Design

Introduction

The design of this project lies on two different key aspect. The first one is the direct JAVA code responsible for all of the application’s behavior and visual aspect. The second one is the database that incorporate an online component to this project in order to store the user’s information on a server instead of each and every device. The first part of the design will be focused on the JAVA local code and the second part will look at the database and its behavior.

The first thing to understand when dealing with android application is the core components of an application and how they interact with the Android environment and the JVM (Java Virtual Machine). Comparatively to a “Scene” in “JavaFx” or similar environments, Android uses an “Activity” in order for the user to see components of the app. Each page that the user will interact with is or extends an Activity. The super-class Activity is provided by Google and comes with specific methods that needs to be overridden in order for the android environment to communicate with the specified page. This instance of Activity follows a rigid lifecycle. At each point of this cycle, a method is called in order to achieve certain actions. Therefore, the class Activity can be extended in order to gain access to such methods. Using those, a given Activity can perform custom tasks.

Image 1 Activity’s lifecycle

In this application, every page the user will interact with is an instance of “AppCompatActivity”. Its behavior is similar to a typical Activity, however it contains certain fields and method in order to accommodate its code to older devices or android versions. In UML diagrams, the fact that every page extends this specific class have been omitted for clarity. Otherwise every UML class would be connected to this super-class and it would then be difficult to read the other relationships from other components of this project.

Another type of class is used in this project. Indeed, “Fragments” have also been used in this project. The purpose of a Fragment is to run alongside an activity, while providing a more dynamic behavior. For example, tabs in a “TabActivity” are instances of Fragments. They have a different, yet similar, lifecycle than an Activity (Image 2). Thus, interacting with them is noticeably more complex. Since they attach to an Activity, sharing the same “Context” object, it can become tricky to make it do what one might really want to do with it. Most of the methods on android require the Context as a parameter. Therefore, one must ensure that each of the fragments is properly attached. Also, when one wants to access a field of the parent Activity, within an attached Fragment, it becomes difficult to actually get the desired results.

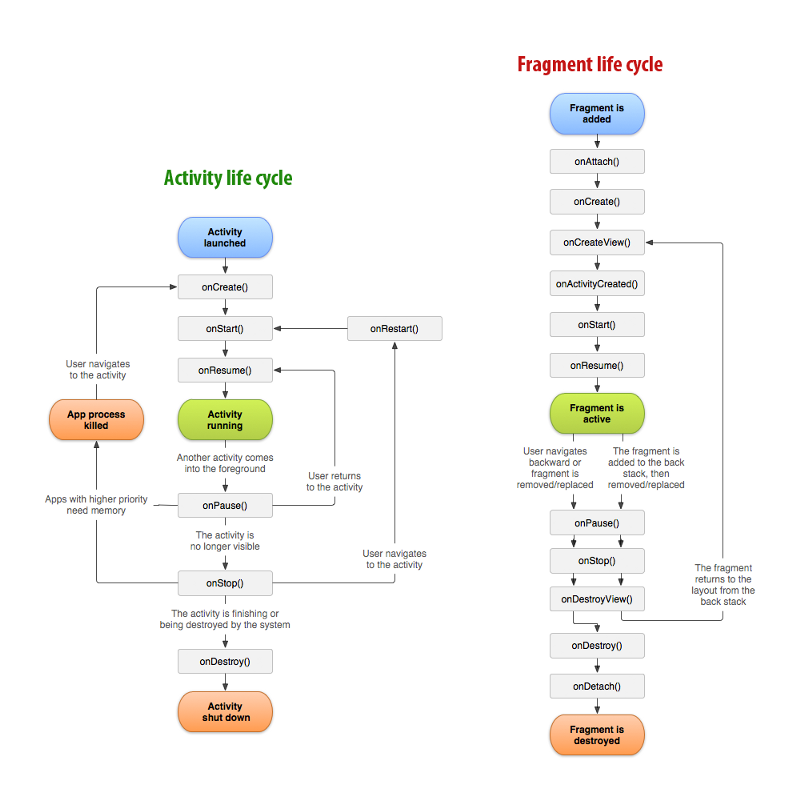
Moreover, the way of interacting inside a Fragment is completely different from a typical Activity. This lead to hours of frustration an incomprehension as one tried to properly understand the meaning behind such behavior. For instance, in an Activity, the visual layout is set with simple method in the onCreate() method. However, the layout is inflated in a Fragment, which requires a completely different approach. Since this method sends an object of type View when inflating, one must keep this view as a field in the current class in order to be able to interact with the desired visual element. On the other hand, the fact that the lifecycle is different and more complex gives a broader range of action to the developer in order to achieve certain actions. Since it is a more dynamic object, it gives the ability to program certain actions in very particular context. For example, when a tab is changed, what should the Fragment do? This is something that does not have to be dealt with while working with Activities.

