Mobile Applications Development - Coursework 1

Manuel Alaminos Dominguez
40333504@napier.ac.uk
Edinburgh Napier University - Mobile Applications Development(SET08114)

1 Introduction

This is the coursework of the Mobile Applications Development module. In this coursework, we had to develop an Android mobile application. During the Mobile Development module's practical classes, we were taught Android development with the Java programming language, but we were free to use the new Kotlin programming language, if we so desired.

I decided to work with Java as I was already familiar with it, and also because the workbook we were using in class was based on Java. It is also very similar to the C# programming language which I recently really enjoyed learning and working with.

I finally decided to develop a calculator mobile application, but it was not before numerous attempts with other types of application. I will talk about this later in the report.

Why did I decide to develop a calculator?

Simply, I thought it was a very good way of getting to know both the Java programming language and the Android mobile development framework. And it was so, due to the fact, a calculator requires many buttons to be pressed by the user. It also has to display various numbers and operations on screen. And last but definitely not the least, I had to program the calculation logic of the calculator with Java.

Another reason I should point out, is a calculator makes sense as a mobile application. What I mean by that is, some mobile applications could easily be replaced by a website, but a calculator is an application in which we need quick access to. Although, as time goes by, PWAs (Progressive Web Apps) may change the need to install many of the applications that we currently have on our mobile phones, calculators included.

After researching calculator applications user interfaces on different devices and operating systems, I concluded, I preferred the look and feel of the iOS implementation, which I could study thanks to my friend's iPhone. I liked its simplicity, and I wanted to apply part of its design to the application I was going to develop.

My next research step was to look for Android calculator examples on the Internet to get inspiration from. There were many examples to learn from. Some of these were:

- innovativenetsolutions
- javapapers
- thecrazyprogrammer

All the examples I found were developed using Relative or Linear layouts, but I decided against it when, as I was learning Android and checking the official documentation for references, I encountered this article which highlighted the good points of Constraint layouts. And I am glad I did.

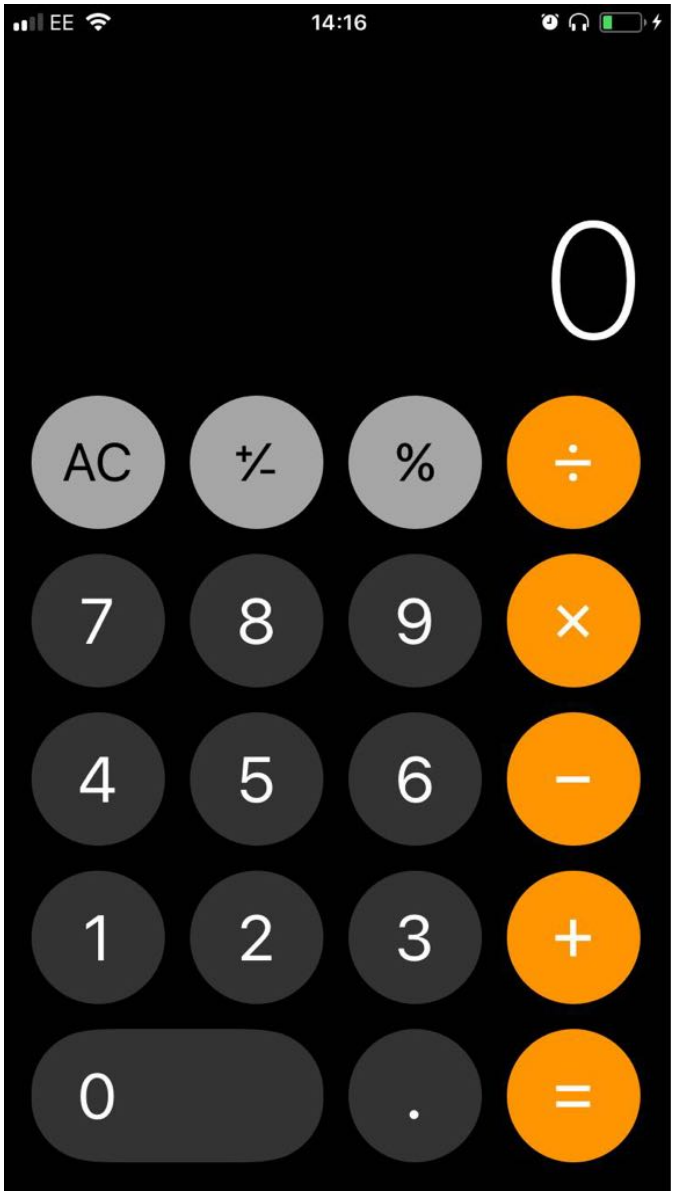

Figure 1: **iOS Calculator** - inspiration

After practising with Constraint layouts, I found it very intuitive to design with both constraint layout and GUI toolbox provided by Android Studio.

