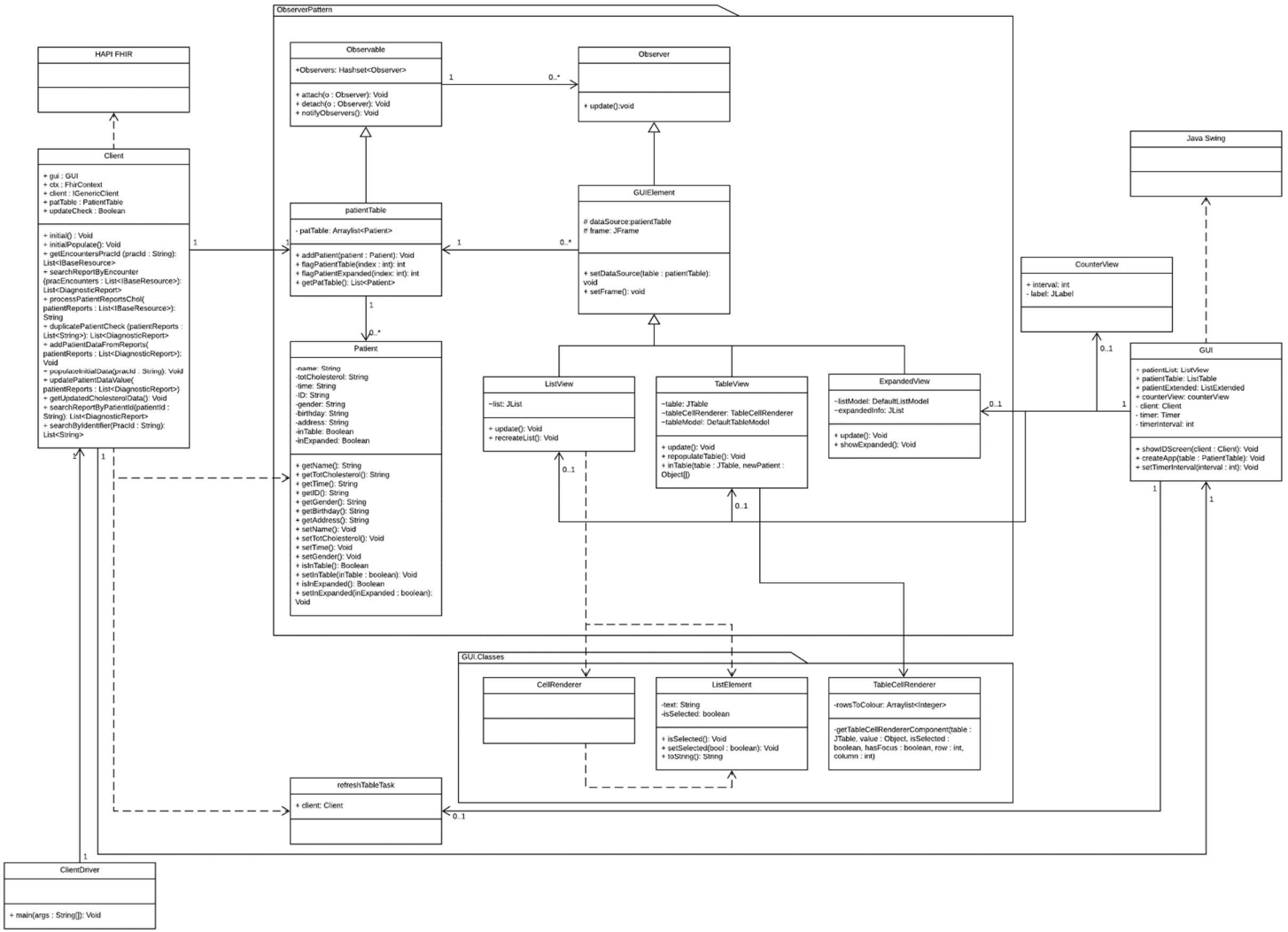
UML Class Diagram



To implement the feature required, an Observer/Observable design pattern was used, with each GUI that needs to update itself due to changes in the data source inheriting from Observer, and the patientTable inheriting from Observable. This is the most important part of the system: every other class just implements other features. Another important note is that the Observers can edit the data source, and every other observer is able to see the changes made. This is necessary as observers do not directly edit each other: this would become extremely messy as the system expanded. Instead, they all edit one central data source, which every observer can see. Any info change in the patient should ripple to the list and every other observer too, and this is made extremely easy with the Observer/Observable design pattern, and what we have implemented. Each view can flag specific patients, and other observers are able to see that and update themselves. Specifically, clicking on a patient in the list flags it for the table, or any other view to pick up. The GUI holds the frame that all the views exist on, and sets up the observer relationships.

The Observable class demonstrates DRY by having methods that the observers can use to edit the table, rather than the code being repeated in each instance of a unique observer. Having these methods makes it very easy to continue to expand the system even if more GUI elements are required.

For the Liskov Substitution Principle and the Open/Closed principle, the GUI Elements and views adhere to these. For the O/C principle, the GUI Element class which is still abstract is open for extension, but closed for modification. This preserves the main functionality of the class within the design pattern: an observer must always have the ability to look at a data source, and in the case of this application, must always be set to a frame. The class is open to extension(it is extended by each view), but closed for modification. Because it complies with this principle, it also complies with the LSP, because any subclass that inherits GUI element, could work in place of the abstract GUI element. The relevant methods are overridden, and new ones are added as necessary.

The classes are also organised into packages so that classes that change together, belong together. This prioritises maintainability over reuse, because reuse is less important to the developer. The system also complies with ADP, ensuring that there are no cyclic dependencies.

The client also adheres to the principle of Don't Repeat Yourself as the methods serve a singular purpose, which allow for modularising a method based on a specific use case, meaning that instead of repeating code to perform a similar task more generalised methods are used in tandem to accomplish the same thing. Additionally, this modularity means that Open/Closed Principle is adhered to as any further methods added do not require the previous methods present to be modified and can instead add new modular methods to assist in the satisfaction of a new use case. The compliance of these two principles can be seen in the methods that is utilised by the initial populating of the data and the subsequent update every N seconds as the two methods that deal with this use the modular methods present to accomplish their tasks, without containing any code within that is repeated elsewhere.

Regarding the overall runtime of the application, it does take a great deal of time due to the searches that are required. As the within the HAPI FHIR database just searching for the practitioner ID is not enough as that does not provide all their patients. Instead each practitioner has an identifier which is then used to find all the practitioner ID's related to that identifier. From there we search encounter containing those identifiers and then using those encounter we attempt to find their related diagnostic report. From there we filter it so only the most relevant remain and their observations and patient data is read from the database and stored in the patientTable.