Image 2 Fragment lifecycle

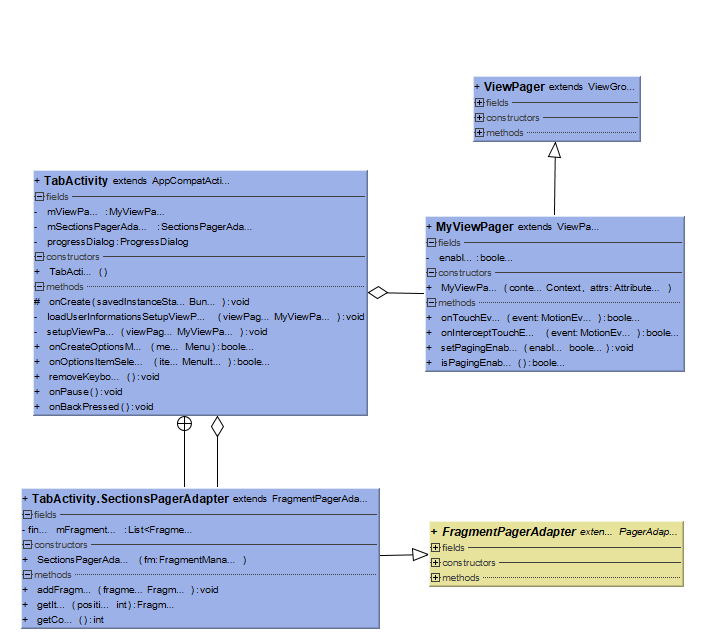
UML

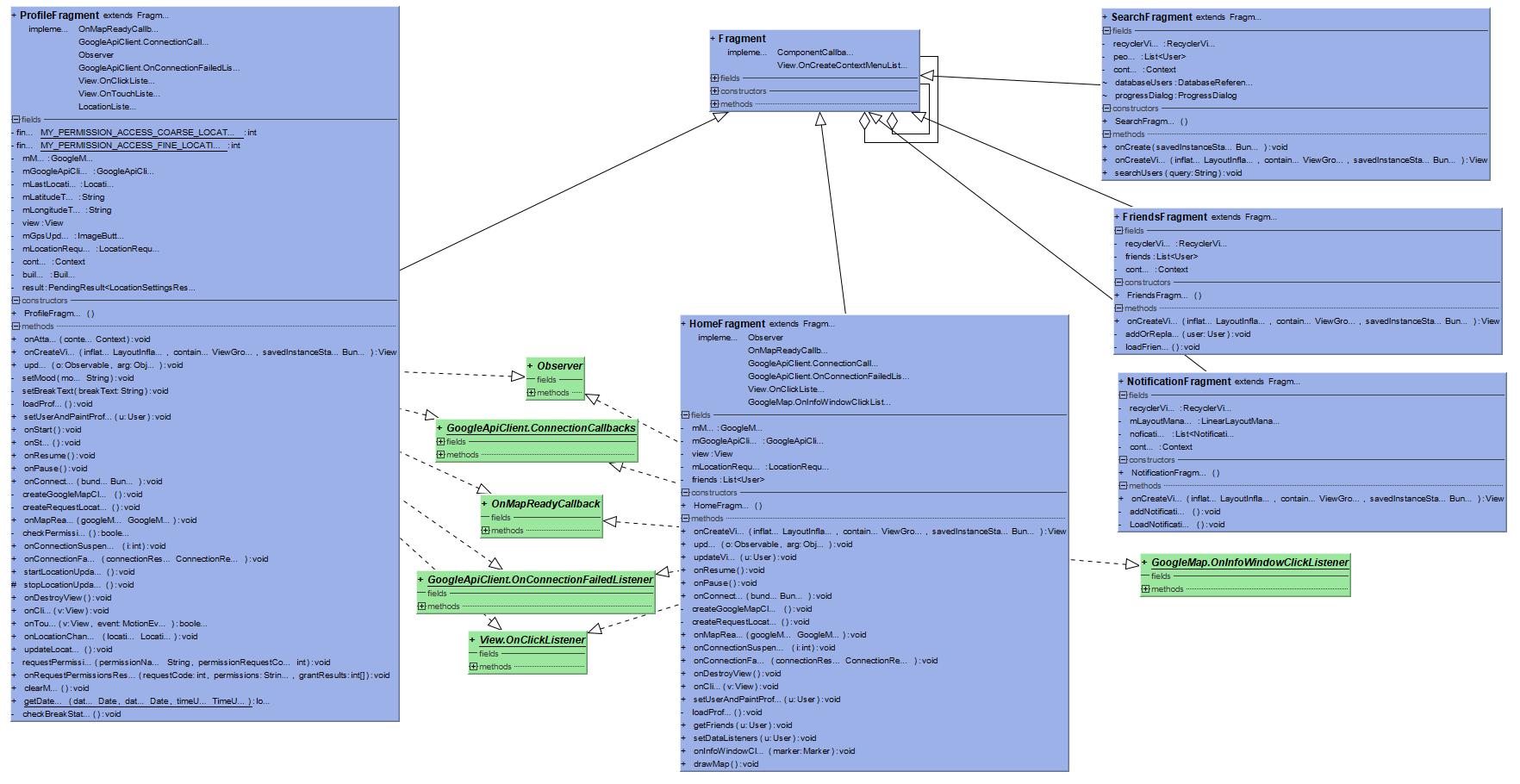
The UML diagrams have been separated in distinct categories for clarity. Since each class have different super-classes and properties, they will be regrouped in different blocks. Thus, the Fragments and their respective Adapters and ViewHolders will be grouped together, the Database models will share the same section and the Activities will also have their own part. It is also important to realise that Fragments and Activities do not have any kind of relationship together. They exist in the same application but rarely interact with each other. As explained in the previous part, some of the super-classes have been omitted for a reason of clarity. Only the important relationships are shown.

