How to support multiple screens using android?

Synopsis

Zealand Institute
of Business and Technology

Author:	Michal Derdak
Supervisor:	Anders Kristian Børjesson
Semester:	4th
Date:	20-5-2016

Content

Introduction

Operating system Android runs on many unequal devices, which have different screen sizes and densities. For applications, Android system offers consistent development environment and deals with most of the work to adapt every application's user interface to screen on which it is displayed. The system offers APIs that allow you to control your application's UI for specific device, in order to adjust your design for different screen arrangements - UI for tablets may differ from UI for handsets.

I have chosen this topic, because during my studies with Android I found tasks related to UI design or tasks like making good-looking application more challenging than actual work on code behind.

Problem definition

Today's variety of devices on market is huge. Many of those use Android system and making good-looking application which can adapt and be displayed properly on different screens is a huge benefit. For this problem, I came up with this set of questions, which I will try to answer.

Main question

- How to support multiple screens using android?

Sub-questions

- What is density-independent pixel (dp)?
- How to achieve density independence?
- Which types of screens are supported by Android system?
- How did Android changed its screen adaptability over the years?

Activities

Since I have 4 weeks for deadline I will divide my work weekly for planning/gathering information, working on main question, working on sub-questions, conclusion/reflection.

My first week, where I will try to gather as much as possible information, will contain endless browsing on Internet and also talking to professionals from Momondo company.

Second week will be dedicated to main question. This is most important week and precise work on this activity will lead me to answer, which will be reviewed in conclusion week.

Third week will be about sub-questions. I will try to divide work evenly between all of them and hopefully I will find answer to all of them.

In the fourth week I will evaluate my answers and work for all questions, summarize them and conclude them.

Planning

For proper division of work I made planning, where I sort all activities and estimated how much time each activity can possibly take.

- Gathering information
    - Searching for information on Internet
        - Estimated time - 3 days
    - Trying to contact professionals
        - Estimated time - 4 days
- Working on sub-questions
    - Sub-question “What is density-independent pixel (dp)?”
        - Estimated time - 1 day
    - Sub-question “How to achieve density independence? “
        - Estimated time - 3 day
    - Sub-question “Which types of screens are supported by Android System?”
        - Estimated time - 1 days
    - Sub-question “How did Android changed its screen adaptability over the years?”
        - Estimated time - 2 days
- Working on main question
    - Theoretical work
        - Estimated time - 2 days
    - Practical work
        - Estimated time - 5 days
- Conclusion / reflection
    - Write conclusion
        - Estimated time - 5 days
    - Write reflection
        - Estimated time 2 days

Sub-questions

In my synopsis, I chose to explain sub-questions before the main question. Sub-questions are going to be leading us to the main question and helping us with better understanding of it.

What is density-independent pixel (dp)?

To answer this question first we need to look at what density of the screen actually is. Screen density is a ratio of screen resolution and display size, this can be defined as dots per inch (dpi). The bigger the dpi, the smaller each pixel is, and screen looks better (more sharp). That means, a higher dpi screen can display more detail per inch, but this does not necessarily correspond with a higher screen resolution.