The official android documentation has been one of my most

frequented resources throughout the development phase, at least most of the time. At times however, I found it difficult to find some information I specifically needed and had to look for answers elsewhere, such as Stackoverflow.

2 Software Design

The chosen approach to develop the Android mobile application was the popular and well known Agile software development methodology.

After the researches mentioned in the Introduction section of this report, I started sketching with pencil and paper and drawing some basic wireframes.

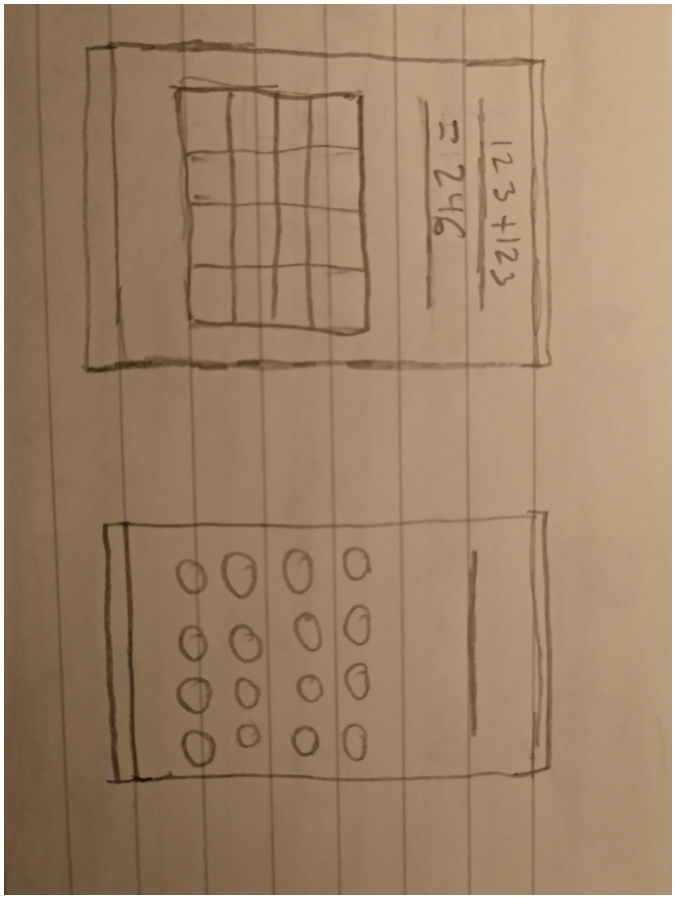

Figure 2: **App Wireframes** - with pencil and paper

I continued with this development approach, and I started implementing the User Interface, logic code with Java, testing every tiny added piece, and back again to add another little piece if I considered it necessary.

The application consists of one Main Activity Java file and two activity_main layout resource files. One for the interface in portrait mode and the second for landscape mode. I did so because I intended to design a slightly different layout for each of them and make use of the screen in a more efficient way.

As mentioned above, I decided to choose the Constraint type of layout, which allows different elements of the interface to be bound to other elements or to parts of the screen.

At the beginning of the development phase, I was not sure how the life cycle of the application was going to work, but as I advanced further, I started to have a better idea of what the life cycle of the application would be. See the image to see the final result.