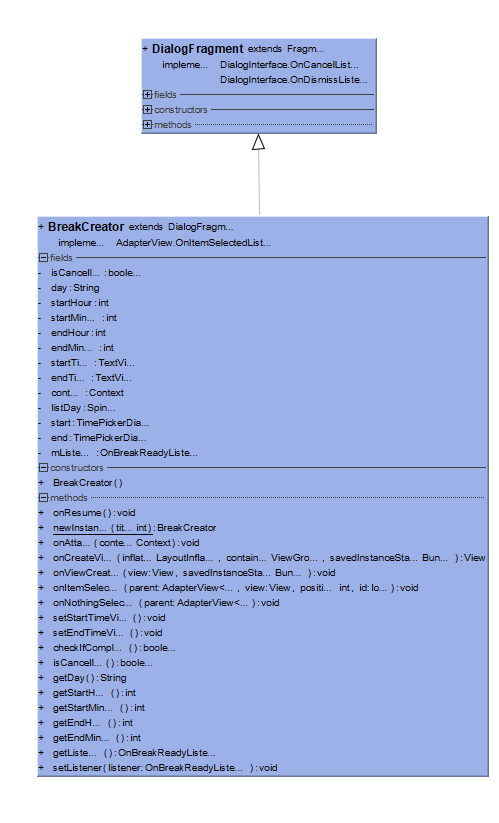
Activities

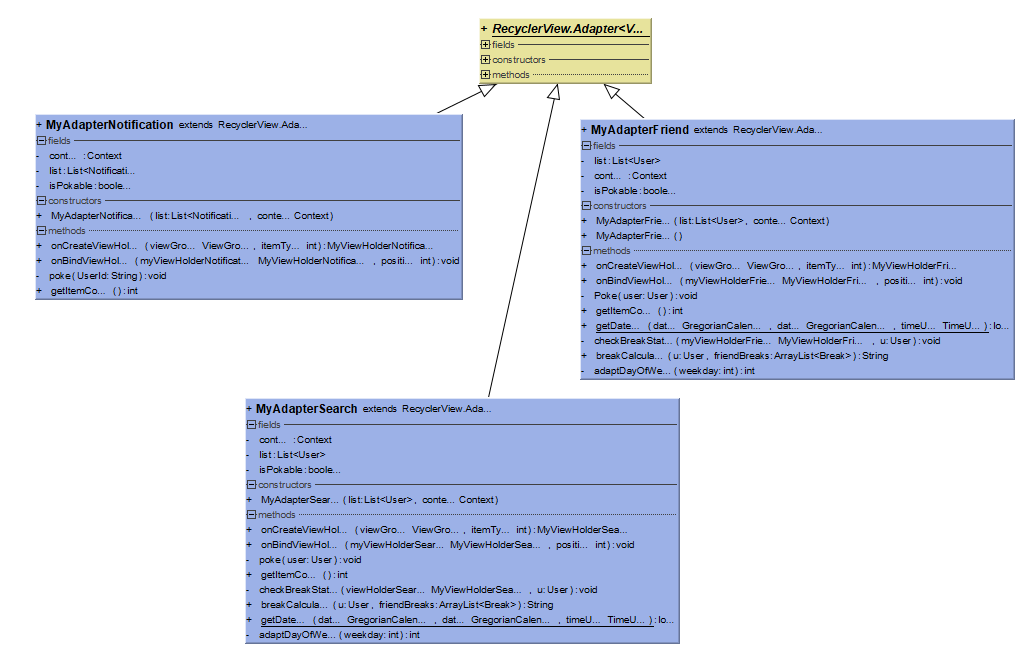
Note: There is no relationship between them.

