Jibran Syed
CSC 472 – Android Development

Final Project Discussion

Android API Usage

API usage in the final project varied between classes from the Android libraries and Java libraries.

There were certain suggested features from the Android API that were mentioned for use with the final project. The following is a list of Android features that were **not** used from the suggested features:

- Using Fragments to support different screen sizes
- Multi-touch gestures
- Accelerometer events
- Gyroscope events
- Location awareness

Of the suggested features, there was one that I did use: 2D graphics drawing. 2D graphics drawing was primarily used in two ways. The first was to draw the different Category icons on every screen. This was done using `ImageView` and getting the resource ID from `R.drawable.*`.

The second way was to load the circle image in the timer screen. That also used `ImageView`, but this time I loaded the image using Android's `Bitmap` and `BitmapFactory` classes. I learned a technique from the Android documentation to load the circle image at an optimized size so that it doesn't take forever to load the image. I made a utility class called `BitmapLoaderUtility` to take care of these details.

As far as using features not discussed in the lectures, the most significant ones are Android's `JsonReader` and `JsonWriter` classes. When used with Android's usual mechanism for saving data to file, these two classes took care of saving the persistent data made by my app. They are both used in my utility class `ChottJsonUtility` to save and load data from `LinkedList`'s into JSON files and vice-versa.

Additional important pieces of API are from Java as opposed to Android, and they are the `Date` and `SimpleDateFormat` classes. These classes were important to save the starting time and ending times of a Project session to file, and for formatting the time data into an easy-to-read time format, as demonstrated in the Project History screen.

Challenges

On of the biggest challenges when making the app was to figure out a clean way to get the serialization of data into a file working.

My app (called Chott) saves user data persistently in the device's internal storage and loads them when the app loads. Understanding how data serialization works in this case was really important, but figuring out when to get serialization to work was not the most intuitive thing to do.

My strategy to resolve this was first to assume that the app had no serialization. By first assuming that the data was somehow "magically" available in memory, I was able to implement a system to reference and make use of the user data in memory so that every part of the app would work properly. This usually implied making up "dummy" data in code, so I could see if my app would handle the data properly first.

Once my app worked great with data already loaded from memory, it was actually easy to fit-in an implementation to load that data from file instead. This entirely narrowed down the problem to just loading the data from file to memory, because I knew that the app, at that point, already worked with data loaded from code. Once I got the serialization part working, the app "just worked". So the take away from this is to make sure your data will work in the app logic first, before loading the data from file.

App Limitations

My app Chott is rather usable for its intended purpose, but of course, there are some important features missing. One of the most important missing features is deleting Projects, whether it's because the user made the Project on accident or if there is confidence that the Project will not be worked on anymore. Renaming Projects would also be useful, but will be tricky to implement because the Project data will have to be renamed in memory, JSON files, and in the Entries. Deleting Entries would also be useful if Entries were mistakenly created. Another feature would be allowing user correcting of Entry data (such as start time), if users forget to finish timing the current project session, for example.

Android SDK Limitations

As far as software engineering is concerned, the Android API isn't perfect, but it's certainly not bad either. It is at least easy to learn if you know where to look. There are some oddities though, such as using `int`'s or `String`'s to distinguish different API codes. Enumerations would have been much better in these situations; the Android API barely uses enumerations.

Another bad aspect of the API is the tendency to downcast very often. The classic example is getting a View object from the method `findViewById()`, that cast might be potentially unsafe if the programmer messes up the casting.

The reliance on super methods is also a little problematic in the API. There are certain life-cycle methods that use super methods, but some of these are required for the app to work, but could be easily removed by a careless programmer.

One last thing has little to do with the Android API itself, but more with Android Studio. On some occasions, the IDE will have internal exceptions that are rather mysterious to me and make the IDE feel badly tested. I've also had hard system crashes: on Windows, for example I often get the Blue Screen of Death when I load the Android Emulator, and on Mac, I suspect (from crash dumps) that Android Studio itself caused a Kernel Panic once.

Reflecting upon Android Development

Besides the strange bugs I just mentioned, the experience of learning Android was a little easier then I thought. Mobile development is definitely a very overwhelming ordeal, especially when you're new to it. However, for me personally, it helped that I already dealt with various programming problems before starting to learn Android.

Before this, I was learning iOS development a few years ago, and I was very overwhelmed by it. But this was probably because I wasn't programming very much back in those days. The more you program, the better you can program. I usually deal with a lot of game development coding in C++ and C#. I almost never program in Java, but knowing C# really makes jumping into Java easy.

These various experiences helped me learn Android a little more easily, because it allowed me to pick up things I was a little familiar with faster, and learn what I wanted to know about Android. There are a lot of things to learn in the Android SDK, but I don't worry about learning everything because that would be too overwhelming. I still don't understand Fragments, for example.

One of the main reasons I joined this class was to make the app that became my final project. To know that I can make simple little apps for my personal utilization feels very empowering.

I personally use an iPhone, but learning Android can be a useful experience (I have an ancient Android phone from 2010). I believe that if (or when) I go learn iOS development again, it will be much easier to learn then when I just started out. And of course, learning Android could be a very useful job skill.