hdpi	240 dpi	6	1.50
Extra High	xhdpi	320 dpi	8	2.00
Extra Extra High	xxhdpi	480 dpi	12	3.00

Density Bucket in Android Studio



Generate Icons

Configure Image Asset

Android Studio

Launcher Icons

Name: ic_launcher

Asset Type: ● Image ○ Clipart ○ Text

Path: oot\res\mipmap-xxxhdpi\ic_launcher.pn

Trim? ○ Yes ● No

Padding: 0 %

Background:

Scaling: ○ Crop ● Shrink to Fit

Shape: None

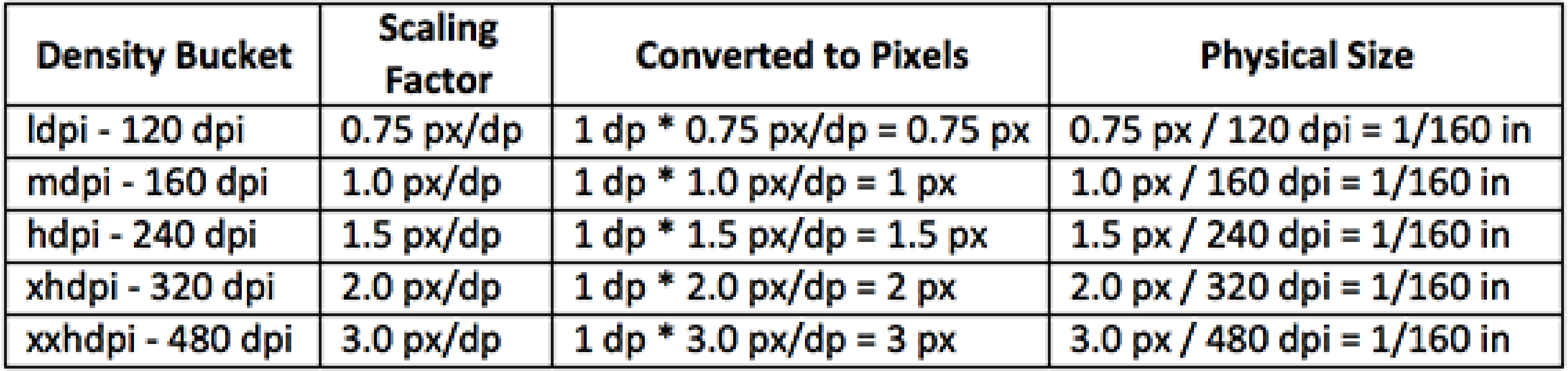
Effect: ● None ○ DogEar

Source Asset:

An icon with the same name already exists and will be overwritten.

How Density Pixel and Scale Pixel are computed into pixels

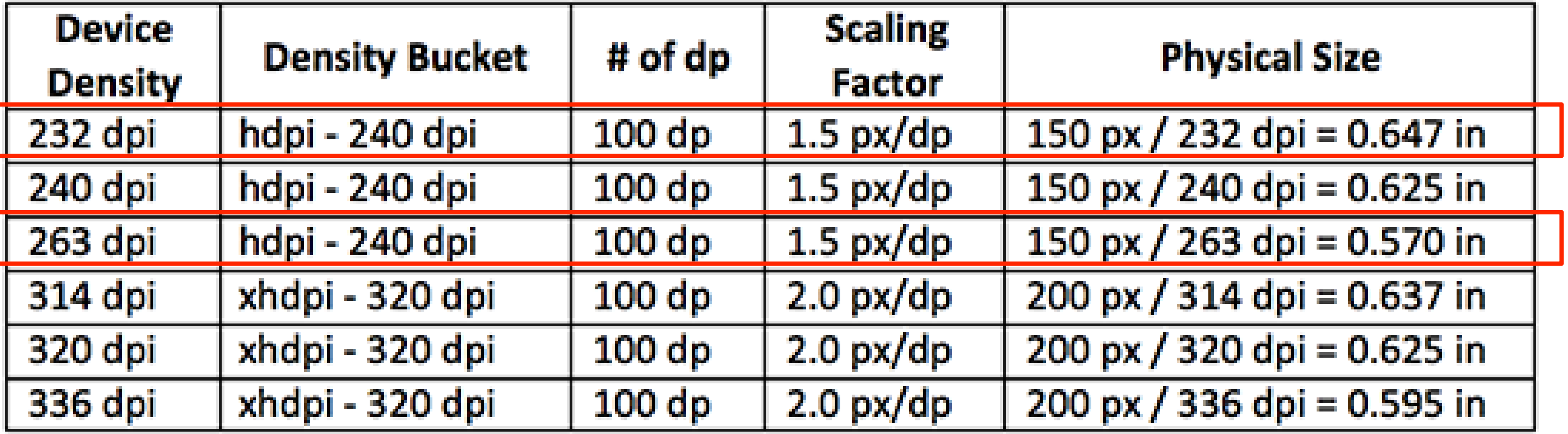
- Android uses mdpi, 160 dpi, as its baseline density, 1dp≈1px.
- Depending on the ratio between a devices density bucket and the baseline density, a scaling factor is applied to convert **dp** to **px**



Density Bucket	Scaling Factor	Converted to Pixels	Physical Size
ldpi - 120 dpi	0.75 px/dp	1 dp * 0.75 px/dp = 0.75 px	0.75 px / 120 dpi = 1/160 in
mdpi - 160 dpi	1.0 px/dp	1 dp * 1.0 px/dp = 1 px	1.0 px / 160 dpi = 1/160 in
hdpi - 240 dpi	1.5 px/dp	1 dp * 1.5 px/dp = 1.5 px	1.5 px / 240 dpi = 1/160 in
xhdpi - 320 dpi	2.0 px/dp	1 dp * 2.0 px/dp = 2 px	2.0 px / 320 dpi = 1/160 in
xxhdpi - 480 dpi	3.0 px/dp	1 dp * 3.0 px/dp = 3 px	3.0 px / 480 dpi = 1/160 in

But not the same physical size on different devices

- The reason **dp** tends to vary in physical size is due to the same scaling factor being applied for the entire density bucket.
- The scaling factor is computed with the density bucket’s **dpi**, and not the device’s actual dpi.



Device Density	Density Bucket	# of dp	Scaling Factor	Physical Size
232 dpi	hdpi - 240 dpi	100 dp	1.5 px/dp	150 px / 232 dpi = 0.647 in
240 dpi	hdpi - 240 dpi	100 dp	1.5 px/dp	150 px / 240 dpi = 0.625 in
263 dpi	hdpi - 240 dpi	100 dp	1.5 px/dp	150 px / 263 dpi = 0.570 in
314 dpi	xhdpi - 320 dpi	100 dp	2.0 px/dp	200 px / 314 dpi = 0.637 in
320 dpi	xhdpi - 320 dpi	100 dp	2.0 px/dp	200 px / 320 dpi = 0.625 in
336 dpi	xhdpi - 320 dpi	100 dp	2.0 px/dp	200 px / 336 dpi = 0.595 in

Dimensions

Dimensions definition

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
       <dimen name="textview_height">25dp</dimen>
       <dimen name="textview_width">150dp</dimen>
       <dimen name="font_size">16sp</dimen>
</resources>
```

Layout: dimensions application

```
<TextView
    android:layout_height="@dimen/textview_height"
    android:layout_width="@dimen/textview_width"
    android:textSize="@dimen/font_size"/>
```

Resource types: style

Styles

- A ***style*** is a set of attributes that can be applied to a **specific component of the GUI** (View) or to the whole screen or application (in this case, it is also referred as “theme”).
- A ***style is an XML resource*** that is referenced using the value provided in the name attribute.
- Styles can be ***organized in a hierarchical structure***.
  - A style can inherit properties from another style, through the parent attribute.
- Use `<style></style>` tags to define a style in the res/ folder. Use `<item>` to define the attributes of the style

Styles

Style definition

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
       <style name="CustomText"
parent="@style/Text">
                  <item
name="android:textSize">20sp</item>
                  <item
name="android:textColor">#008</item>
      </style>
</resources>
```