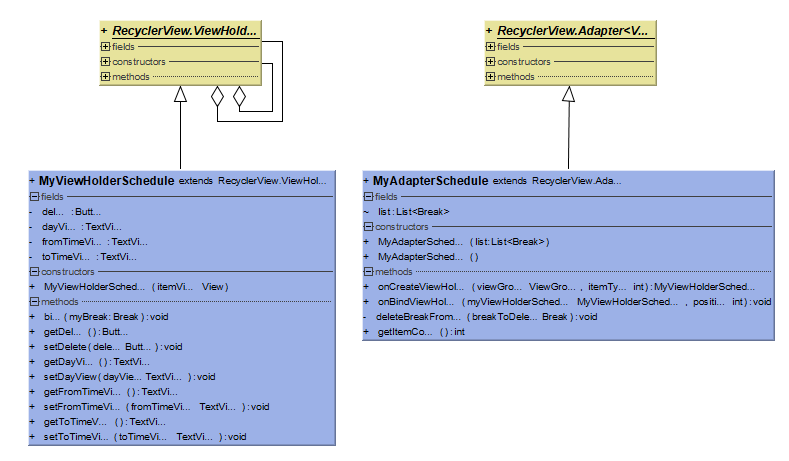


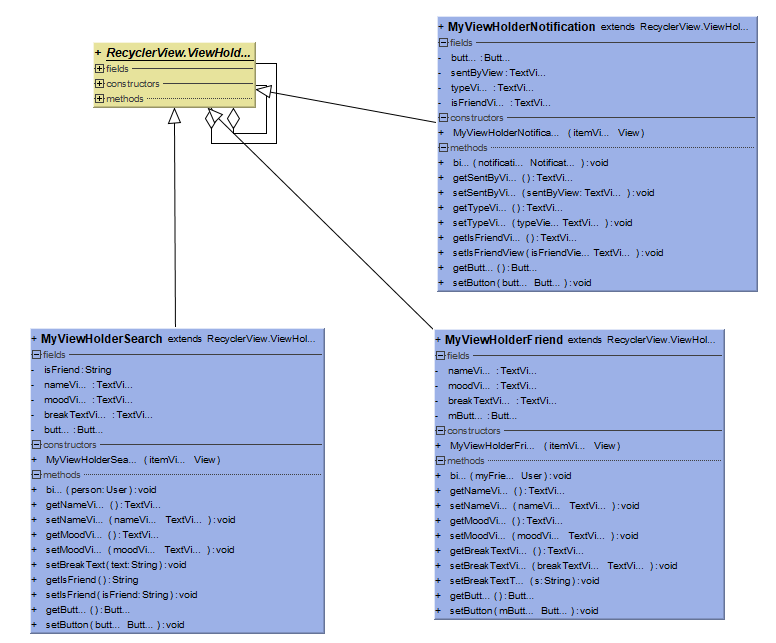


Fragments

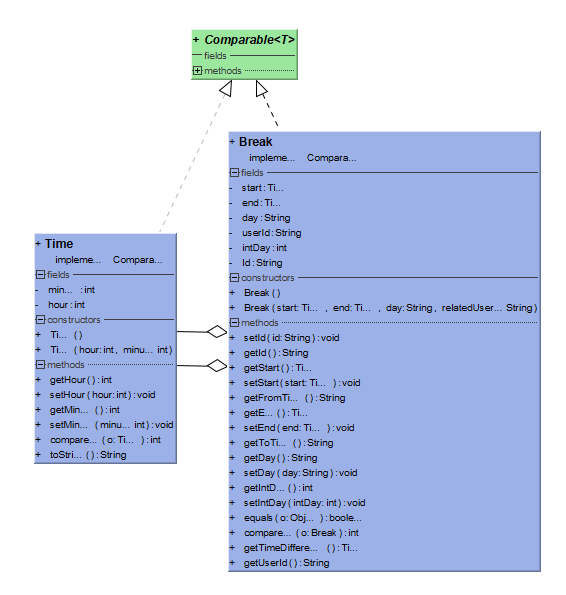


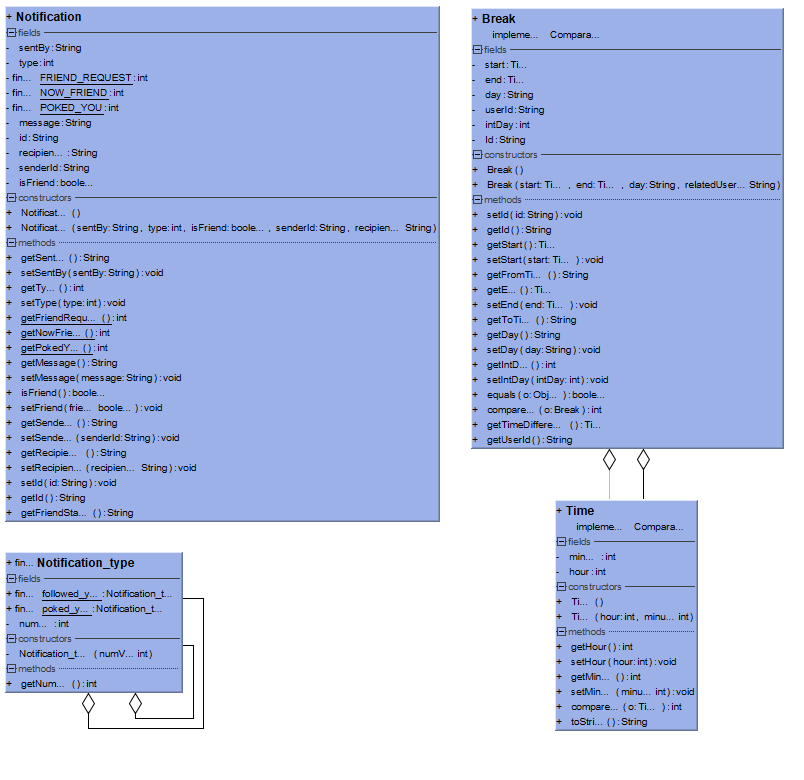
Adapters and View Holders

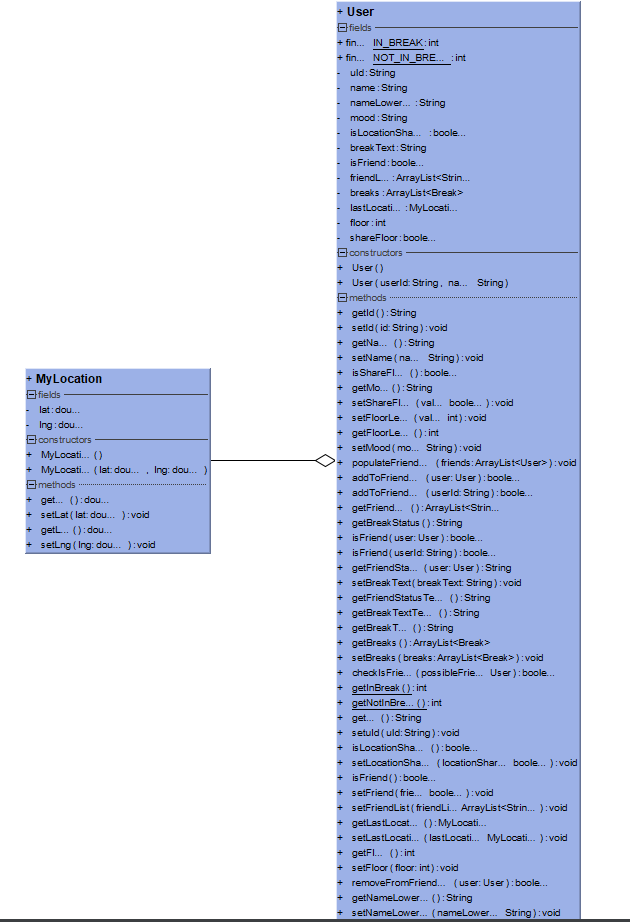




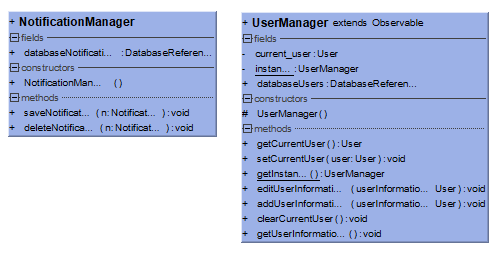
Models







Managers



Documentation

Since the documentation is extremely large (i.e. more than 300 pages long), it has been put as appendix to this document. Therefore, if the JavaDoc needs to be consulted, it will be found at the end of this document.

Note: The format of this documentation is not the best due to a large amount of conversions (HTML to PDF to RTF to DOCX). However, the key elements are still available to the reader.