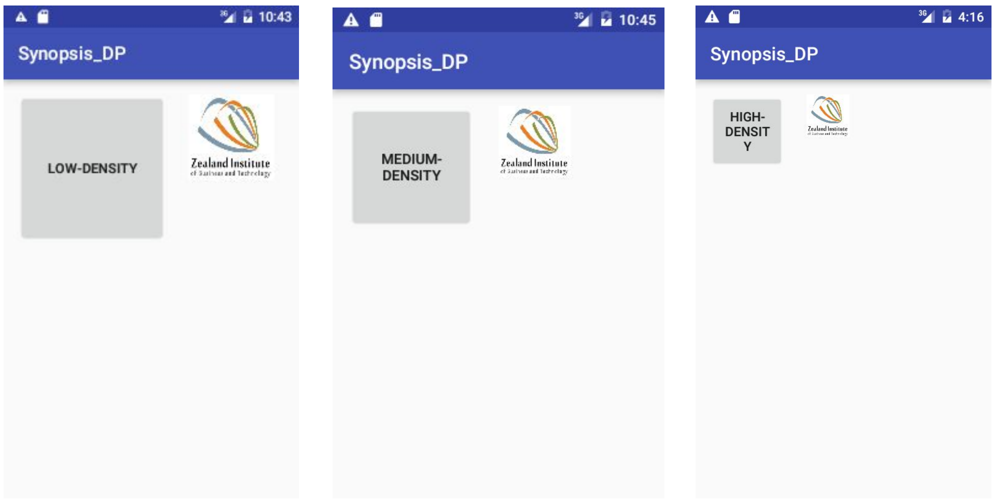
"For example, the Galaxy Nexus (4.65" diagonal) has a 720x1280 px resolution, while the Nexus 7 (7" diagonal) has an 800x1280 px resolution. It is a common misconception to assume that they have about the same screen density, since their resolutions are almost identical. However, the Galaxy Nexus has a screen density of about 316 dpi and the Nexus 7 has a screen density of 216 dpi, not even close." Steven Byle, Understanding Density Independence in Android, https://www.captechconsulting.com/blogs/understanding-density-independence-in-android

This is because while they have the same resolution, they also have different screen sizes, thus they are showing same amount of pixels but on different amount of space.

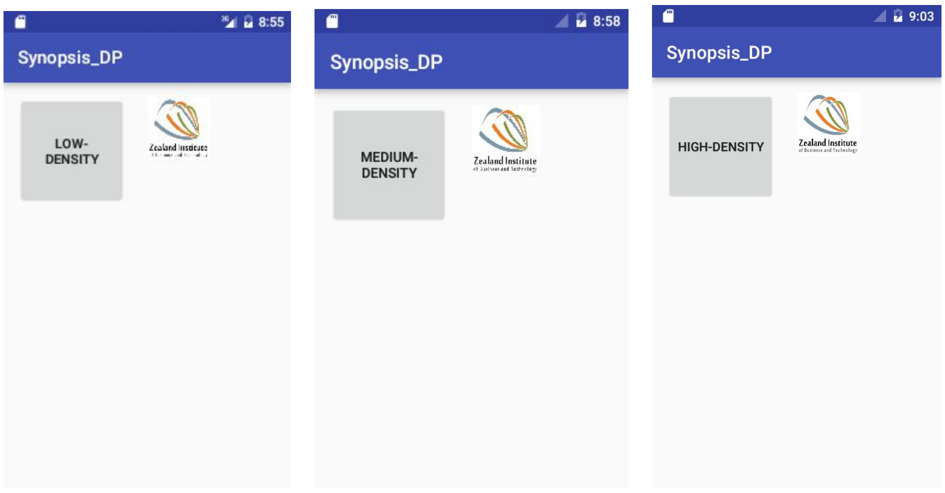
So, what is dp unit then? The density-independent pixel is equal to one physical pixel on a 160 dpi screen (medium-density) - this is baseline. Android system scale dp units, based on the actual density of running screen. From this we can calculate how many pixels are equal to certain amount of density-independent pixel unit on certain screen density using this equation: px = dp * (dpi / 160). That means, that 1 dp unit on 240 dpi screen would be equal to 1.5 physical pixels. To ensure your application has appropriate display of your UI on screens with different densities, you should always use density-independent pixel units when you are defining your application's UI elements.

How to achieve density independence?

In order to have best appearance for your application you should achieve "density independence". Without it, UI elements in your application may look physically bigger on low-density screens and smaller on high-density screens or there may occur any different shortcomings . See examples 1 and 2 below.