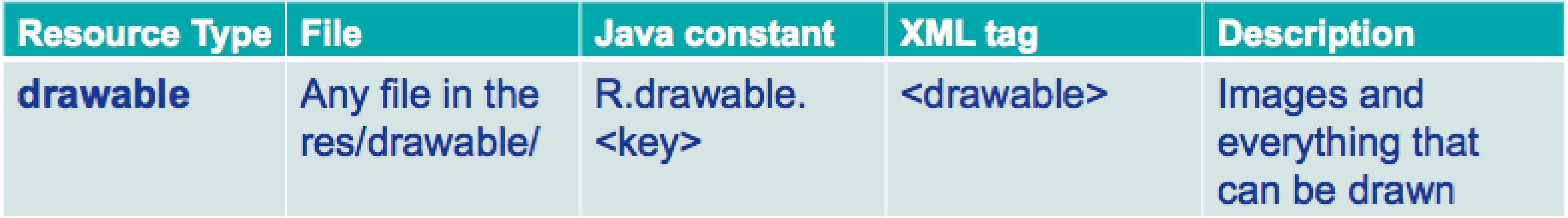
Layout: style application

```
<EditText style="@style/CustomText"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Hello, World!" />
```

Editable text field in layout

Resource types: drawable

Resources: drawable



Resource Type	File	Java constant	XML tag	Description
drawable	Any file in the res/drawable/	R.drawable.<key>	<drawable>	Images and everything that can be drawn

- Drawable: general concept for a graphic that can be drawn on the screen:
  - Images
  - XML resources: `android:drawable` and `android:icon` (e.g., a Button can have a drawable resource as background).

XML drawable resources

http://developer.android.com/guide/topics/resources/drawable-resource.html

Resources: drawable

- ***BitMap***: `.png, .jpg, .gif` files.
  - Android creates a BitMap resource for any of these files saved in the res/drawable directory.

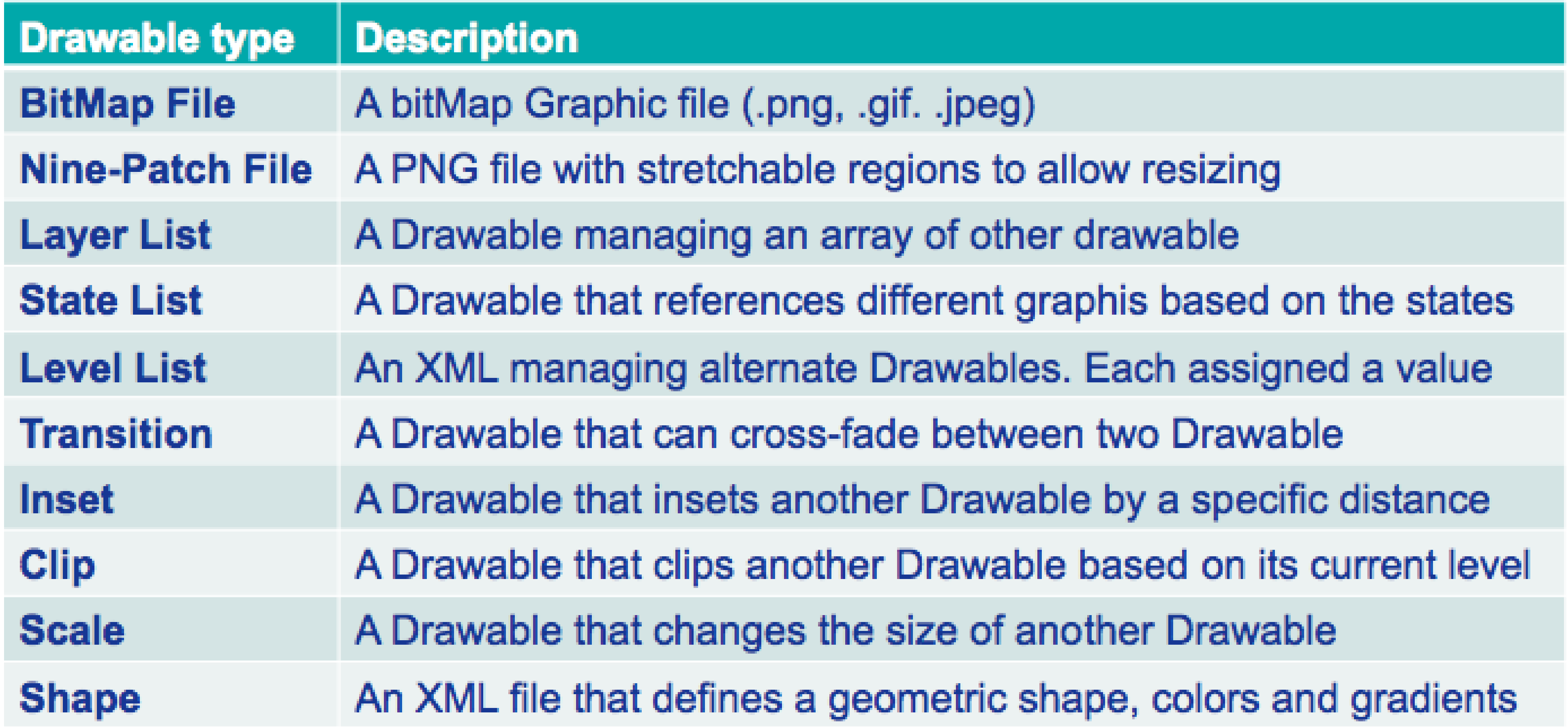
Layout: display image

```
<ImageView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:src=”drawable/image" />
```

Code: retrive image

```
Drawable draw = res.getDrawable(R.drawable.image)
```

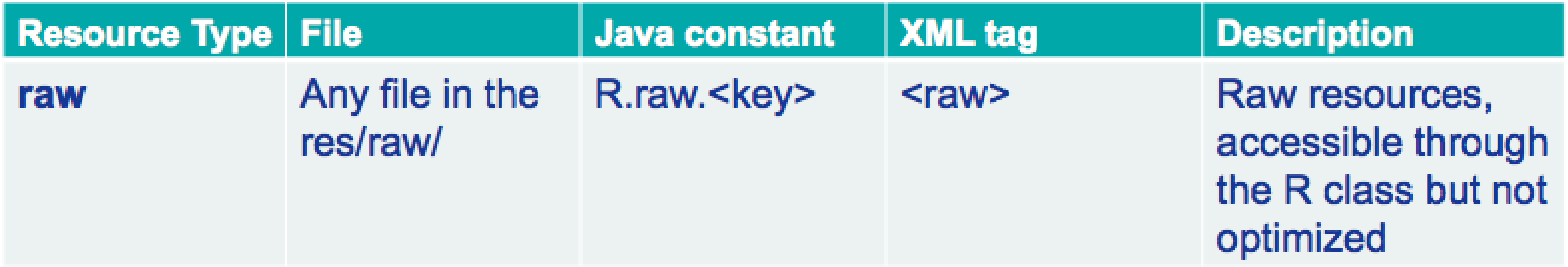
Drawable



Drawable type	Description
BitMap File	A bitMap Graphic file (.png, .gif. .jpeg)
Nine-Patch File	A PNG file with stretchable regions to allow resizing
Layer List	A Drawable managing an array of other drawable
State List	A Drawable that references different graphis based on the states
Level List	An XML managing alternate Drawables. Each assigned a value
Transition	A Drawable that can cross-fade between two Drawable
Inset	A Drawable that insets another Drawable by a specific distance
Clip	A Drawable that clips another Drawable based on its current level
Scale	A Drawable that changes the size of another Drawable
Shape	An XML file that defines a geometric shape, colors and gradients

