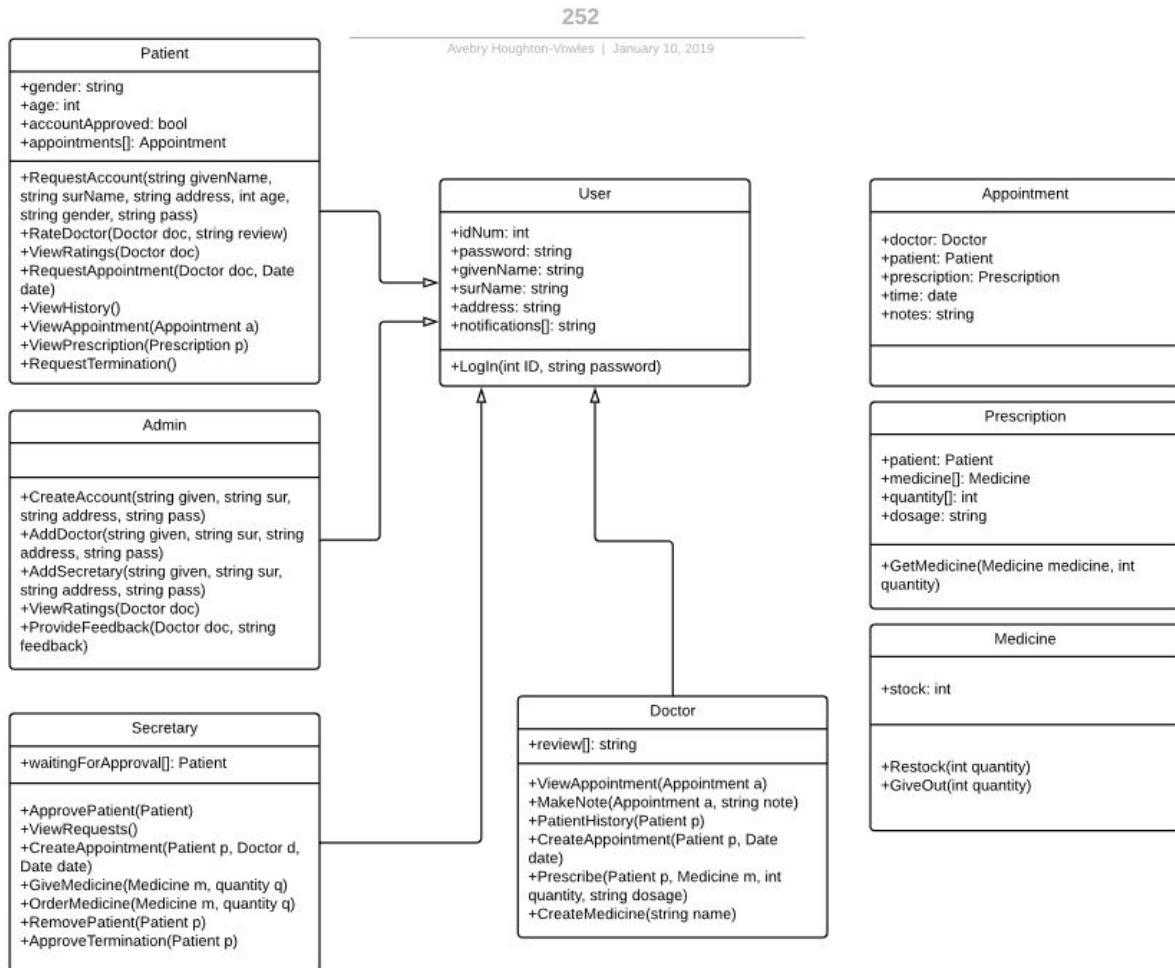


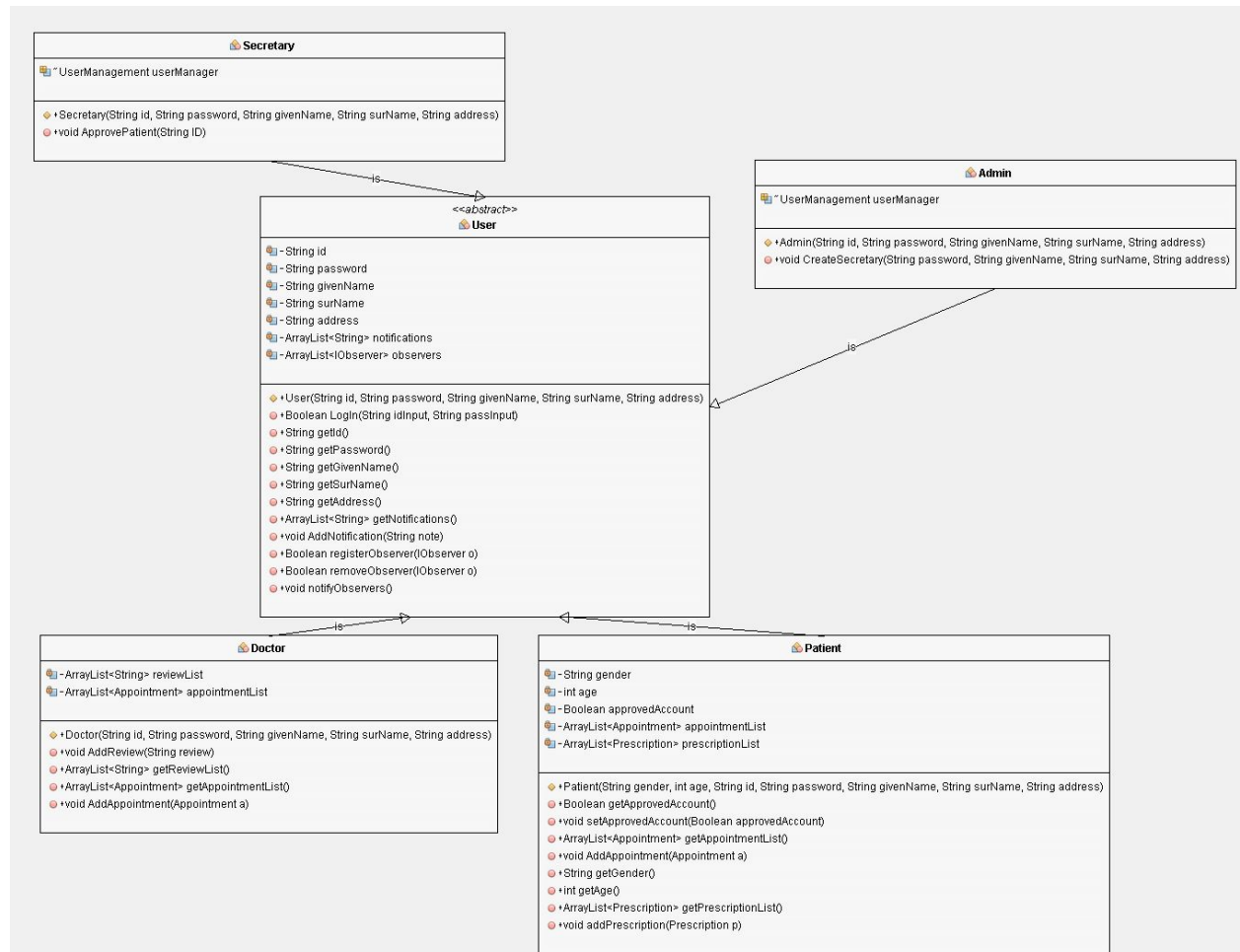