Example 1. On these images of applications we can see that without supporting density independence elements in UI appear smaller with higher density screens. Shown on the low, medium and high-density

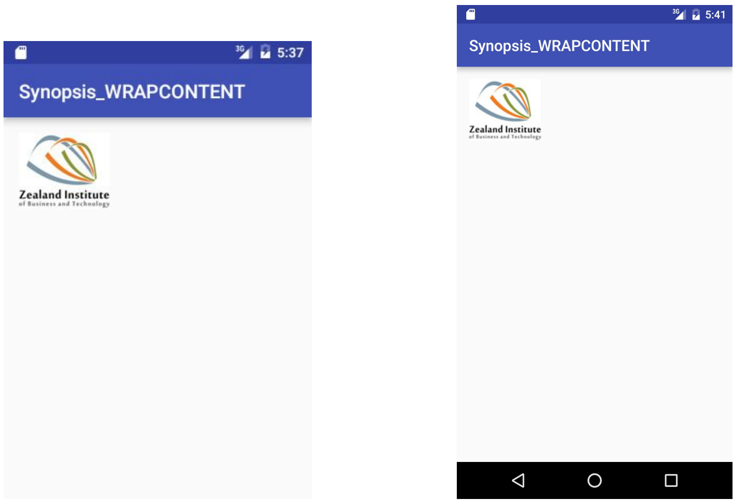


Example 2. This example shows us how application with a good support for different densities looks like. Shown on low, medium and high-density.

In screenshots in example 1, UI elements have dimensions defined in pixels (px). That means, when elements are shown on low-density screen they appear large even though they take same amount of pixels as in high-density screen, because high-density screen has more pixels per inch (higher concentration of pixel on same physical area). In example 2, the elements are defined in density-independent pixel (dp). As mentioned above, the baseline for dp units is one pixel on medium-density screens. Because of this elements in example 1 and 2 in the pictures showing medium-density have same sizes.

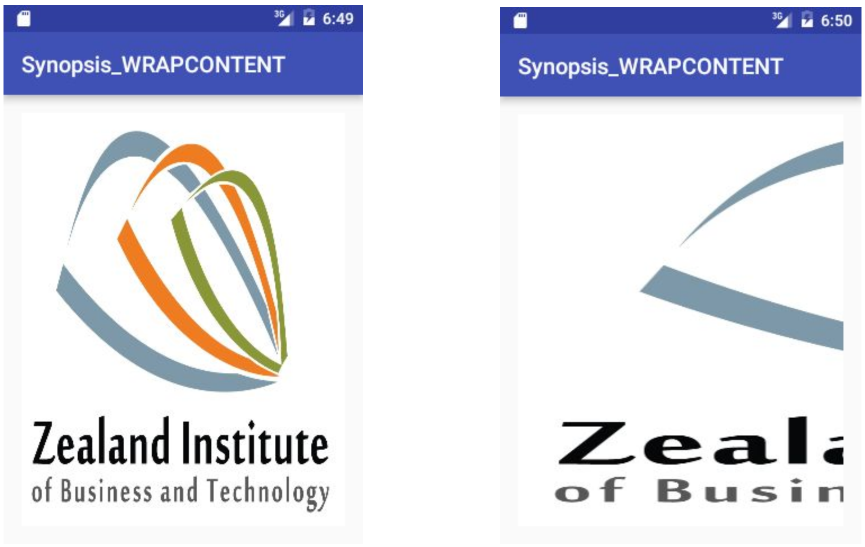
However using hardcoded values may not always be the best option for your application’s layout. You can achieve density independence by using “wrap_content”. “Wrap_content” will always adjust borders of the element to be just big enough to contain its content (pictures, text, etc.), children elements contained within it, plus padding, but that is not the only thing it does. “Wrap_content” will also scale your drawable file, so it does have the same physical size on every screen. For example when you define picture which is 100x100 px this will become the baseline and for other screen densities, your drawable file will be scaled in these ratios: ldpi - 0.75, mdpi - 1, hdpi - 1.5, xhdpi - 2, xxhdpi - 3, xxxhdpi -

4. That means when you have 100x100 px image, it will be scaled to 150x150 px when your application will run on hdpi screen. This may cause blurriness or other shortcomings, see example 3.



