Patient Management System Reflective Document

The design process began with researching different types of prominent Patient Management Systems(Hereby referred to as PMS) to further understand how their user interface is commonly designed. I noted reoccuring themes such as user feedback and tabs and used this as a basis for the design. The final project should be a fully working local PMS with a set of predefined criteria. These criteria were used to develop very short sprints; which I used to methodically work through the criteria, thus following a semblance of a Top-Down Program Design. At the end of each sprint, I uploaded the updated project to github with an appropriate title. I prototyped these short sprints within their methods before connecting them to their relevant methods. For example, in registering a new patient I first checked that when the patient attempts to register, the output was correct, before moving onto the secretary approving this registration and creating a new patient. I planned to use iteration to get a working product; then refactor the code until it was more efficient afterwards.

As I treated this as though I was developing it as a live project, rather than a ‘proof of concept’ that university projects normally entail, when designing the login system I made the password use a password field to star out the password when logging in and registering. This security was also maintained through the use of tabs - I hid all tabs from the applications launch, besides registering and login - This was to hide any data and layout that could be gleaned from having the different user tabs open, even if they were not interactable, while also decluttering the UI.

The use of inheritance for the types of people classes allowed me to reuse some code for the different roles of people. The use of Arraylists also made it easier to keep track of and iterate through the various class instances created during the program. Furthermore, I could have simply used one Arraylist - a user list and used that instead. However, in cases where a patient or doctor may have the same name as a patient, this could have provided inconsistency in which user would be logged in and a user may unintentionally gain or lose privileges. In many methods I prefaced the main function of the code with an if statement checking whether the values of all inputs were not null. This was useful as it stopped any irregularities in the objects that were created and lowered the chance of an error.

As previously mentioned, the security of a login system is particularly important. I would have preferred to hash the passwords for security, however as there are some predefined users for each role in the source code as per the criteria, anyone who was willing to force access into the

system would be able to by accessing those users, particularly an admin instance, which could have devastating consequences.
The different roles of users also inherit from a base person class, while this is better than not inheriting at all it would have been preferable to use composition over inheritance.
There is a distinct lack of design patterns used in this project which directly affects its modularity. Preferably more would have been used which would have made maintenance more inexpensive and less time consuming during production. For example, I could likely have used an observer pattern to update doctors, secretaries and patients on messages they should receive. I also could have used a singleton pattern to update the current user once they've logged in, then filled all information that was required in here, instead of immediately on login.
Code reuse was very minimal here, which is bad practice for larger products. In combination with the lack of design patterns, given more time this could be made far more efficient.
The user layout is particularly inefficient and could be made more user friendly by providing prompts and more feedback after actions are taken. It also likely would have been useful to create the uml diagrams before starting the programming as I could use that as a basis for my code layout, making my design process more succinct. One criteria that I had to omit was to make the user classes serialisable which would allow the project to get the previous state on application exit and restart. While this would have been useful, I was unable to meaningfully implement it with my current iteration.
I also used for loops a great deal to iterate through user lists, through code refactoring I could have used an 'active' user variable for each role and referred to that, rather than iterating through the list anytime I needed to find a name or username.

Github Link:

https://github.com/10203040506070809/Patient-Management-System