Complete list:
http://developer.android.com/guide/topics/resources/drawable-resource.html

Raw

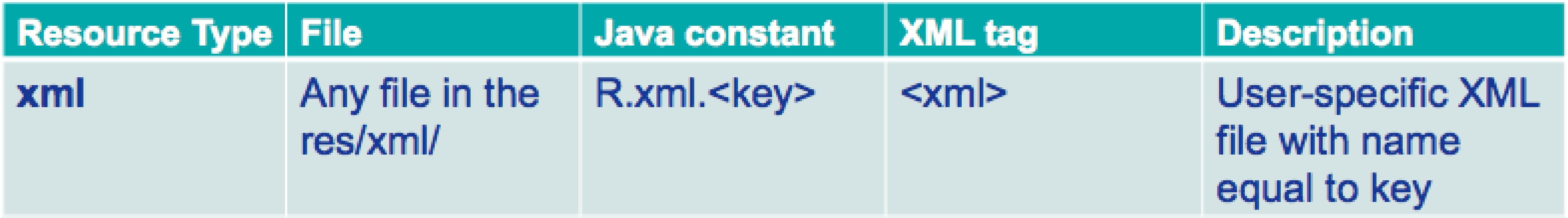


Resource Type	File	Java constant	XML tag	Description
raw	Any file in the res/raw/	R.raw.<key>	<raw>	Raw resources, accessible through the R class but not optimized

- *raw*: resources for which no run-time optimization must be performed (e.g. audio/video files). They can be accessed as a stream of bytes, by using Java InputStream objects:

```
InputStream is = getResources().openRawResource(R.raw.videoFile)
```

XML



Resource Type	File	Java constant	XML tag	Description
xml	Any file in the res/xml/	R.xml.<key>	<xml>	User-specific XML file with name equal to key

- ***xml***: arbitrary XML files that can be read at runtime through the `R.xml.<filename>` constant.
- It is possible to parse the XML file through a `XMLResourceParser`.

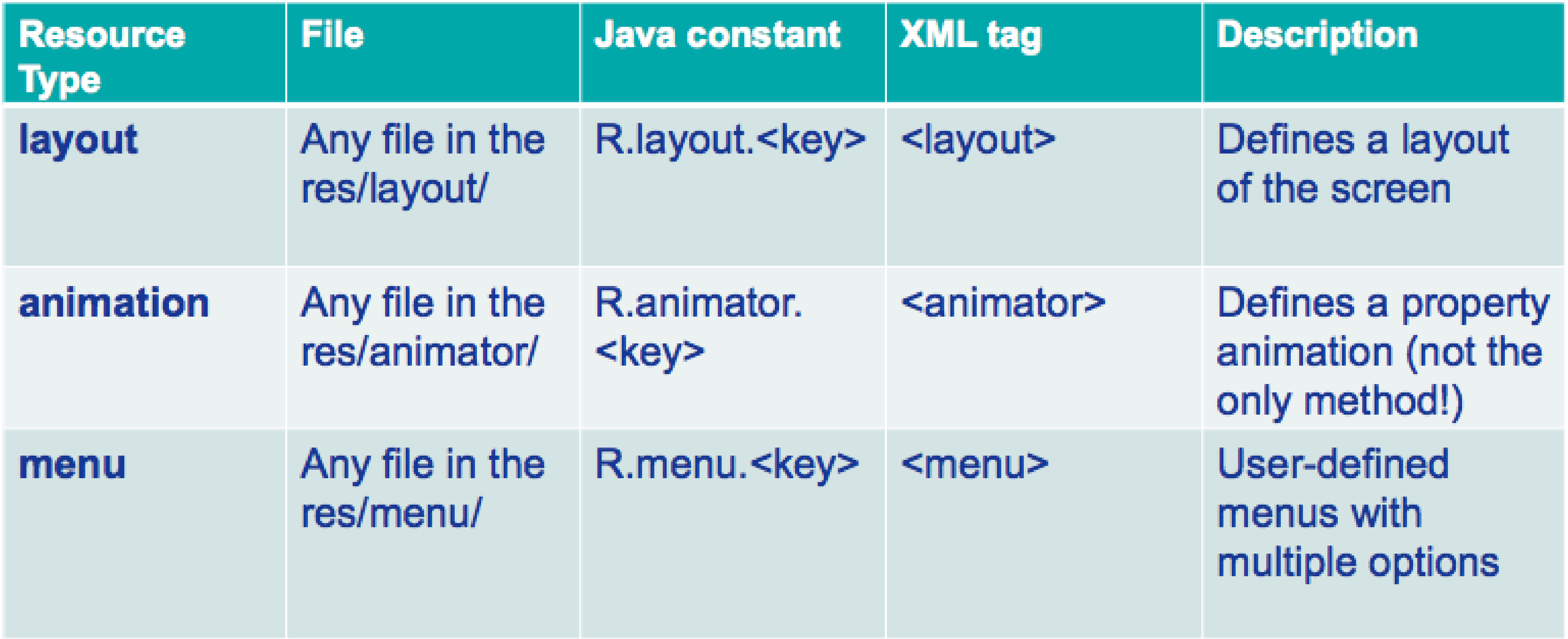
```
XMLResourceParser parser = getResources().getXML(R.xml.myfile)
```

XMLBitmap

- An ***XMLBitmap*** is an XML resource that points to a bitmap file.
  - Alias to the ***raw*** bitmap file
  - It specifies additional properties (tiling, gravity, ….)

```
<?xml version="1.0" encoding="utf-8"?>
<bitmap
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:src="@[package:]drawable/drawable_resource"
    android:antialias=["true" | "false"]
    android:dither=["true" | "false"]
    android:filter=["true" | "false"]
    android:gravity=["top" | "bottom" | "left" | "right" | …]
    android:mipMap=["true" | "false"]
    android:tileMode=["disabled" | "clamp" | "repeat" | "mirror"] />
```

Other resource types



Resource Type	File	Java constant	XML tag	Description
layout	Any file in the res/layout/	R.layout.<key>	<layout>	Defines a layout of the screen
animation	Any file in the res/animator/	R.animator.<key>	<animator>	Defines a property animation (not the only method!)
menu	Any file in the res/menu/	R.menu.<key>	<menu>	User-defined menus with multiple options

Accessing platform resources

- Android contains a number of standard resources, such as styles, themes, and layouts.
- To access these resource, qualify your resource reference with the android package name.

```
android.R.<resource_type>.<resource_name>
```

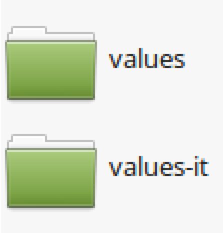
See: http://developer.android.com/reference/android/R.html

Resource selection

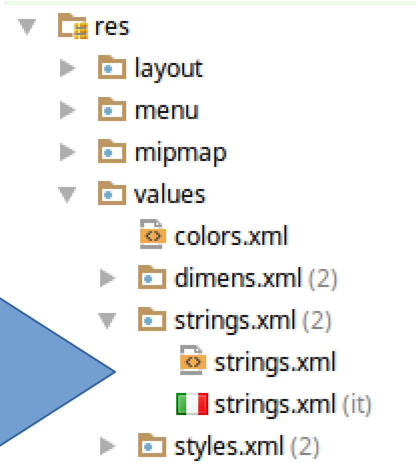
Resource selection

- Android applications might provide ***alternative*** resources to support specific device configurations (e.g. different languages).
- **At runtime**, Android detects the current device configuration and loads the appropriate resources for the application.
- Configuration-specific alternatives are specified as resource files in a directory in `res/` named:
  - `<resources_name>-<config_qualifier>`