Example 3. Mdpi screen on the left, xxhdpi screen on the right. See full image sizes in appendix

When the content of the element is larger than actual screen size “wrap_content” will adjust element’s size to fit its parent so it cannot exceed it, this may cause funny stretching as you can see in example 4.



Example 4. On the left is image using “wrap_content”, on the right is image with hardcoded dp units,

However, scaling of bitmap drawables can sometimes lead to blurriness or pixelated bitmaps. To deal with this problem you should make sure that you provide alternative bitmap drawables for different screen densities, which we are going to talk about in next question.

Which types of screens are supported by Android system?

Android system started supporting different screen sizes and densities with version 1.6 (API level 4), by providing us with a configuration qualifiers. Configuration qualifier is a string that you can add to the resource folder in your Android project. You can provide alternative resources to the specific folder and, thus only those resources will be used for certain screen configuration.

To make things simple Android divided all screen sizes into set of four generalized sizes: small, normal, large and xlarge. However, starting Android 3.2 (API level 13), these groups were replaced by new technique for managing screen sizes based on the available screen width.

Starting with first generation of tablets in Android 3.0 the only proper way to define layout for them, was to put it into xlarge configuration qualifier. This brought a problem because 7 inch tablets were in the same group as 5 inch handsets. Rather than trying to fit and stretch layouts, Android developers decided to come up with a new technique. Instead of generalizing screens into groups, why wouldn't you be able to restrict your layout, so it can be used only on certain devices. This is done by specifying width and/or height available for your layout in dp units. For example if your layout is set to 600dp screens, that range becomes minimum screen size your layout can be displayed on.

Old configuration qualifiers for different screen configurations: small, normal (this is the baseline size), large, xlarge

New configuration qualifiers for different screen configurations.

- smallestWidth - sw<N>dp
  - Examples: sw600dp, sw720dp.
  - If you are planning to run your application layout only on screens with 600dp screen width or bigger, then you should use this qualifier to create the layout resources, res/layout-sw600dp/. Important difference between smallestWidth qualifier and others is that device's smallestWidth does not change when the screen's orientation changes.
- Available screen width - w<N>dp
  - Examples: w720dp, w1024dp.
  - Define minimum available width in dp units of the screen your layout can be run on. When the screen's orientation changes between landscape and portrait, the system changes corresponding value for the width. You can use this to specify the minimum width required for the layout, instead of using both the screen size and orientation qualifiers together.
- Available screen height - h<N>dp
  - Examples: h720dp, h1024dp
  - Define minimum available height in dp units of the screen your layout can be run on. When the screen's orientation changes between landscape and

portrait, the system changes corresponding value for the height. You can use this to specify the minimum height required for the layout, instead of using both the screen size and orientation qualifiers together.

Android system provides us with the basic six generalized densities. There are 2 other density characteristics, which are described in table below.

Densities	ldpi	Resources for low-density screens	120dpi
	mdpi	Resources for medium-density screens. This is the baseline density.	160dpi
	hdpi	Resources for high-density screens.	240dpi
	xhdpi	Resources for extra-high-density screens.	320dpi
	xxhdpi	Resources for extra-extra-high-density screens.	480dpi
	xxxhdpi	Resources for extra-extra-extra-high-density uses. This is used for the launcher icon only.	640dpi
	nodpi	Resources for all densities (density-independent resources). Not concerning current screen's density, the system does not scale resources tagged with this qualifier.	
	tvdpi	Resources for screens somewhere between mdpi and hdpi. This density group is mostly considered for televisions. If you will need to provide tvdpi resources, you should size them at a factor of 1.33*mdpi. For example a 100 x 100px image for mdpi screens should be 133 x 133px for tvdpi.	213dpi

How did Android changed its screen adaptability over the years?

Android's screen adaptability went through a lot of changes throughout the years, especially with advent of the tablets and large screen handsets. On the presentation you are going to hear a bit more about this topic, comparison of past, present and about possible future.

