Design Rationale

The design of this application takes inspiration from the Model View Controller design pattern and splits the classes present into 3 main categories. The first of these are the view classes that are responsible for how the interface looks and is presented to the user. The second are the data organization classes, being the two patient list classes. These act as somewhat of a controller to organize the data being given to the view classes. Finally, the request classes obtain and handle the incoming data so it can be used by the controller classes, acting like the core model for the application. The main class brings these classes together in order to create a functional application. Using this approach, only the main and view classes have to be dependent on the UI package used, and only the model classes are dependent on the FHIR requests class used. This leaves the controller classes independent from these external libraries.

Originally intended to use an adaptor to use Patient class inside the ObservableList, but Patient did not allow child classes to be compatible with requesting and parsing the server data through the hapi-fhir API so it was decided that a new cholesterolPatient class was to be made

* More lightweight
* Easier access to relevant information
* Compatible with TableView

Main Class:

This class extends the Application class in the library javafx. This enables it to launch the javafx application, and is given a stage to build the application from. This is essential for any application to

View classes:

There are several “View” classes present in the design, such as MonitorPatientsTableView and AddPatientsTableView. These are all children of the javaFX Region class. This allows them to be easily implementable into any javaFX scene, and gives modularity to how the UI is built, whilst still leaving them relatively closed These view classes were designed with the Open-Closed principal, as they use inheritance to add functionality to the Region classes, in order to add content to the region, and being able to use the view classes like any other region, following the Liskov substitution principle.

Request classes:

The requests class handles all of the network requests, and from this, all of the data obtaining for the application. In order to make the method of obtaining data independent from the classes that require this information, and to better conform to the dependency inversion principle, an interface was used to access this service, in the form of GetPatients. This reduces dependencies on requests so it can be changed more freely, and will allow easier extension of requests if other data is needed in the future, as well as an easy route into applying the interface segregation principle if this data is needed for a different purpose to what is currently used.

Once the requests class has obtained the data, it is often sent to the CholesterolPatient class for easier organization within the application. The Hapi Fhir framework has an inbuilt patient class that was considered for this role through the use of an adaptor, but this was decided against as an adaptor class did not work well with the framework, and the inbuilt patient class was complex without all of the functionality that was required. Thus, the CholesterolPatient class was created to better encapsulate this data. This also allowed the class to work with the observer framework built in to the TableView class of the UI, which was implemented in the PatientList classes.

PatientList classes

In the system, there are two PatientList classes. The parent class PatientList handles the basic adding and removing of patients to the system, and the two core attributes of the patient list and the patient getter. Following the open/closed principle, the parent class is open to modification through inheritance, where a child class can add the lower level functionality that is needed from the PatientList, for example, in the MonitoredPatient class, it extends functionality to be able to store and monitor the cholesterol for each of the patients, but still using the core functionality from the parent class. This will allow easy modification if for example, another statistic was needed to be added to the system.