File System
res/



```
values
values-it
```

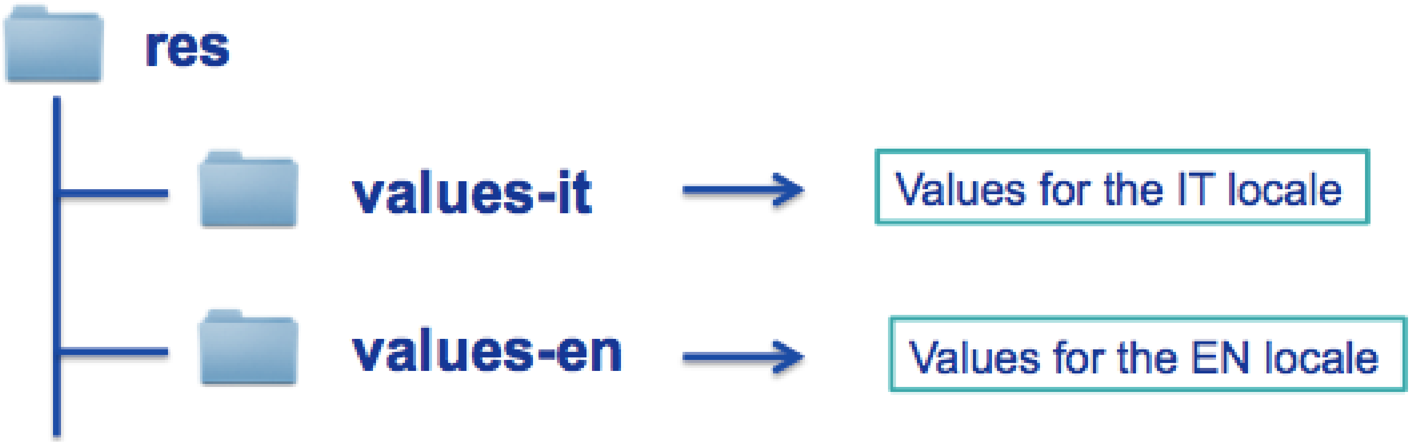


```
▼ res
  ▶ layout
  ▶ menu
  ▶ mipmap
  ▼ values
      colors.xml
    ▶ dimens.xml (2)
    ▼ strings.xml (2)
        strings.xml
        strings.xml (it)
    ▶ styles.xml (2)
```

Android Studio

Resources: alternatives specification

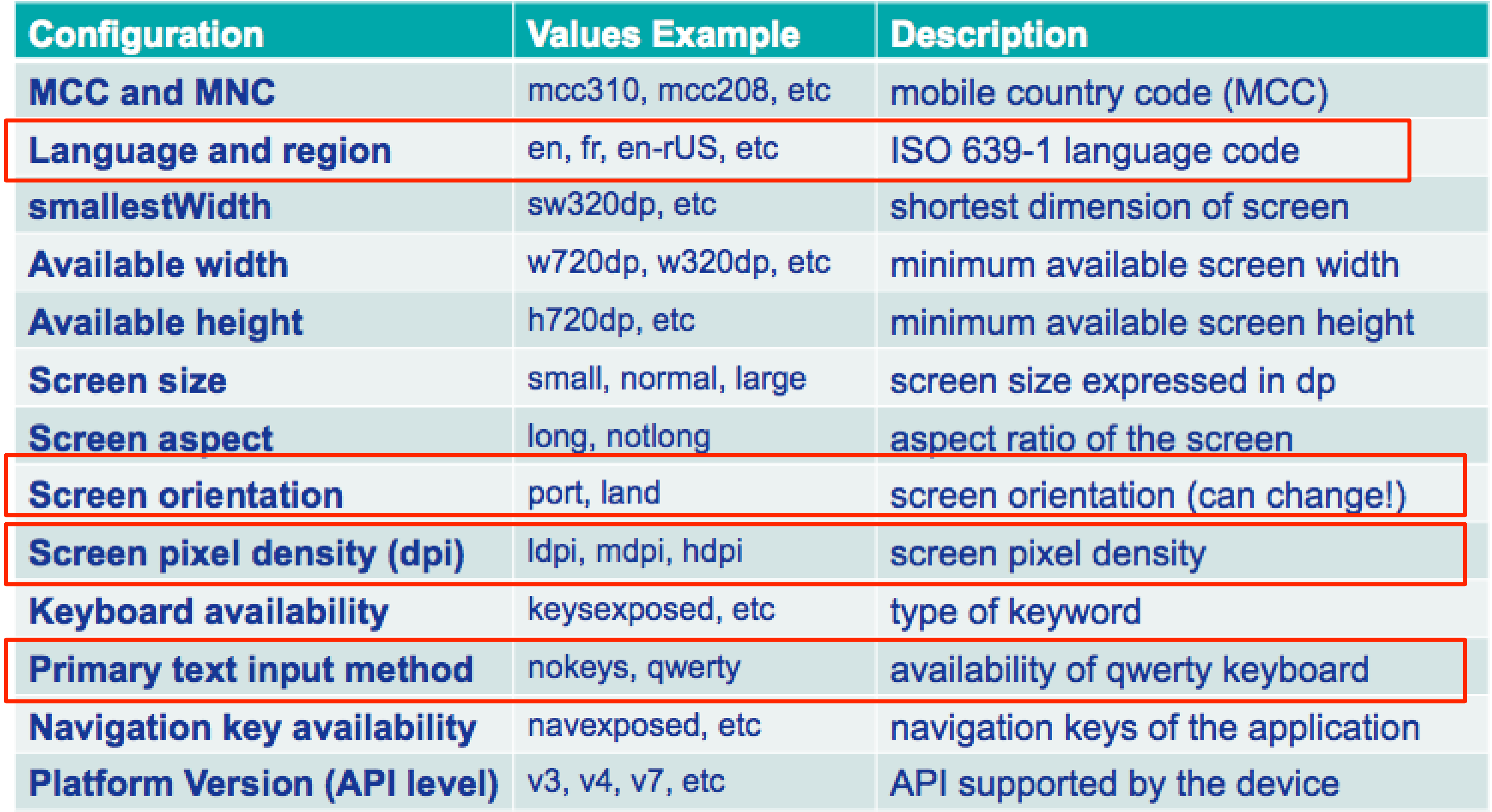
- `<resources_name>-<config_qualifier>`
  - `<resources_name>`: directory name of the corresponding default resource
  - `<qualifier>`: specifies an individual configuration