Main question

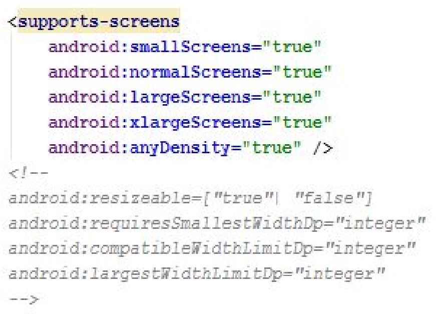
How to support multiple screens using Android?

Android system will always try to render application's layout and bitmap drawables in a suitable way for the ongoing screen configuration. Even though the system does most of the work for you to render your application's layout correctly on each screen configuration by resizing layouts and adapting them for different screen sizes and densities, it is not always appropriate, because your application may stretch in funny ways or some of your image files may be blurry and so on, therefore you should try following methods to improve your application adaptability.

I am going to demonstrate how to use following methods and make non-adaptive application into application, which can run and look good on multiple screens.

- **Declaring in the manifest which screen sizes your application supports**

Using manifest declarations, you can guarantee that only users with device screen your application support can download your application.



```
<supports-screens
    android:smallScreens="true"
    android:normalScreens="true"
    android:largeScreens="true"
    android:xlargeScreens="true"
    android:anyDensity="true" />
<!--
android:resizeable=["true"| "false"]
android:requiresSmallestWidthDp="integer"
android:compatibleWidthLimitDp="integer"
android:largestWidthLimitDp="integer"
-->
```

Example 5. <supports-screens> element declared in AndroidManifest file. Some attributes are commented out, because I won't need them in my examples

android:resizeable
- Defines whether your application is resizeable for different screen sizes. This attribute is deprecated and you should not use it.

android:smallScreens
- Defines whether the application can run on smaller screens. A small screen is defined as one with a smaller aspect ratio than the "normal" screen.

android:normalScreens
- Defines whether the application supports the "normal" screens. Normally this is a HVGA medium density screen, but WQVGA low density and WVGA high density are also in "normal" group.

android: largeScreens
- Indicates whether your application can be run on larger screens. A large screen is defined as a screen that is considerably bigger than a "normal" handset screen.

android:xlargeScreens
- Define wether the application supports extra large screens. An xlarge screen is defined as a screen that is larger that a "large" screen - for this group fall tablets (or something bigger).

android:anyDensity
- Define whether your application is provided with resources to adjust to any screen density.

android:requiresSmallestWidthDp

- Define smallest width required to run the application. The smallestWidth is the smallest, shortest line on the screen (in dp units), that must be available, so your application can be run on certain device.

android:compatibleWidthLimitDp

- Declare maximum (limit) of “smallest screen width” for device, on which your application can be displayed. If device has greater “smallest width”, user can still download application but he will have a option to use screen compatibility mode.

android:largestWidthLimitDp

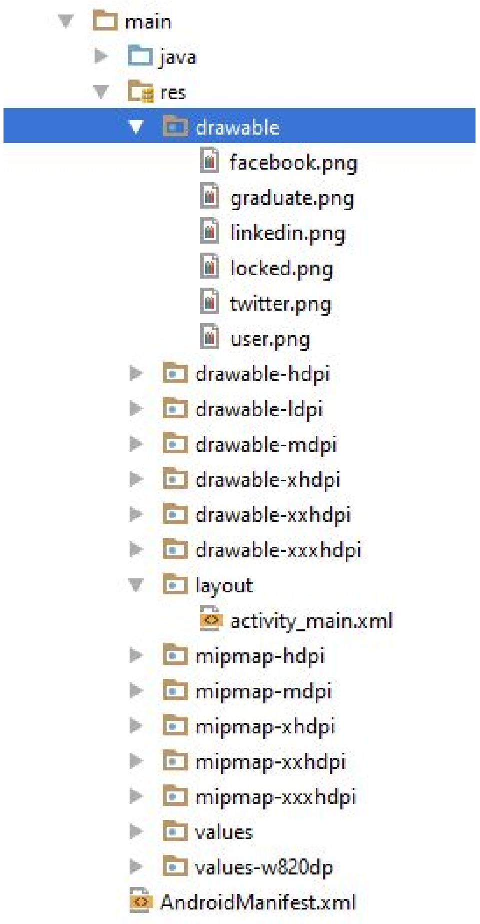
- Specify maximum (limit) of “smallest screen width” for device, on which your application can be displayed. If device has bigger “smallest width” user is forced to use screen compatibility mode.