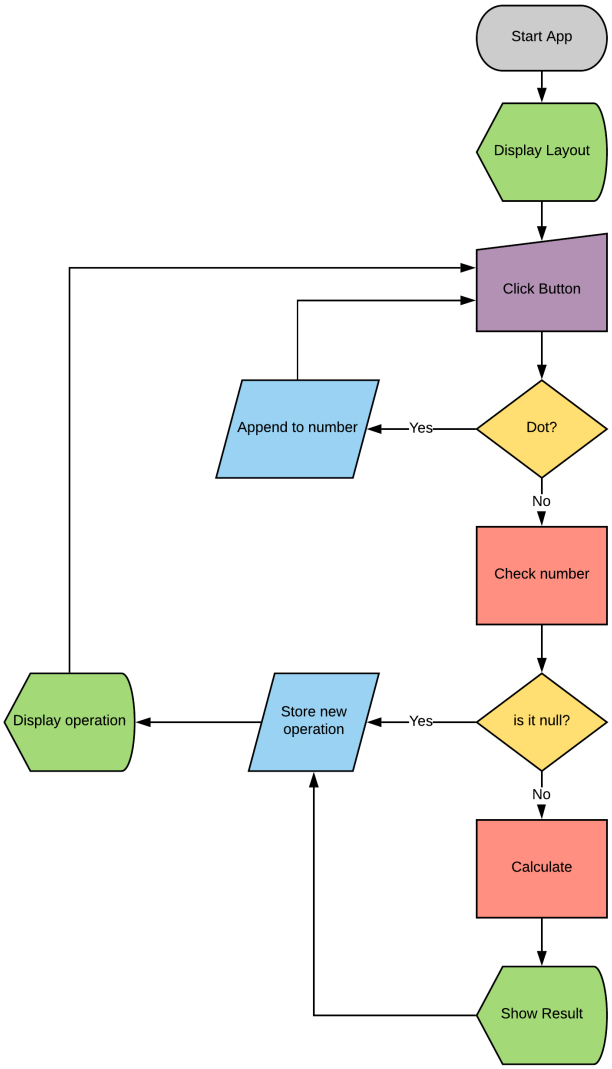

Figure 3: **App Life Cycle** - workflow

3 Implementation

With only one activity, the application is quite simple and easy to use.

I made the buttons and numbers significantly large to cater to a great range of users. By doing so, I made sure that young and old individuals could use the application, as well as people that cannot see small objects very well. It also helps people that have large fingers, as it will be easier for them to press the different buttons. The result is good accessibility overall.

The operation buttons have different colour to easily identify them. The accent colour used in these buttons are also used for the numbers displayed on top, which really make

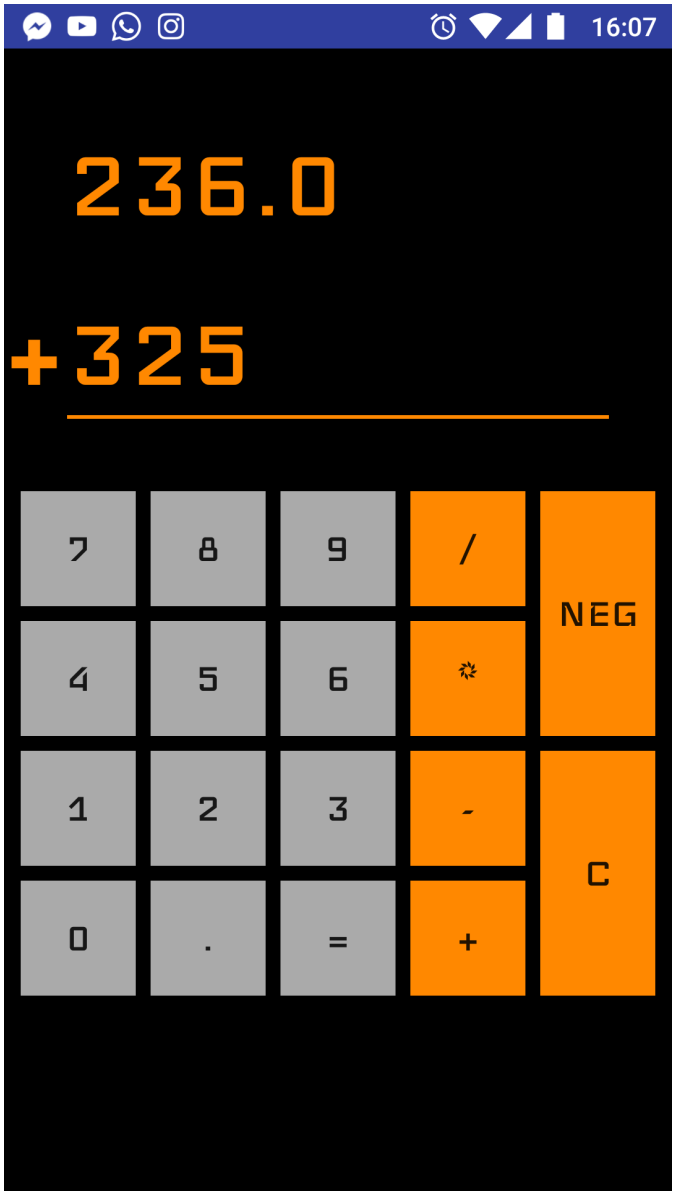

Figure 4: **Portrait Layout**

them stand out with the black background colour, making it surprisingly easy to view in most conditions.

Apart from the basic operations, the application also has a button to convert a number to a negative, another one to clean everything in screen and internal operations, and also the possibility to use decimals numbers thanks to the introduction of the dot key.