Values for the IT locale

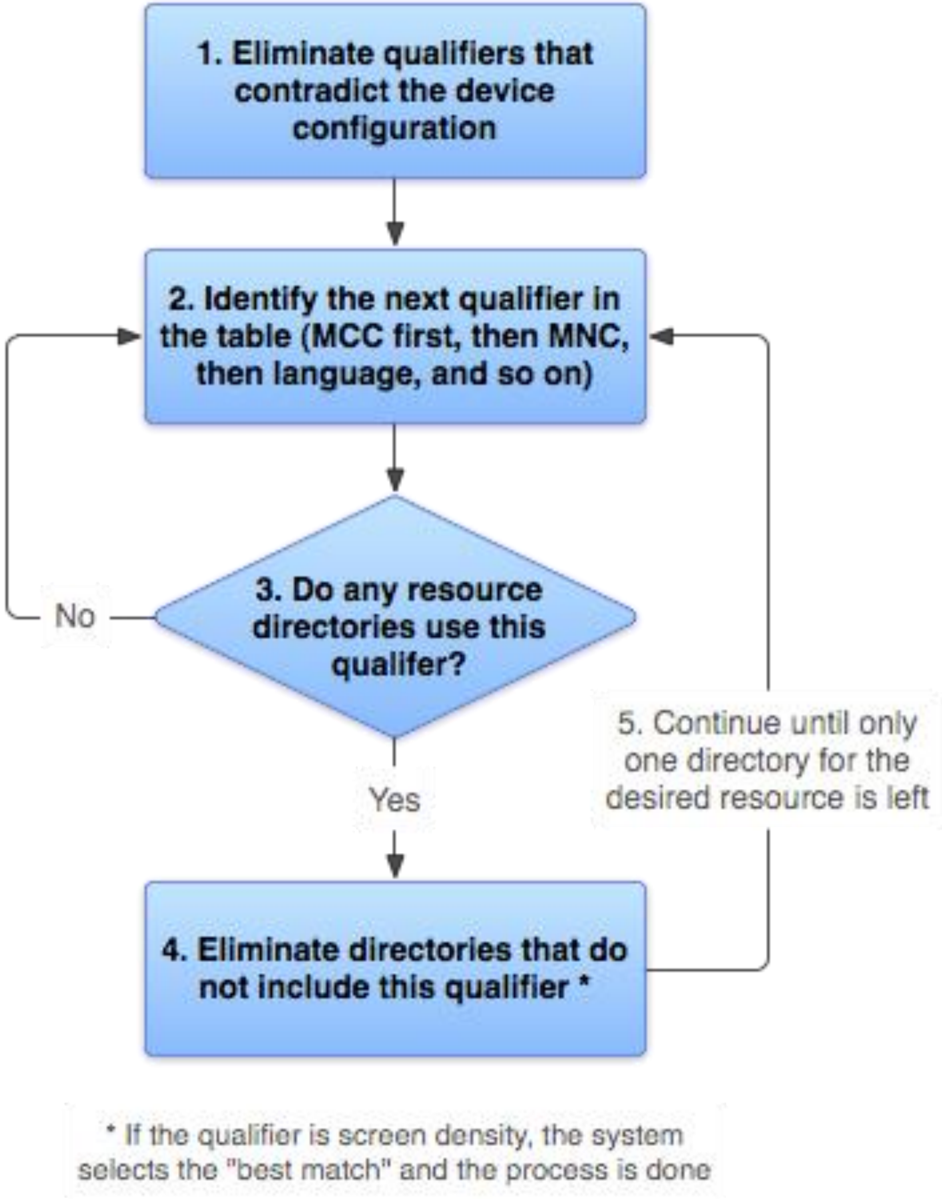
Values for the EN locale

Resources: alternatives specification - qualifiers



Configuration	Values Example	Description
MCC and MNC	mcc310, mcc208, etc	mobile country code (MCC)
Language and region	en, fr, en-rUS, etc	ISO 639-1 language code
smallestWidth	sw320dp, etc	shortest dimension of screen
Available width	w720dp, w320dp, etc	minimum available screen width
Available height	h720dp, etc	minimum available screen height
Screen size	small, normal, large	screen size expressed in dp
Screen aspect	long, notlong	aspect ratio of the screen
Screen orientation	port, land	screen orientation (can change!)
Screen pixel density (dpi)	ldpi, mdpi, hdpi	screen pixel density
Keyboard availability	keysexposed, etc	type of keyword
Primary text input method	nokeys, qwerty	availability of qwerty keyboard
Navigation key availability	navexposed, etc	navigation keys of the application
Platform Version (API level)	v3, v4, v7, etc	API supported by the device

Best-matching Resource


1. Eliminate qualifiers that contradict the device configuration

2. Identify the next qualifier in the table (MCC first, then MNC, then language, and so on)

3. Do any resource directories use this qualifer?

No

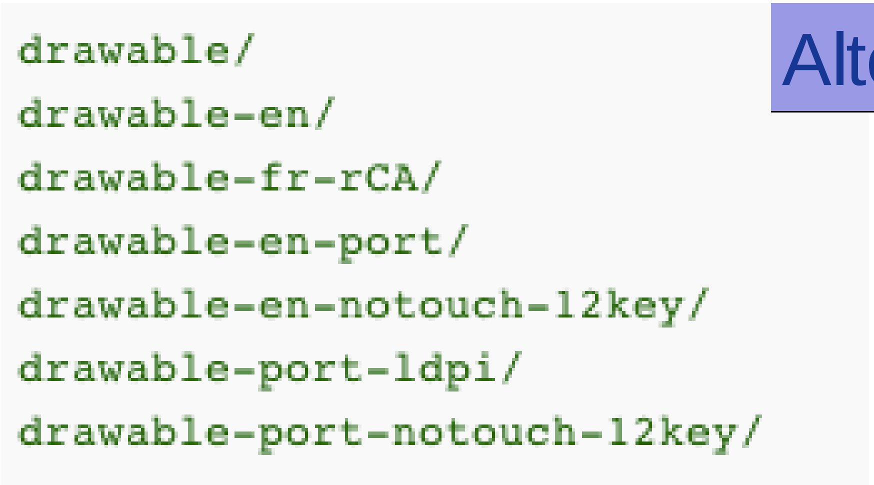
Yes

4. Eliminate directories that do not include this qualifier *

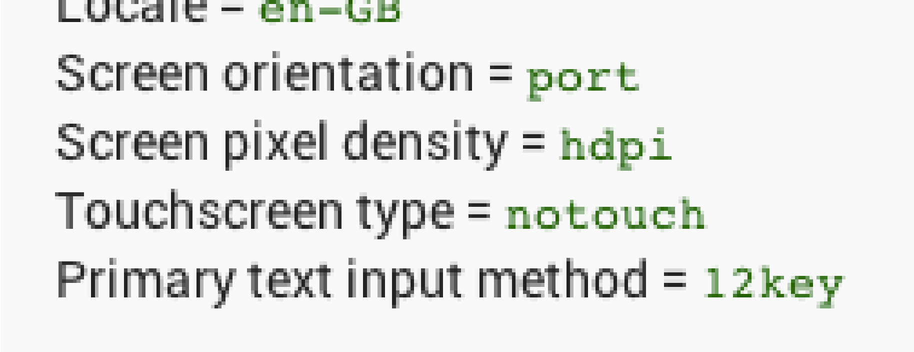
5. Continue until only one directory for the desired resource is left

* If the qualifier is screen density, the system selects the "best match" and the process is done