What is screen compatibility mode? If you design your application for handsets only, it cannot be displayed on bigger screen such as tablets properly. Screen compatibility mode is workaround for this kind of problem. There are two versions of screen compatibility mode: one for Android 1.6-3.1, which is deprecated and we are not going to talk about it. The second one is for version Android 3.2 and greater. In version two Android system takes layout for normal handset (approximately 320dp x 480dp screen) and the scales it to up to fill the screen. The system basically zooms in on your application layout to make it bigger. This of course causes make shortcomings such as blurriness.

- **Offer variety of bitmap drawables for various screen densities**

Android system resizes your bitmap drawables (.png, .jpg, .gif. etc.) and renders them in the suitable physical size on every device. That means, if you provide bitmap drawable only for medium screen density (mdpi) which is baseline by default, the system enlarges them when they are used on high-density screens and reduces them when used on low-density screens. This resizing of bitmaps can cause blurriness or any other shortcomings. To make sure your bitmaps are at their best, you must include alternative versions of them with different resolutions for different screen densities. When your application is run Android system ensures that the best bitmap drawable is used for current screen.

Drawables provided in your application’s configuration qualifiers, which we already talked about in detail, are used to match current screen the best. If the device, which use your application has a high-density screen and your application use variety of drawables, the system will look for a drawable resource folder, which best matches the device configuration, in this case drawable-hdpi/.



```
▼ main
  ▶ java
  ▼ res
    ▼ drawable
        facebook.png
        graduate.png
        linkedin.png
        locked.png
        twitter.png
        user.png
    ▶ drawable-hdpi
    ▶ drawable-ldpi
    ▶ drawable-mdpi
    ▶ drawable-xhdpi
    ▶ drawable-xxhdpi
    ▶ drawable-xxxhdpi
    ▼ layout
        activity_main.xml
    ▶ mipmap-hdpi
    ▶ mipmap-mdpi
    ▶ mipmap-xhdpi
    ▶ mipmap-xxhdpi
    ▶ mipmap-xxxhdpi
    ▶ values
    ▶ values-w820dp
    AndroidManifest.xml
```

Example 6. Alternative drawable resource directories.

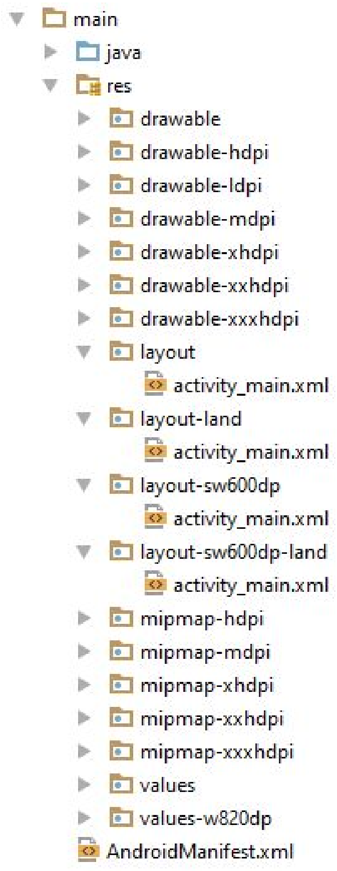
For my example I added alternative drawable file directories. Each directory has image with the same name as in default directory. Images are scaled according to ratios: ldpi - 0.75, mdpi - 1, hdpi - 1.5, xhdpi - 2, xxhdpi - 3, xxxhdpi - 4. These ratios are the same to the wrap_content scaling ratios.
At runtime, Android system makes sure that the best display on the certain screen is achieved by the following steps:

1. The system uses the appropriate alternative resource based on the size and density of the current screen.
2. When there is no alternative resource, the system will scale default file so it fits current screen size and density. Default files are in "drawable/" directory. However sometimes the system will not always use default file provided in "drawable/" directory, but instead will use different alternative resource to achieve best results.

