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 all 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, allowing the construction of the application to be more modular and extensible, giving it a better separation of concerns. This makes it easier to have, for example, alternate user interfaces or data requests if needed and makes testing easier due to the de-coupling of each of the elements.

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 their details closed for editing. These view classes were designed with the Open-Closed principal, as they use inheritance to add functionality and content to the Region classes. This allows the view classes to be used like any other region, following the Liskov substitution principle. This also makes the UI easily extensible, as new nodes can be created and added to the system without interfering with those already present
Most of the time, the UI will require data from the patientLists in response to user input, which each view can obtain from their attribute of the list, however, as the association between the classes only goes one way, the tableViews use an inbuilt observer framework to observe the ObservableList attributes in the PatientsList class. This prevents a cyclic dependency, whilst allowing the PatientsList classes to update the UI as the values in the list changes. Another feature of the TableView library used is the use of factories in generating the cells of each of the collumns in the table view. This greatly simplified the code as it allowed the creation of all the appropriate cells automatically from the list of cholesterolPatients found in the PatientList class

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 conform to the dependency inversion principle, an interface was used to access this service, in the form of GetPatients. This reduces dependencies on requests which increases its modifiability and extensibility. This was seen when the machine learning extension was added to the class, using the interface segregation principle for an alternate client requesting data. A possible weakness in this design is the tendancy for the interfaced class, in this case requests, to become excessively large, as it violates the open-closed principle. In the future it may be beneficial to modularize requests through inheritance, instead of compartmentalizing it's features through interfaces, however considering the scale of this application, the simplicity and efficiency of having a single requests class with multiple interfaces was chosen instead.

Once the requests class obtains the data, it is often sent to the CholesterolPatient class for better organization within the application. The Hapi Fhir framework has an inbuilt patient class that was consideredto be used in the application through the use of an adaptor, but this was decided against as an adaptor did not work well with the Fhir 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 functionality of managing a observable list of patients. Following the open/closed principle, the MonitoredPatient class extends this basic functionality to be able to store and monitor the cholesterol for each of the patients, while still using the core functionality it inherited from the parent class. This design allows easy extension of the system with different ways patients can be monitored, whilst keeping core functionality simple and stable

Weka

MachineLearning
-request: GetWeka
+grabData(filename: String, bundleCount: int)
+testData(test: String)

MachineLearningView

JavaFX

ActionButtonTableCell

scene.control.TableCell

MonitoredPatientsList
-updateCholesterol: Runnable
-updateCholesterolService: ScheduledExecutorService
-cholesterolGetter: GetPatientsCholesterol
-updateCholesterol(CholesterolPatient: patient)
-averageCholesterol(): BigDecimal
+setUpdateFrequency(int: timeBetweenUpdates)
+isBelowAverage(CholesterolPatient: patient):boolean

MonitorPatientsTableView
-patientTable: TableView
-detailsView: DetailsView
-patients: MonitoredPatientsList
-generateUpdatesView(): Region

scene.layout.Region

DetailsView

PatientsList
-GetPatients: patientGetter
-patients: ObservableList<CholesterolPatient>
+addPatients(String practitionerIdentifier)
+removePatient(Patient: patient)
+addPatient(Patient: patient)

AddPatientsTableView
createPractitionerIdentifierSearch(sourceList: PatientList):Region

«Interface»
GetPatients
+getPatientForPractitioner(practitionerIdentifier: String): ArrayList
+getPatient(patientID:String): CholesterolPatient

CholesterolPatient
«get»-cholesterolValue:BigDecimal
«get»-cholesterolString: StringProperty
«get»-updateTime: StringProperty
-address: Address
-gender: Enum
-birthDate: DateType
+getGenderString(): String
+getBirthdateString(): String
+getAddressString(): String
+updateCholesterolAndTime(cholesterolBase: Base)

Main
+start(Stage primaryStage)
+main()

application.Application

«interface»
GetWeka
+getAllOfObservation(BundleCount: int): List
+getPatientWeka(url: String): Patient

«Interface»
GetPatientsCholesterol
+getPatientCholesterol(patientID: String): Base

BasePatient
«get/set»-name: String
«get»-id: String

Requests
-client: IGenericClient
-CHOELSTEROL_CODE: String

org.hl7.fhir.r4.model

FhirContext

IGenericClient