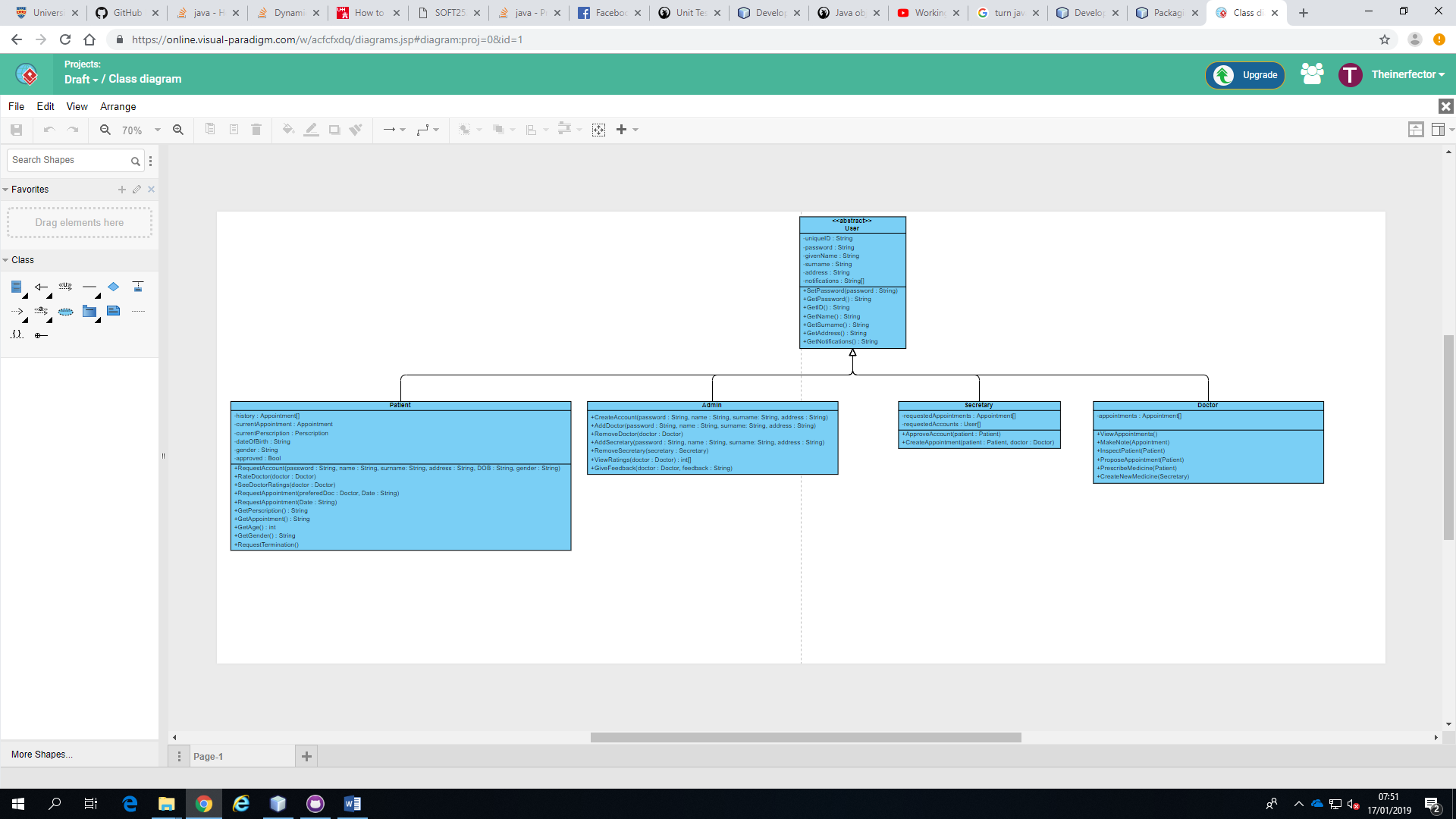
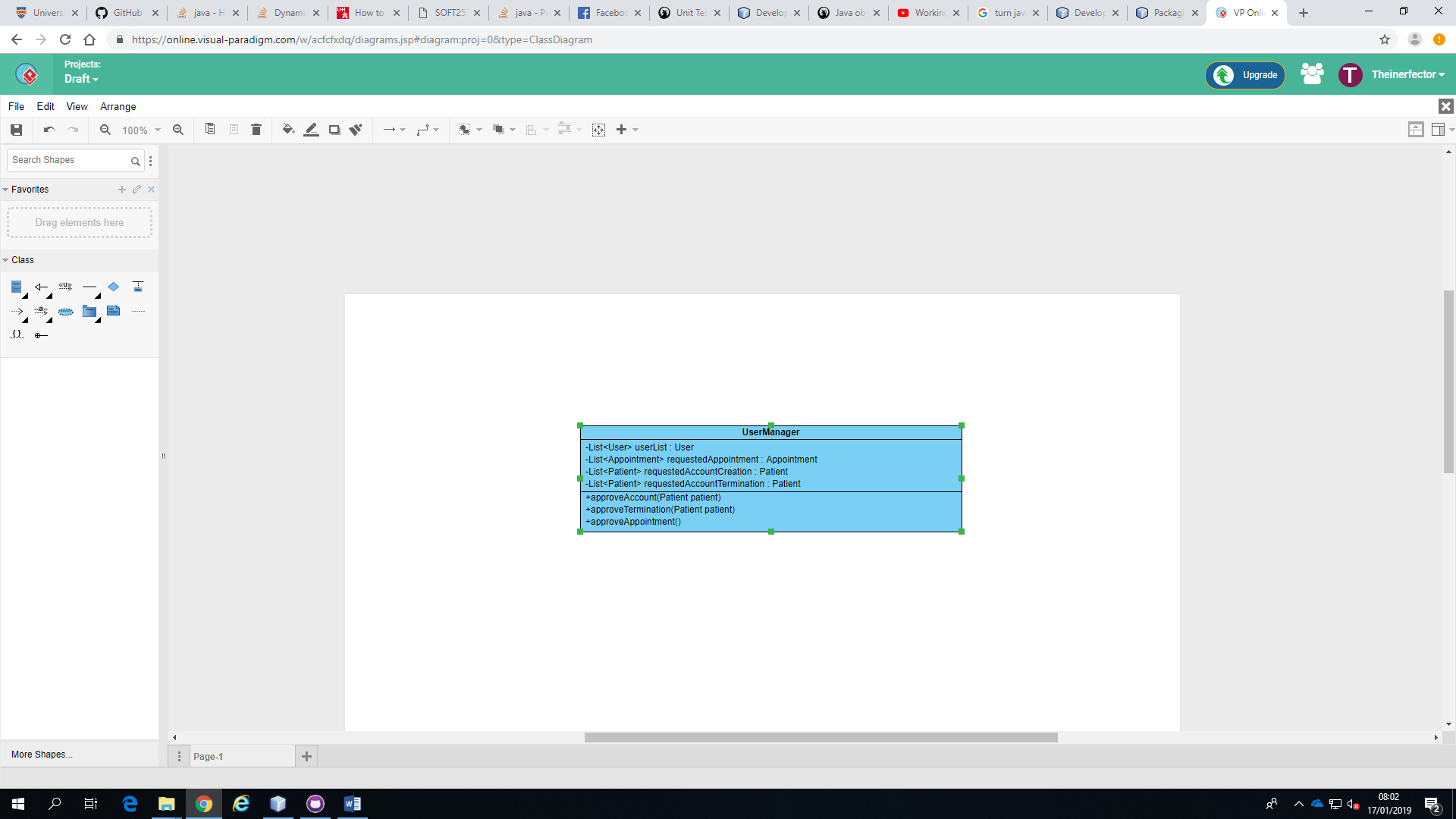
Reflective document