- **Use different layouts for different screen sizes**

Android system tries its best to scale your application layout's elements to fit every screen nicely, but sometimes (let's be honest, every time) it is not enough. For example, for large screens you might want to you different position or maybe different size of elements in UI, for smaller screen you might use smaller sizes so everything fits in.

You can use the configuration qualifiers, which we already mentioned to provide particular layouts for each screen.



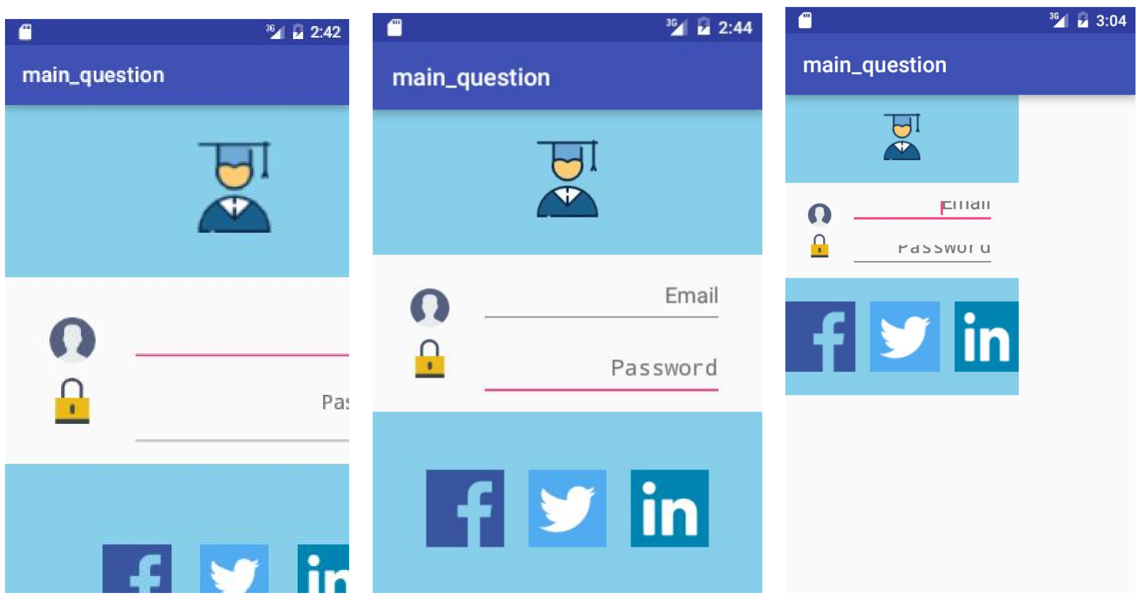
```
main
  java
  res
    drawable
    drawable-hdpi
    drawable-ldpi
    drawable-mdpi
    drawable-xhdpi
    drawable-xxhdpi
    drawable-xxxhdpi
    layout
      activity_main.xml
    layout-land
      activity_main.xml
    layout-sw600dp
      activity_main.xml
    layout-sw600dp-land
      activity_main.xml
    mipmap-hdpi
    mipmap-mdpi
    mipmap-xhdpi
    mipmap-xxhdpi
    mipmap-xxxhdpi
    values
    values-w820dp
  AndroidManifest.xml
```

Example 7. Multiple layout resources.