Code wise, I followed certain rules throughout the entirety of the development process:

- Variables on top. These being global or local variables, they would sit at the top of their respective scope. This was done to make sure that the person that is reading the code can easily identify where were these variables initialise and possibly assigned too.
- DRY (Don't Repeat Yourself). I reduced the amount of code as much as I could.
- Meaningful names to variables and methods.

Every button in the user interface had to be stored in a variable of type Button, like so:

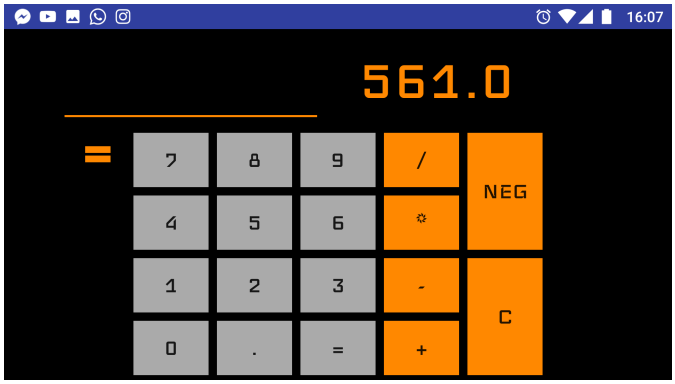

Figure 5: **Landscape Layout**

Listing 1: storing the UI buttons in Java

```
1  Button btn0 = findViewById(R.id.button0);
2
```

Likewise, I made use of a general method to listen to button clicks, like so:

Listing 2: listening to the user button actions

```
1  View.OnClickListener numListener = new View.↩
     OnClickListener() {
2      @Override
3      public void onClick(View v) {
4          Button btn = (Button) v;
5          newNumber.append(btn.getText().toString());
6  }
7
```

With this in place, I did not have to create a new event listener method for every button, but just pass this through every button, like so:

```
1  btn0.setOnClickListener(numListener);
2
```

The same was done with the operator buttons.

I also made use of Android methods to save and restore instances of the application. By doing so, the instance would not lose the current information when the application is being killed temporary when, for example, I rotate the device to a landscape screen orientation coming from the portrait mode.

The negative number was achievable thanks to simply multiplying by -1 the number that was selected by the user. If there is no number, it would just write a minus sign on screen.

4 Critical Evaluation

Due to the application simplicity, the final result does not distance itself much from what I first thought of doing.

Trying to differentiate my application design from the iOS one, I did not make the buttons rounded. I also chose the

closest font that reminded me to the typical physical calculator 'font' style, which also gave the application an unique and neat design, a mix of modern and retro, if that is even possible.

One feature I would have liked to develop is a scientific calculator in the landscape layout, the same functionality the iOS calculator app provides which was the one that inspired my design. But I quickly scratched this idea for being quite difficult unfortunately.

The scientific calculator is not the only thing lacking in this application. For instance, the Android calculator offers a history, in which you can see what you have typed previously.

I tried to implement this by creating a menu, but as I deleted the title bar, I could not make the three dots menu to show up and desisted in my idea of trying this after several tries.