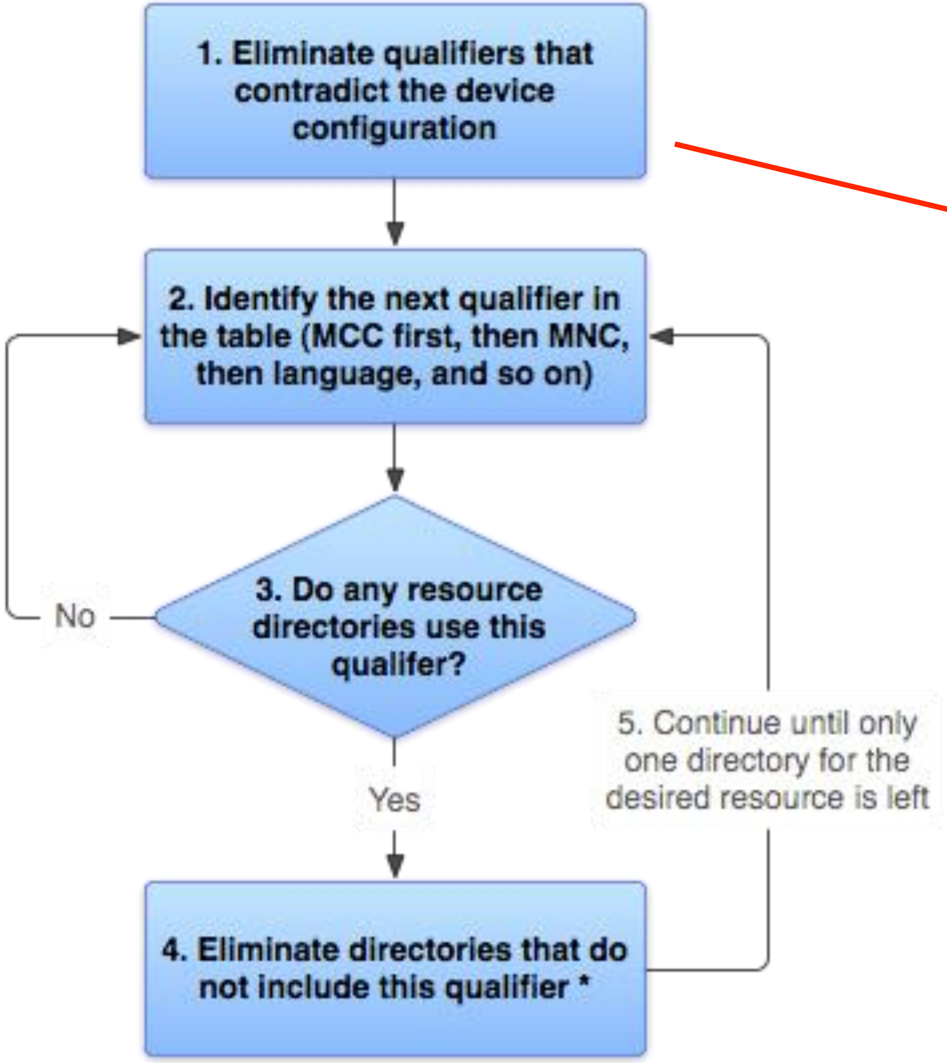
Alternatives


```
drawable/
drawable-en/
drawable-fr-rCA/
drawable-en-port/
drawable-en-notouch-12key/
drawable-port-ldpi/
drawable-port-notouch-12key/
```

Device configuration


Locale = `en-GB`
Screen orientation = `port`
Screen pixel density = `hdpi`
Touchscreen type = `notouch`
Primary text input method = `12key`

Best-matching Resource


1. Eliminate qualifiers that contradict the device configuration

2. Identify the next qualifier in the table (MCC first, then MNC, then language, and so on)

3. Do any resource directories use this qualifer?

No

Yes

4. Eliminate directories that do not include this qualifier *

5. Continue until only one directory for the desired resource is left

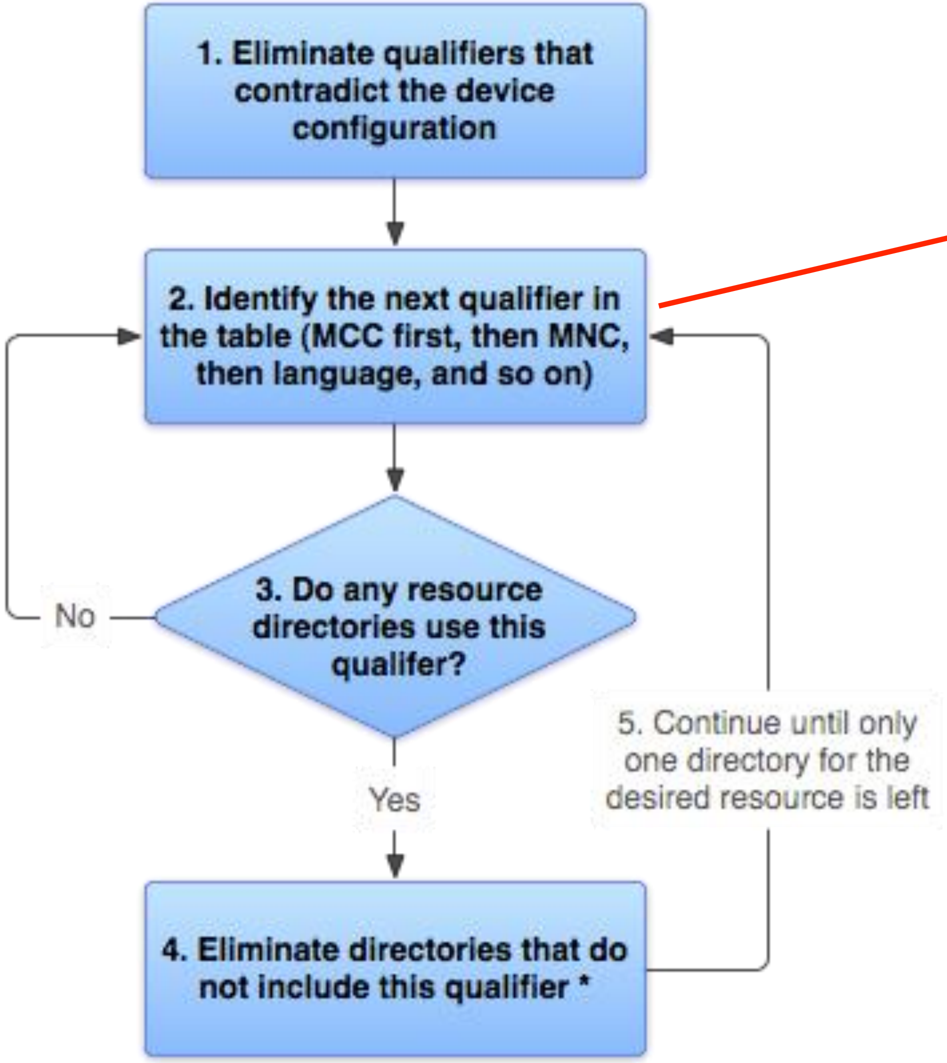
* If the qualifier is screen density, the system selects the "best match" and the process is done

```
drawable/
drawable-en/
~~drawable-fr-rCA/~~
drawable-en-port/
drawable-en-notouch-12key/
drawable-port-ldpi/
drawable-port-notouch-12key/
```

eliminated, because it contradicts the en-GB locale

Locale = `en-GB`
Screen orientation = `port`
Screen pixel density = `hdpi`
Touchscreen type = `notouch`
Primary text input method = `12key`