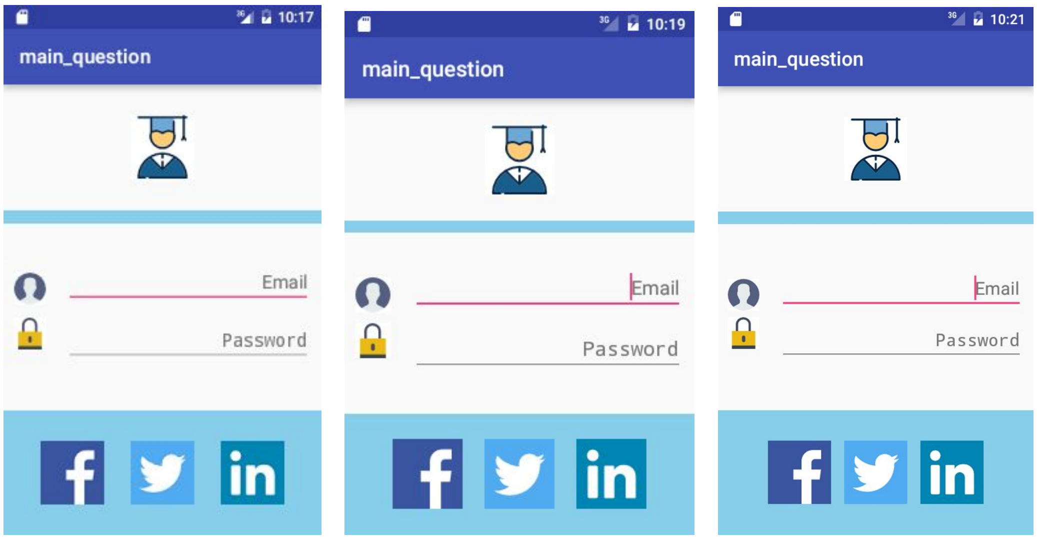
I have created layout for ordinary handsets which is included in layout directory. However UI design didn't look good on landscape mode so I decided to create landscape layout for handsets as well (layout-land directory).

For bigger screens, my example layout didn't look good as well thus I made smallestWidth configuration qualifier for big handsets and tablets. As mentioned above, screen with at least 600dp wide width is going to use layouts from layout-sw600dp and layout-sw600dp-land directories.

Using these simple methods we could went from this (example 8.) to this (example 9.)



Example 8. On the left picture is low-density screen, middle is medium-density screen, right is high density screen.



Example 9. Left is low-density screen, middle is medium-density screen, right is high-density screen

In example 8 where I used hard coded pixel values, layouts didn't look good on different screens except the middle layout (medium-density), which was designed at first. However after few simple steps we can be sure that our application is going to look great on every screen. For more screenshots and also screenshots of tablets please see appendix.

Conclusion

What is density-independent pixel?

Density-independent pixel is a unit you should use when you want to have fixed physical size of any kind of UI element across multiple screen densities. Dp unit is equal to one pixel on mdpi screen (160 dpi screen).

How to achieve density independence?

You can achieve density independence simply just by using dp or sp units, wrap_content or fill_parent/match_parent.

Which types of screens are supported by Android system?

Using various configuration qualifiers you can support range of screen from small screens with small densities to extra large screens with high densities.

How to support multiple screens using Android

Android system went through many changes and today using few simple methods your application can support almost any screen. In my opinion when you follow these three steps your application will be good to go for any screen you desire.

1. Don't use hard coded pixel values
2. Always use dp units, wrap_content or fill_parent/match_parent when you are defining your layout

3. Always use various types of drawable files for different screen densities

Today technologies provide really good adaptability for any platform - Windows, iOS, Android or even web pages. I believe that understanding and usage of these technologies at least in one of these platform can have huge benefit in understanding others more easily.

Reflection

When I look back on what I did and how I did it, I would surely choose different kind of plan or method, because sticking to estimated plan which has strict number of days for work to be completed was not really good solution for me, due to my unpredictable schedule.

I think writing this work was fun and I learnt a lot and I surely will continue increasing my knowledge in this area.

List of references

Android Developers *Guides, Screen support,*
http://developer.android.com/guide/practices/screens_support.html
Main source of knowledge.

Steven Byle *Understanding Density Independence in Android, December 27, 2013,*
https://www.captechconsulting.com/blogs/understanding-density-independence-in-android
Used this article for deeper understanding of achieving density independence in Android.