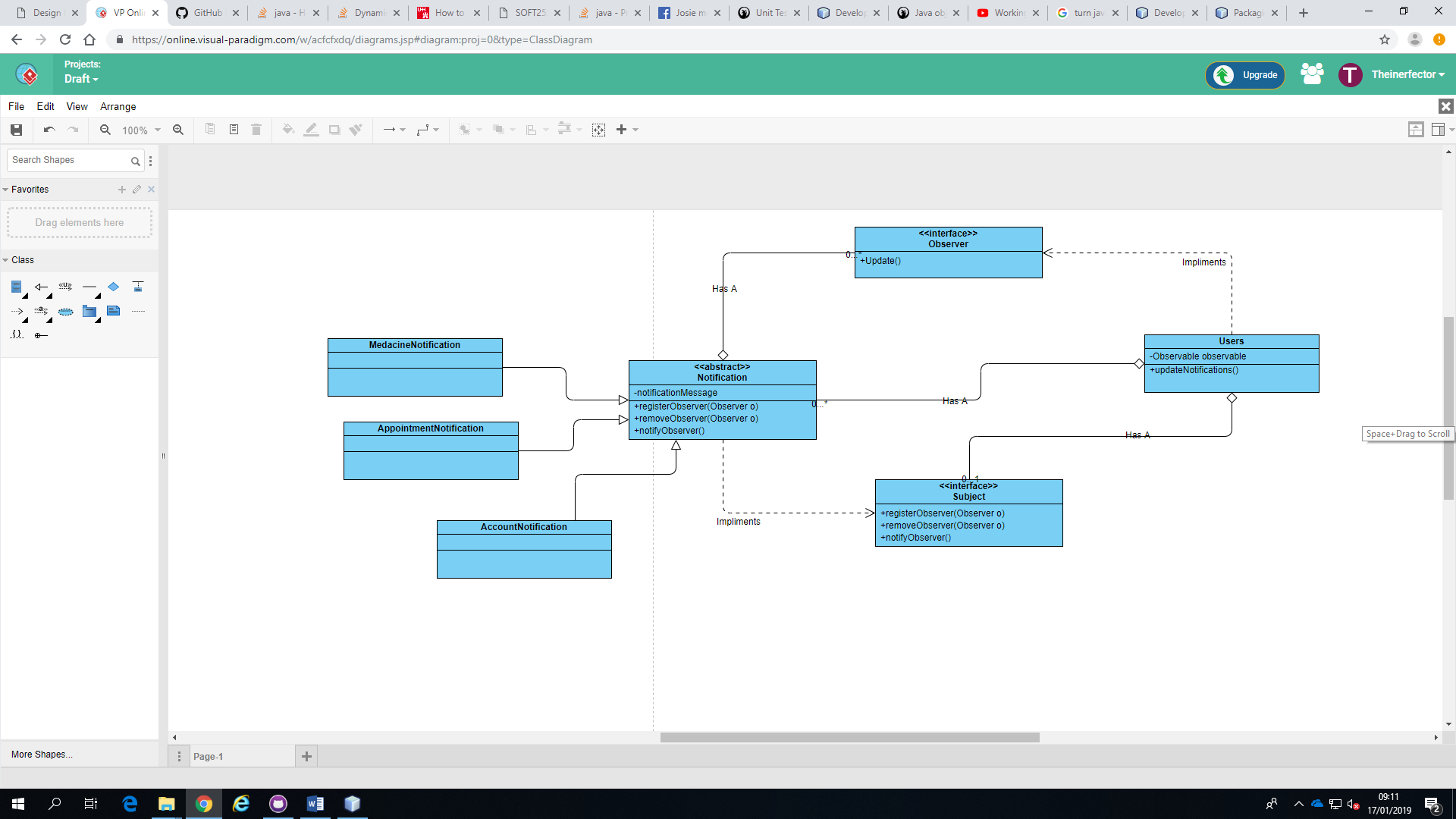
Git Hub Link: <https://github.com/HungryHorse/Patient-Management>