SOFT252 Coursework Reflective Report

Design Process

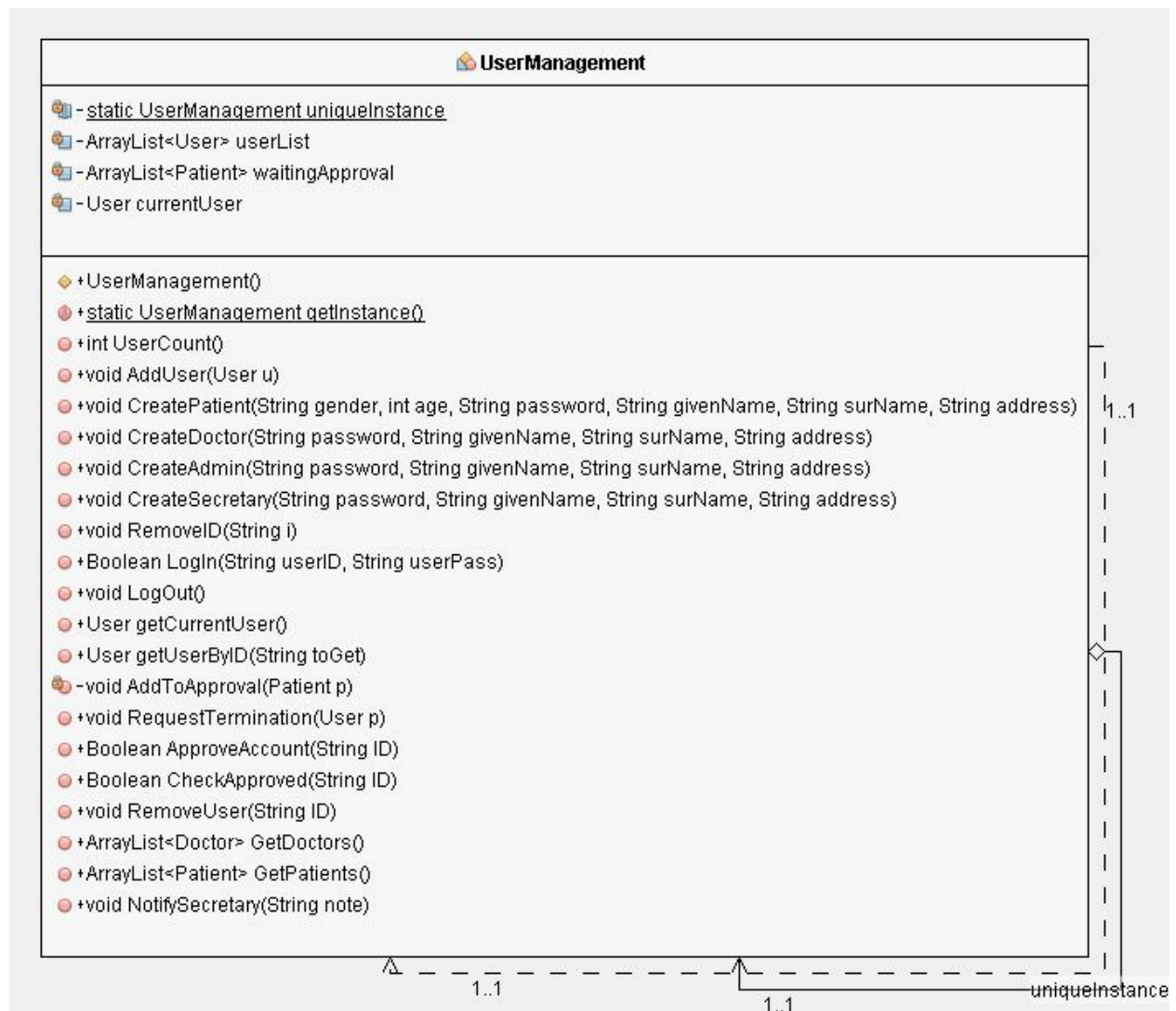
