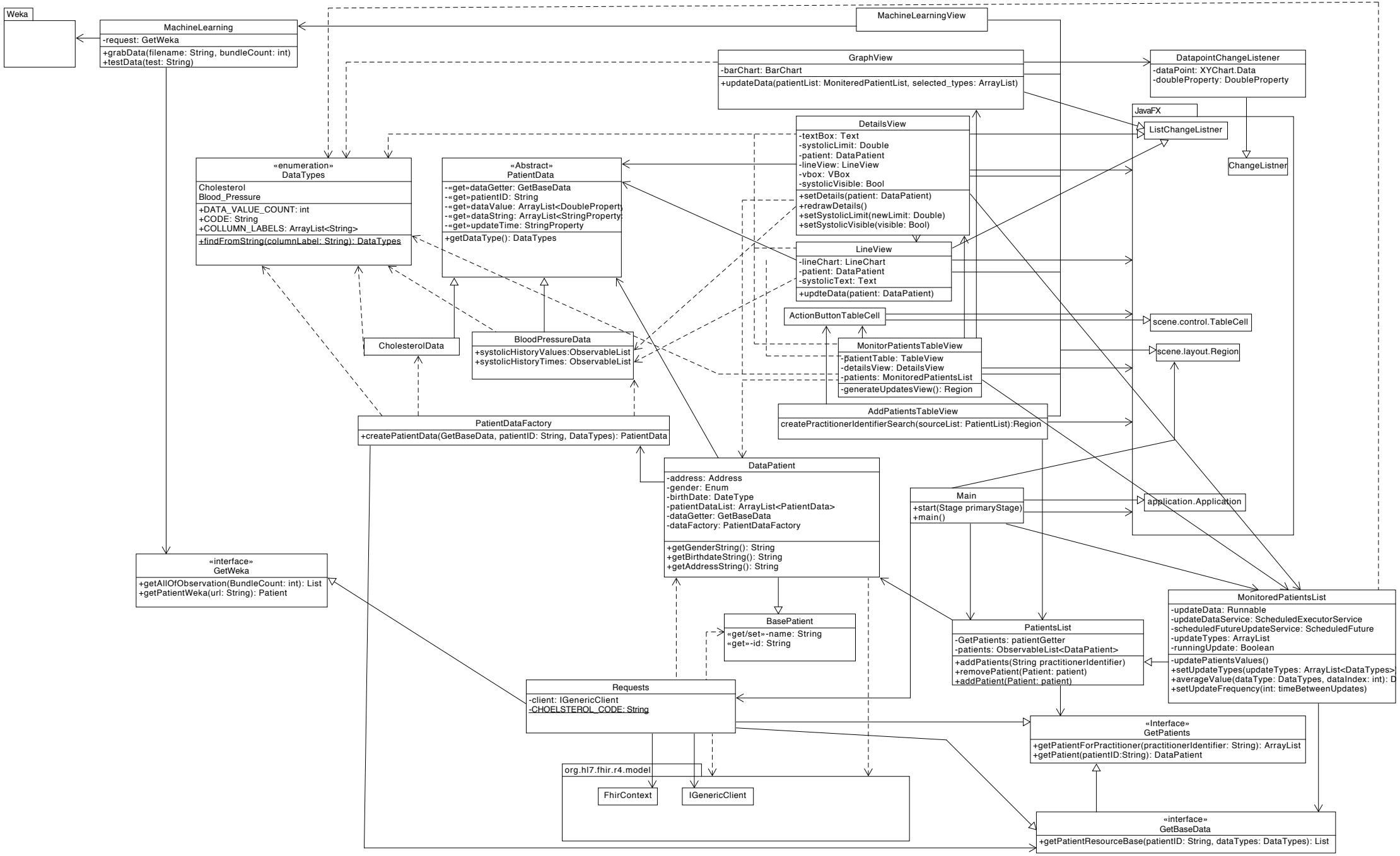
The design iteratively improved upon the initial design from Assignment 2. Some design choices from assignment 2 worked well, such as the unified requests class handling all of the networking of the application, and the view classes, that handled the UI of the application. However, major refactoring was required in the data classes that handled the information pulled from the server. Initially, our design used a single CholesterolPatient class to manage the information on each patient, however with the application now required to hold blood pressure data and cholesterol data, it was decided that CholesterolPatient would be refactored into DataPatient, and the attributes containing the cholesterol data would be extracted into a new class CholesterolData. In order to facilitate different types of data in the application, methods and attributes in CholesterolData were pulled up into a new abstract class, PatientData that could be inherited off with the addition of more data types. This allows the dependency inversion principle to be used to follow the open-closed principle, as the abstract PatientData class is independent from concrete implementation, and it is able to be extended off for each different data type required. The factory pattern was then used to manage the creation of these data classes, so no other classes would need to be dependent on the data classes. This new approach leads to a significantly more extensible design, where new data types to track can be added easily, whilst also decreasing the stability of the previously monolithic CholesterolPatient class, into several classes with increased cohesion and decreased dependencies.