I started by creating a UML diagram to show my user as an abstract class and the different classes that extend it:



This UML diagram is based off of the sheet given to us in spec. I wanted to make sure that I implemented at least 2 different design patterns, I next created a UML for my observer pattern and singleton pattern





The reason I chose these design patterns is that I believe they fit in extremely well and also make the software and code cleaner. These design patterns also have low coupling and high cohesion which is an import part of making a flexible program that is open to extension and closed to modification as you can add more classes as needed and you will not have to change much at all. After creating these models I tried to work out how to do Junit testing, however after struggling for about 3 hours I decided to instead move on to creating the model. I first coded the User class and all of its children. This was an easy task due to my detailed UML, however things that I didn’t expect to be necessary stared coming up as I continued on to create my singleton User Manager. As I created my user manager I realised that I had to create multiple functions and parameters that I didn’t initially suspect. This shows that the UML stage of my design could have been a lot more in depth than it was. After creating the model I once again tried without luck to create Junit tests. This is an issue as without testing I had no idea whether my model worked. To combat this I used my package to test some of the interactions. This helped relieve the issue of not being able to test with Junit. After creating my model I moved on to my GUI and controller. One part of this project I’m very proud of is my ability to stick to the MVC principal. The view interacts with the controller and the controller interacts with the model. There are few times when this is not the case but I found them unavoidable. I spent most of my time creating the links between the GUI and the model via the controller and I believe this shows as you can see the links obviously between the two. After creating the controller I finally went back to Junit testing and finally cracked it, it turns out it wasn’t too hard in the first place. I just didn’t really understand how to implement it. Once I implemented some Junit tests and checked that they were returning correctly. All the test I implemented passed. As I didn’t really have time to create a way to save to file, as was learning serialization for the first time, so I instead just hard coded some users for the testing process. I then built the project ready to be uploaded. Notifications have been coded but will only appear if something changes during the program so the notifications that should show up at the start don’t.

The positive aspects of my project are the implementation of MVC and the models neat structure.

The negative was defiantly the lack of Junit tests and regular commits.