Early UML diagram of the user classes:



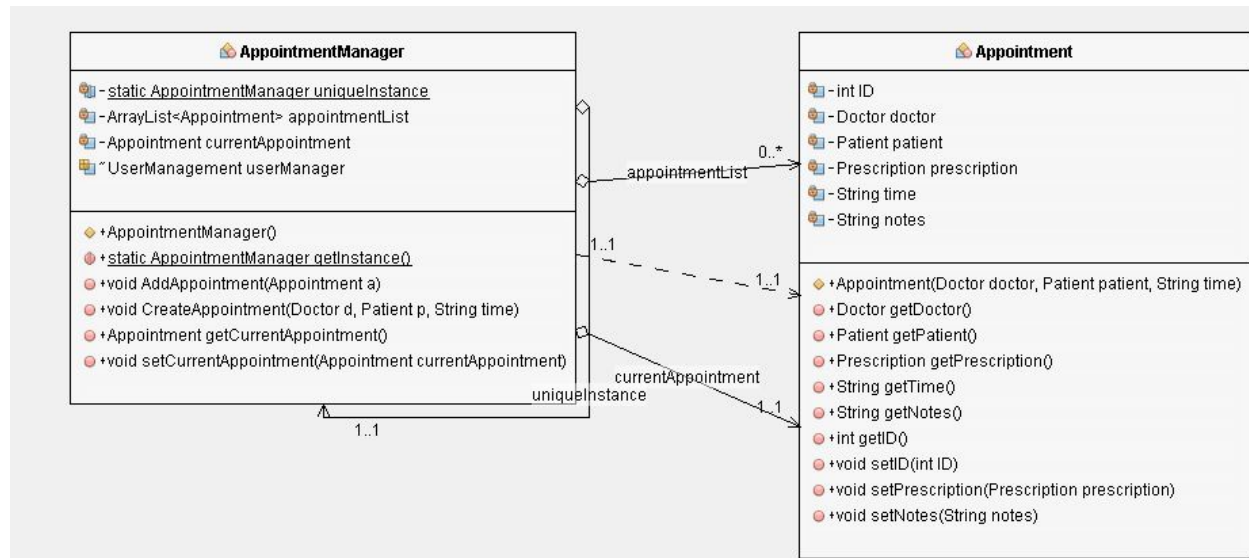
Final UML diagram:



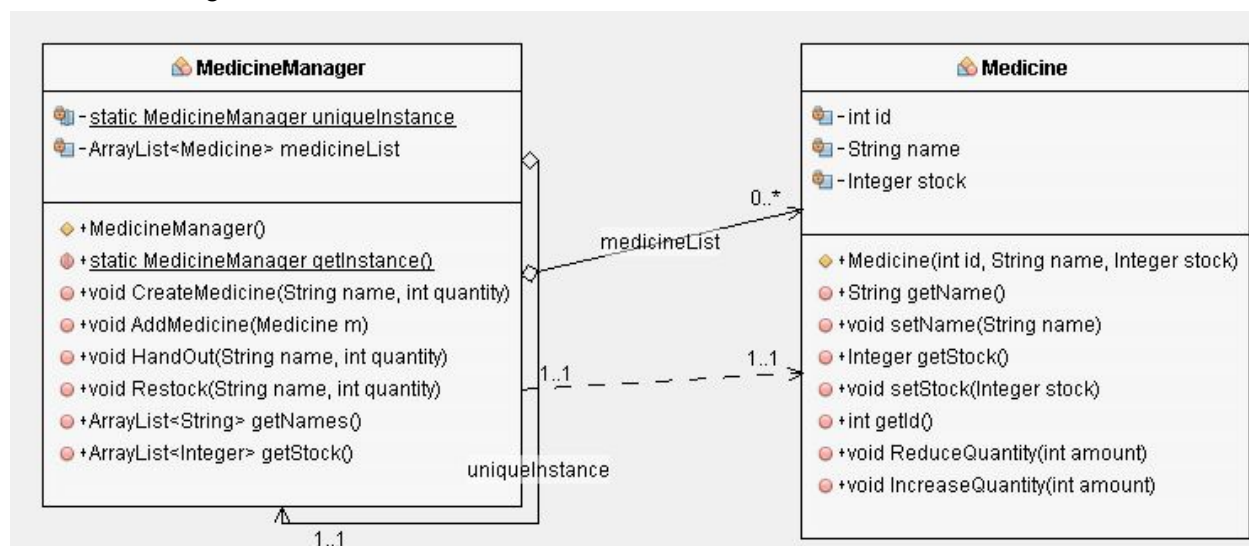
User Management UML:



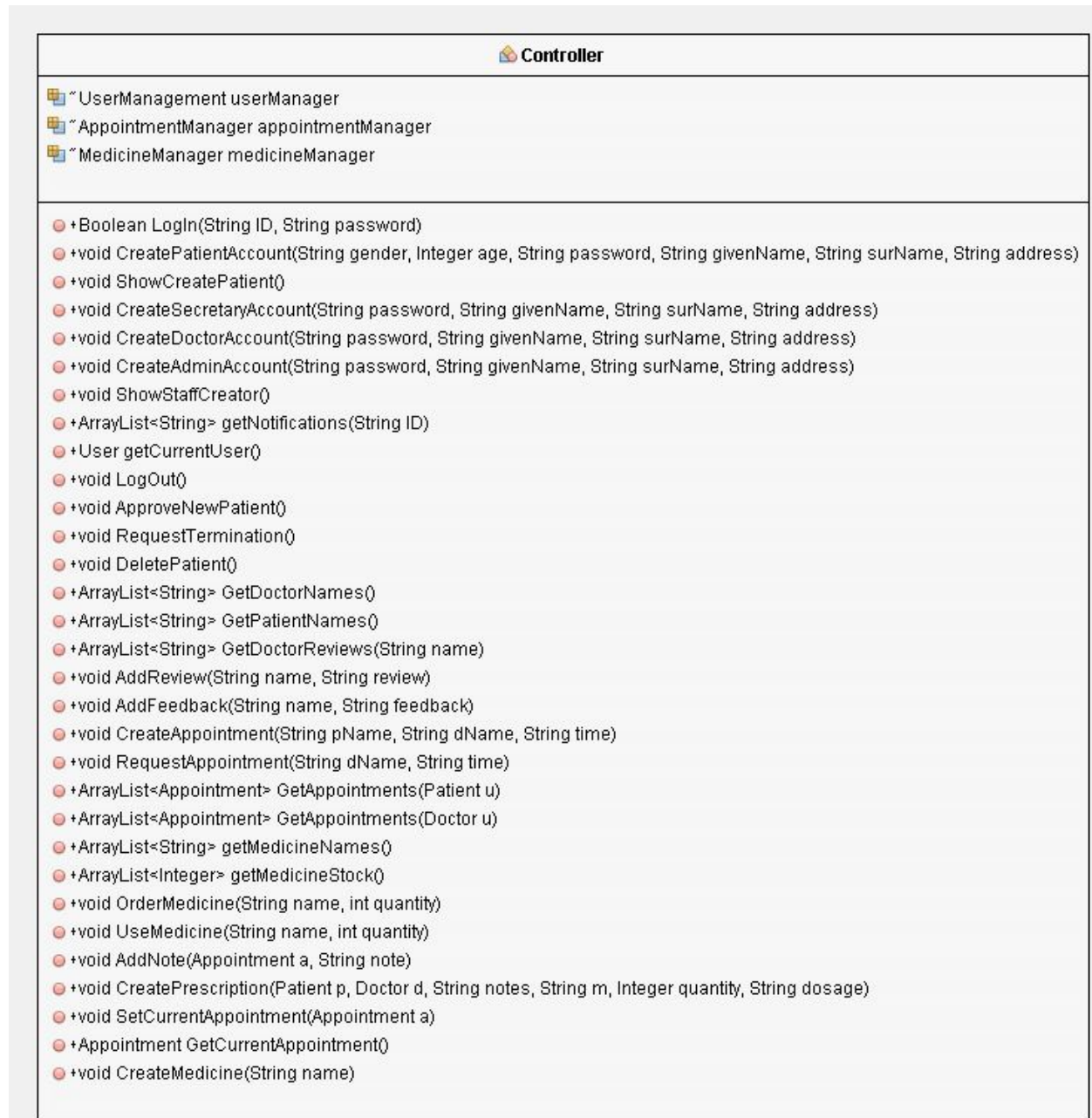
Appointment Manager UML:



Medicine Manager UML:



Controller UML:



Design Choices

The class UserManagement stores a list of all users in the program, and handles most operations related to them. As this would need to be accessed by most classes in the program, I decided to use the Singleton design pattern for it. This allows the same instance of UserManagement storing the list of users to be accessed everywhere. As one instance of the classes AppointmentManager and MedicineManager are also required throughout the program, they also utilise the Singleton pattern.

As the notification output should subscribe to a user and update whenever a new notification is added, it should be built using the Observer pattern.

Successes

In the view, the GUI for each type of user is constructed in a separate JFrame. This helps prevent overlap between the permissions and methods available to each type, and means that each user will only see the options relevant to them. Having separate frames for each major task also means that frames can be shared between functions if necessary, such as viewing and writing doctor reports for both admins and patients.

Another success is the implementation of the userManager class. By gathering most of the user-related methods and lists into one class, then applying the Singleton pattern to it, these functions and data can easily be accessed throughout the program in other packages without needing multiple imports. The Singleton pattern also guarantees that the main storage lists will not be overwritten.

When an input with limited options would have to be entered into a JFrame, such as selecting patients and doctors to be assigned an appointment, the input is done via a Combo Box instead of an open text field. When the combo box model is properly assigned, this eliminates the risk that the user will enter an invalid input.

Shortcomings

A shortcoming in the submitted work is that I feel the implementation of MVC is lacking. This is due to the fact that the View still contains its own methods, instead of passing everything to the Controller. An example of this is in the JFrame DoctorForm, which has its own methods to retrieve the list of notifications instead of handing it off to the controller. This could be remedied by better planning out the functions in the Controller.

Another shortcoming in the code is that the notifications for a user are only currently checked when the user logs in, or directly adds a new notification. If a user were to receive a new notification while logged in, it would not display until next login. This could be solved by properly implementing the Observer pattern with some kind of notification manager, which I failed to do due to poor planning and time constraints.

The submitted program is also incapable of saving data, as I was unable to figure out how to get the Serialiser working in time. Again, this is due to poor planning and time management, and could be solved if I had fewer pieces of coursework to do at the time.