Best-matching Resource



1. Eliminate qualifiers that contradict the device configuration

2. Identify the next qualifier in the table (MCC first, then MNC, then language, and so on)

3. Do any resource directories use this qualifer?

No

Yes

4. Eliminate directories that do not include this qualifier *

5. Continue until only one directory for the desired resource is left

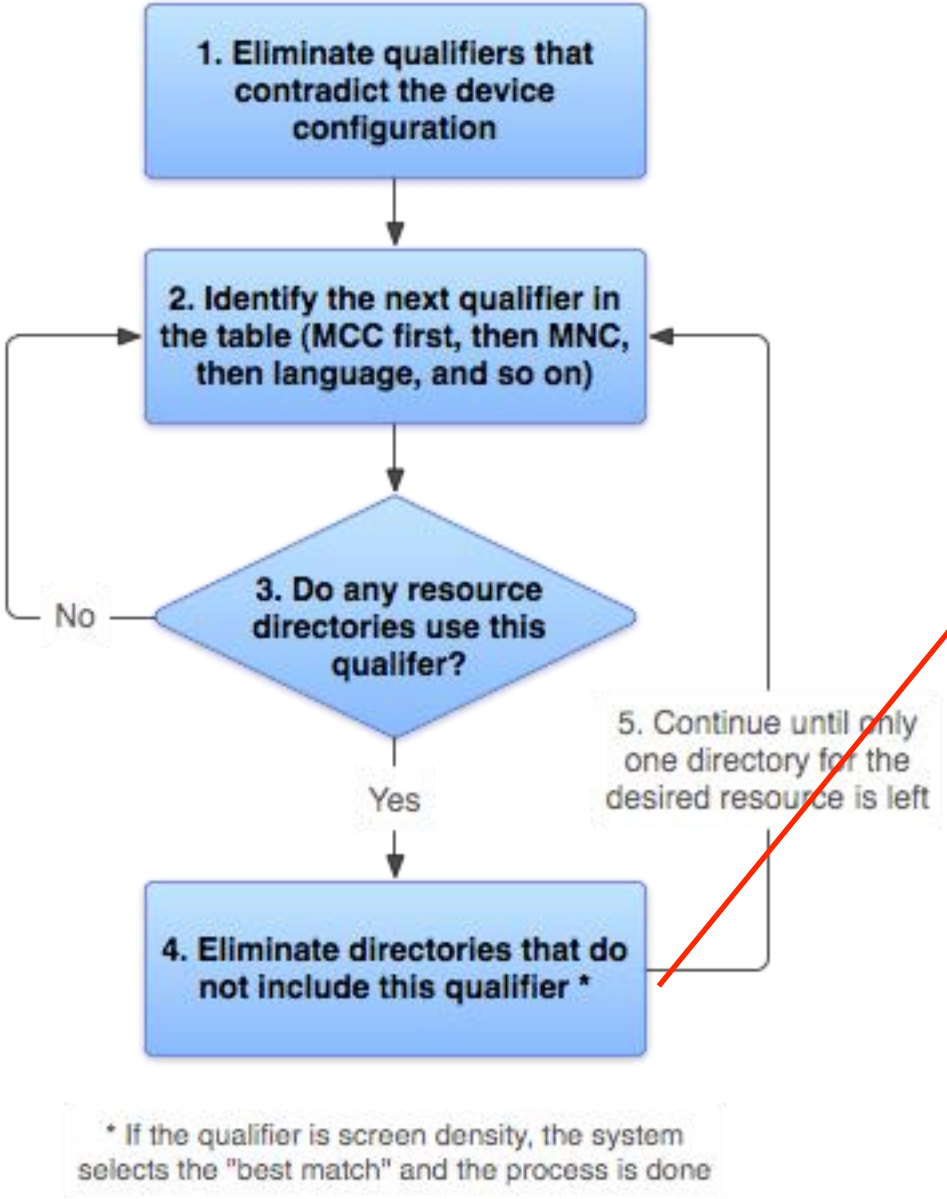
* If the qualifier is screen density, the system selects the "best match" and the process is done

Pick the (next) highest-precedence qualifier in the list (start with MCC, then move down)

```
drawable/
drawable-en/
~~drawable-fr-rCA/~~
drawable-en-port/
drawable-en-notouch-12key/
drawable-port-ldpi/
drawable-port-notouch-12key/
```

Locale = `en-GB`
Screen orientation = `port`
Screen pixel density = `hdpi`
Touchscreen type = `notouch`
Primary text input method = `12key`

Best-matching Resource



1. Eliminate qualifiers that contradict the device configuration

2. Identify the next qualifier in the table (MCC first, then MNC, then language, and so on)

3. Do any resource directories use this qualifer?

No

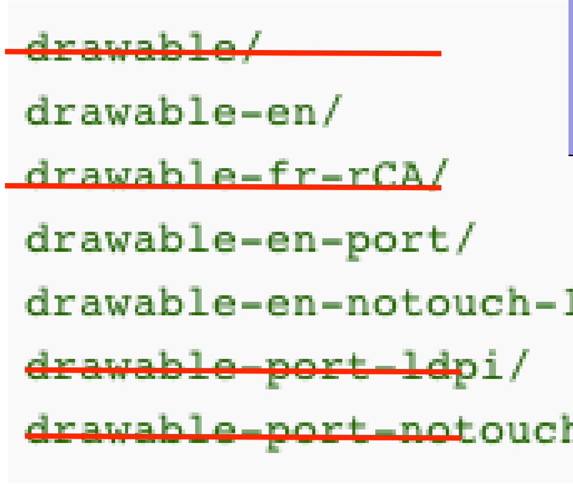
Yes

4. Eliminate directories that do not include this qualifier *

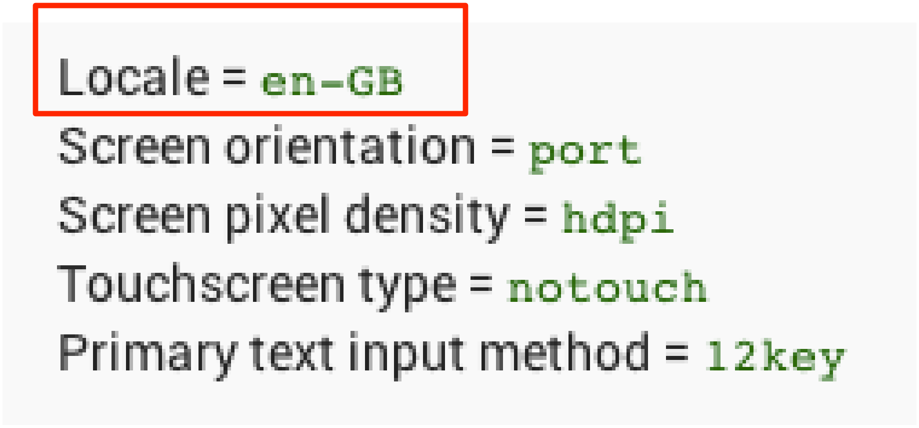
5. Continue until only one directory for the desired resource is left

* If the qualifier is screen density, the system selects the "best match" and the process is done