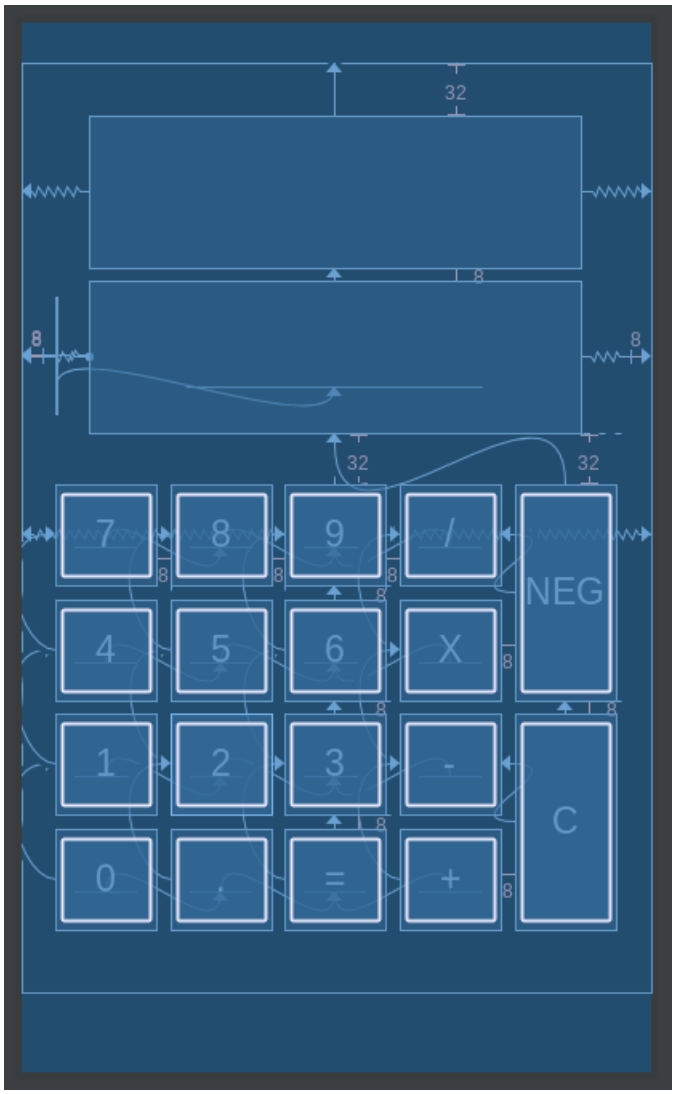

Figure 6: **App Blueprint** - constraint layout

Another feature I think would have improved the application is the possibility to choose light or dark theme. I tried to implement this feature within the menu I mentioned above, with a group menu in which a single option could be selected, and I even found the way to access the background of the constraint layout. However, I could not make it to work and I still could not make the menu to show up without the title bar.

On the positive side, the application looks clean and modern. It is also very easy to understand and use. Moreover, the application is quite polished and works flawlessly without any apparent issue.

5 Personal Evaluation

This has been my first adventure in mobile development, and so it was in Java software development as well. There were a lot of new things that I did not know about:

- Android application life cycle. Very confusing at first.
- Android framework. And all its classes and methods that makes development for Android much easier.
- Java programming language. Very similar to C#, although I prefer the common good practices for brackets in C#, as it is common to put them in a new line -looks more readable in my honest opinion-. Whereas in Java is similar to in JavaScript. And I understand this in JavaScript, as auto semi-colon can arise code issues when, for example, you return a literal object. But I think this does not apply to Java.
- Android Studio and all the help that comes with it. I absolutely loved Alt+Tab for, for example, extract strings. Very enjoyable to create getter, setter, constructor and methods by just pressing Alt+Insert (in Linux), it was surprisingly handy.
- Different type of layouts: Relative, Linear and Constraint.

However, there were many challenges I faced. Starting from deciding what application I should develop. I did actually start developing other applications before the calculator. Unfortunately, I encountered different issues and desisted in my idea to progress through those routes.

One example of this was a YouTube application, I was excited to use an external API. So, I learned how to set the YouTube API and set the Internet permissions to the application. Then, I realise that the YouTube API was quite limited and that I needed the YouTube Data API as well, which was quite big and difficult to know where to start to be able to create what I had in mind.

And, although the calculator application does not make use of external libraries, I am glad that at least I learned how to use them in the Android mobile development context.

Another challenge I faced was Android Studio itself, and more concretely, the Android Virtual Device (AVD). It was working without any issue for a few weeks, but suddenly it started giving me issues. Later on, I discovered that I had to 'wipe data' or 'cold boot' the AVD every time I opened Android Studio and ran my project for the first time. This issue was replicated by other student mates in different platforms. Overall, my impressions of Android Studio leave a lot to desired in regards of how polished it felt.

I have to admit that all these negative things mentioned affected my general motivation towards this coursework. If you add to this the arrival of Kotlin in the Android development scene, cross-platform mobile native application development such as Xamarin (C#), Ionic, React Native(React) or NativeScript(JavaScript or Angular) and even PWAs(Progressive Web Apps), developing app for Android in Java was getting less appealing, as the possibilities to use it outside of University were decreasing.

Having said all that, I am a positive individual after all and I tried my best to primarily focused on the good points of developing for Android with the Java programming language.

Developing with Java meant that I was practising a static typed language, which it would serve me well for C# and Angular with TypeScript in my future placement. Moreover, all the programming code used to solve the problems I encountered while developing the calculator is pretty much timeless and language agnostic. Not to mention all the knowledge acquired about Android development, which will stand in whatever language we decide to develop on Android.

Lastly, I feel very proud of how my application works flawlessly without any issue and that it looks clean and modern, at the same time that it aims to different audience, keeping accessibility in check. I also love how the code works and how clean and readable it looks, as I am always thinking about if other developers would be able to understand my code without much hassle.

Despite what I just said, I cannot hide feeling that I could have accomplished more, as the application is quite basic, it is not innovative, nor I have made use of external libraries or use of mobile hardware capabilities, which would have been very interesting to work on.