Other refactorings were required to the design of the application in order to reduce repetition of code. In the requests class, methods such as getPatientResourceBundle were extracted from getPatientCholesterol which allowed the elimination of repeated code in getAllOfObservation. On the user interface side, MonitorPatientsTableView required large chunks of code to be extracted into independent methods, as this then allowed them to be called multiple times from different places. This also facilitated the ability to dynamically update the data shown on screen, as well as reduce the amount of repeated code.

One of the challenges faced in the design of this application was the requirement of updating values across the application every n seconds, specified by the user. We overcame this challenge without creating unnecessary dependencies by using the observer pattern, seen in the Double/StringProperty attributes and ObservableList implementations in javaFX used. These allowed our view classes to be implemented as listeners, where updates could be executed when the observer notified them of changes. This allowed the view classes to be updated without being dependent on the implementation details of the data classes behind them, effectively avoiding cyclic dependencies. For example, the GraphView class implements ListChangeListener, in order to be an observer of the MonitoredPatientList and receive onChanged updates when this list changes.

Other than this, our user interface, or View classes were relatively easy to extend with new functionality, where two new view classes could be created for the new graph functionality, in the exact same way that the table views were added to the UI initially. This allowed the UI elements to be handled like modular building containers, where they could be placed into a relevant view and handled mostly independent of one another.


Weka
MachineLearning
-request: GetWeka
+grabData(filename: String, bundleCount: int)
+testData(test: String)
MachineLearningView
GraphView
-barChart: BarChart
+updateData(patientList: MoniteredPatientList, selected_types: ArrayList)
DatapointChangeListener
-dataPoint: XYChart.Data
-doubleProperty: DoubleProperty
JavaFX
ListChangeListner
ChangeListner
DetailsView
-textBox: Text
-systolicLimit: Double
-patient: DataPatient
-lineView: LineView
-vbox: VBox
-systolicVisible: Bool
+setDetails(patient: DataPatient)
+redrawDetails()
+setSystolicLimit(newLimit: Double)
+setSystolicVisible(visible: Bool)
«enumeration»
DataTypes
Cholesterol
Blood_Pressure
+DATA_VALUE_COUNT: int
+CODE: String
+COLLUMN_LABELS: ArrayList<String>
+findFromString(columnLabel: String): DataTypes
«Abstract»
PatientData
-«get»dataGetter: GetBaseData
-«get»patientID: String
-«get»dataValue: ArrayList<DoubleProperty
-«get»dataString: ArrayList<StringProperty
-«get»updateTime: StringProperty
+getDataType(): DataTypes
LineView
-lineChart: LineChart
-patient: DataPatient
-systolicText: Text
+updteData(patient: DataPatient)
ActionButtonTableCell
scene.control.TableCell
scene.layout.Region
CholesterolData
BloodPressureData
+systolicHistoryValues:ObservableList
+systolicHistoryTimes: ObservableList
MonitorPatientsTableView
-patientTable: TableView
-detailsView: DetailsView
-patients: MonitoredPatientsList
-generateUpdatesView(): Region
AddPatientsTableView
createPractitionerIdentifierSearch(sourceList: PatientList):Region
PatientDataFactory
+createPatientData(GetBaseData, patientID: String, DataTypes): PatientData
DataPatient
-address: Address
-gender: Enum
-birthDate: DateType
-patientDataList: ArrayList<PatientData>
-dataGetter: GetBaseData
-dataFactory: PatientDataFactory
+getGenderString(): String
+getBirthdateString(): String
+getAddressString(): String
Main
+start(Stage primaryStage)
+main()
application.Application
«interface»
GetWeka
+getAllOfObservation(BundleCount: int): List
+getPatientWeka(url: String): Patient
BasePatient
«get/set»-name: String
«get»-id: String
PatientsList
-GetPatients: patientGetter
-patients: ObservableList<DataPatient>
+addPatients(String practitionerIdentifier)
+removePatient(Patient: patient)
+addPatient(Patient: patient)
MonitoredPatientsList
-updateData: Runnable
-updateDataService: ScheduledExecutorService
-scheduledFutureUpdateService: ScheduledFuture
-updateTypes: ArrayList
-runningUpdate: Boolean
-updatePatientsValues()
+setUpdateTypes(updateTypes: ArrayList<DataTypes>)
+averageValue(dataType: DataTypes, dataIndex: int): D
+setUpdateFrequency(int: timeBetweenUpdates)
Requests
-client: IGenericClient
-CHOELSTEROL_CODE: String
«Interface»
GetPatients
+getPatientForPractitioner(practitionerIdentifier: String): ArrayList
+getPatient(patientID:String): DataPatient
org.hl7.fhir.r4.model
FhirContext
IGenericClient
«interface»
GetBaseData
+getPatientResourceBase(patientID: String, dataTypes: DataTypes): List