Selection on language qualifier

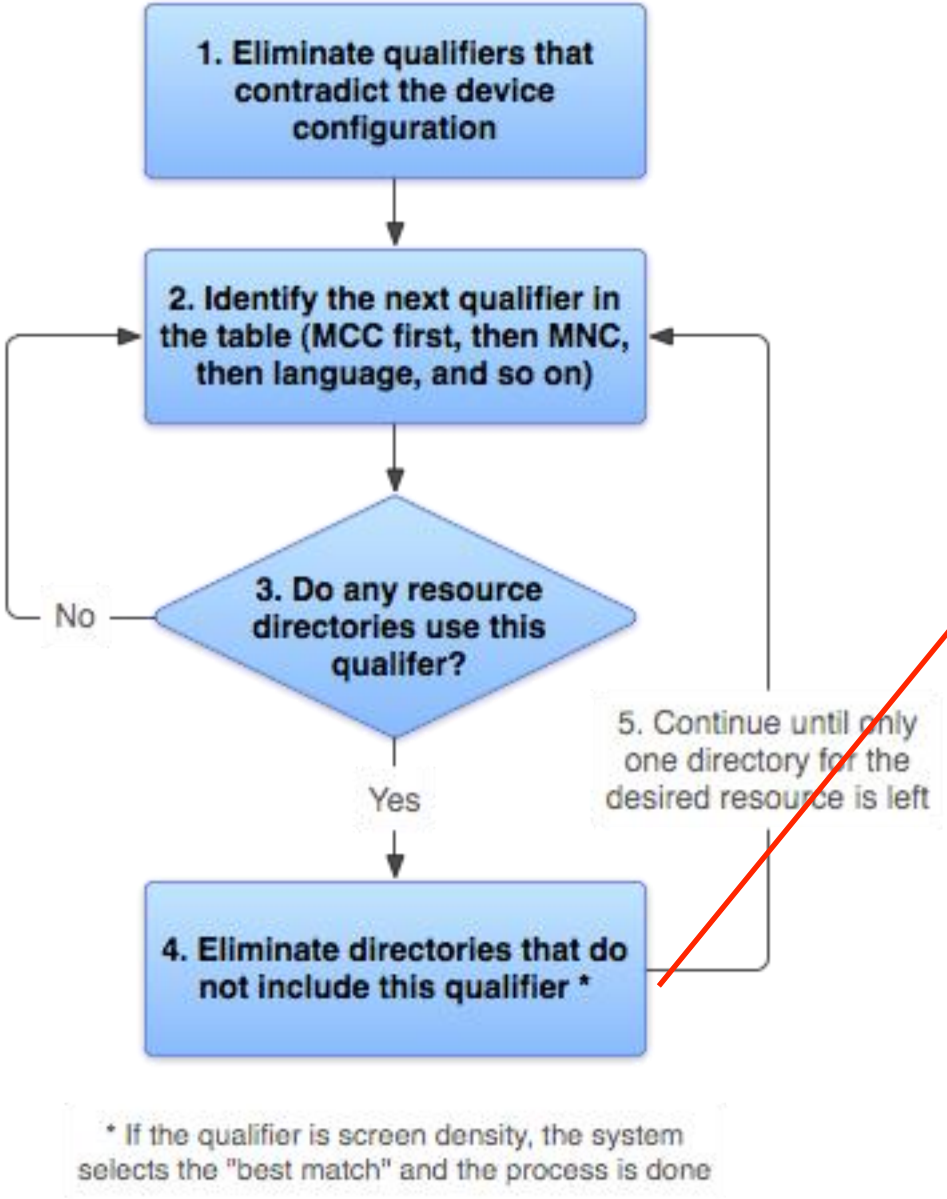


~~drawable/~~
drawable-en/
~~drawable-fr-rCA/~~
drawable-en-port/
drawable-en-notouch-12key/
~~drawable-port-ldpi/~~
~~drawable-port-notouch-12key/~~



Locale = en-GB
Screen orientation = port
Screen pixel density = hdpi
Touchscreen type = notouch
Primary text input method = 12key

Best-matching Resource



1. Eliminate qualifiers that contradict the device configuration

2. Identify the next qualifier in the table (MCC first, then MNC, then language, and so on)

3. Do any resource directories use this qualifer?

No

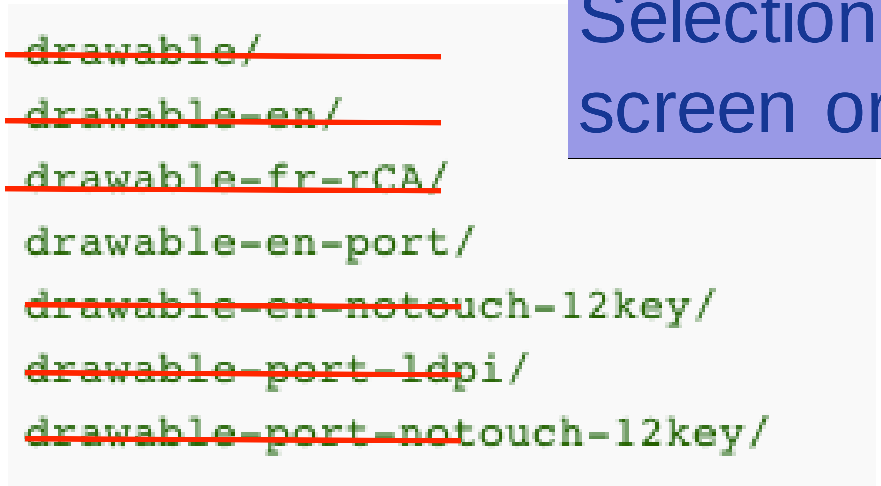
Yes

4. Eliminate directories that do not include this qualifier *

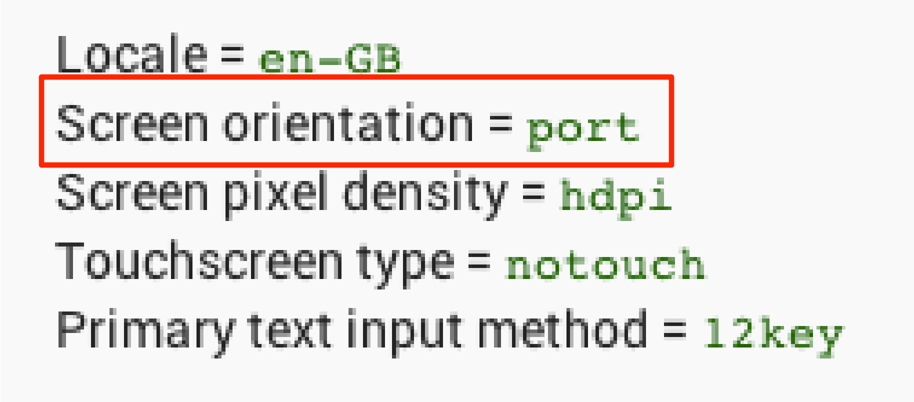
5. Continue until only one directory for the desired resource is left

* If the qualifier is screen density, the system selects the "best match" and the process is done

Selection on screen orientation



```
~~drawable/~~
~~drawable-en/~~
~~drawable-fr-rCA/~~
drawable-en-port/
~~drawable-en-notouch-12key/~~
~~drawable-port-ldpi/~~
~~drawable-port-notouch-12key/~~
```



Locale = `en-GB`
Screen orientation = `port`
Screen pixel density = `hdpi`
Touchscreen type = `notouch`
Primary text input method = `12key`

Handling Runtime Changes

Runtime configuration changes

- Android handles runtime changes to the language, location, and hardware by ***terminating and restarting*** the active Activity.
  - onDestroy() is called, followed by onCreate()
  - This forces the resource resolution for the Activity to be reevaluated.
- In some special cases this default behaviour may be inconvenient
  - e.g, applications that don’t want to present a different UI based on screen orientation changes.
- You can customize your application’s response to such changes by detecting and reacting to them yourself.

Managing configuration changes

- Add `android:configChanges` attribute to its manifest node, specifying the configuration changes you want to handle.
  - ***mcc and mnc***: a Sim has been detected and the mobile country or network code (respectively) has changed.
  - ***locale***: the user has changed the device's language settings.
  - ***keyboardHidden***: The keyboard, d-pad, or other input mechanism has been exposed or hidden.
  - ***keyboard***: the type of keyboard has changed; for example, the phone may have a 12-key keypad that flips out to reveal a full keyboard, or an external keyboard might have been plugged in.
  - fontScale — The user has changed the preferred font size.
  - uiMode — The global UI mode has changed. This typically occurs if you
  - switch between car mode, day or night mode, and so on.

Managing configuration changes

- ***fontScale***: the user has changed the preferred font size.
- ***uiMode***: the global UI mode has changed (day/night mode,…).
- ***orientation***: change between portrait and landscape.
- ***screenLayout***: the screen layout has changed; typically occurs if a different screen has been activated.
- ***screenSize***: (from Honeycomb MR2-API level 12) occurs when the available screen size has changed (e.g.: landscape/portrait).
- ***smallestScreenSize***: (from Honeycomb MR2-API level 12), occurs when the physical screen size has changed, such as when a device has been connected to an external display.

Manifest

manifest.xml

```
<activity
android:name=”.MyActivity” android:label=”@string/app_name”
android:configChanges=”screenSize|orientation|keyboardHidden”>
…
</activity>
```

pipe (OR)

Activity

```
@Override
public void onConfigurationChanged(Configuration newConfig) {
  super.onConfigurationChanged(newConfig);
  // [ ... Update any UI based on resource values ... ]
  if (newConfig.orientation == Configuration.ORIENTATION_LANDSCAPE) {…}
  if (newConfig.keyboardHidden == Configuration.KEYBOARDHIDDEN_NO) {…}
  …
}
```

When `onConfigurationChanged` is called, the Activity’s Resource variables have already been updated with the new values, so they’ll be safe to use.