Design Patterns and Principles Used:

Observer Pattern:
As the textual monitor is also observed at the same rate of the normal table, the update function of the subject is called at the same rate. Therefore, a new observer was created to observe the changes in latest recordings of the patients (if the latest recording is updated then the textual monitor table is updated).
This allows us to detach / attach observers to patients whose levels go above the set threshold when the users decide to start/stop monitoring. Therefore, the subject does not need to know that it is being monitored as the observer handles this, and made it easier for us to extend our program.

Model-View-Controller:
In addition to the last assignment’s design, the controller is also responsible for creating the charts in which the models' data are displayed on. This reduces the dependencies of the Table Models on the JFreeChart API, and allows the extensibility of adding other charts of the same type (i.e. bar chart or xy series), with different data sets, to different tables through the controller.

Although it allows extensibility, a disadvantage is that it may make the Controller class code more complex as settings for the chart need to be added- but also makes it easier to navigate when altering the default appearance/settings of the chart.
Trade-off between having both cholesterol and blood pressure in one table vs having separate tables:

Having multiple views instead of both in one table allows us to modify what, and how it is displayed for a specific patient when monitoring a given measurement. The downside of this decision is that with a larger number of measurements, the number of tables will increase. This may cause the performance to hinder due to more visual elements to populate and update.

Liskov Substitutability Principle:
When constructing the various concrete table models, and concrete observers, this design principle was considered to ensure that the properties of inheritance were not abused in this design. This consideration ensures that whenever an instance of a subclass is given in the context of the parent class it behaves as expected by the parent class.

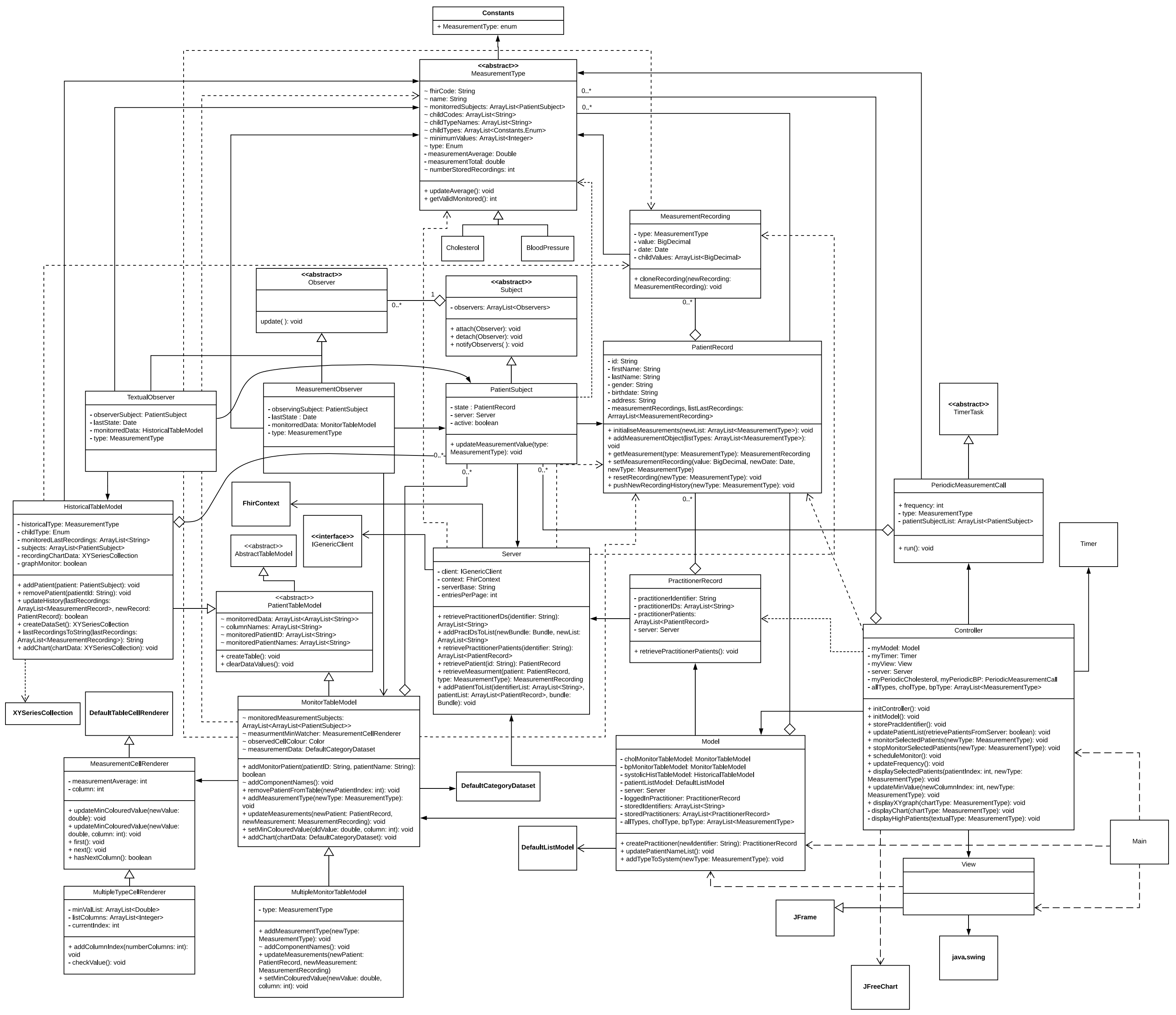
Refactoring Techniques
- **Renamed the method** to set the value for the cell renderer to change the colour of values in the table. Previously it was named as setting the average, however since this method is used when the cell’s value exceeds the set value (not necessarily an average) - it is more understandable as a “minimum value” rather than the average.

- **Pull up field:** This was done midway through assignment 3. Prior to the final design, the multipleMonitorTableModel contained an instance variable of a multipleTypeCellRenderer. As this was not necessary, it was pulled up into the parent class. The parent class of cell renderers was used instead. Doing so clears up unnecessary code that would clutter up.

Assumptions:

- No measurement records from the server will have a value of zero. It is currently being used as an initial value to test if a record has been retrieved/modified yet

Class Diagram


Constants
+ MeasurementType: enum
<<abstract>> MeasurementType
~ fhirCode: String
~ name: String
~ monitorredSubjects: ArrayList<PatientSubject>
~ childCodes: ArrayList<String>
~ childTypeNames: ArrayList<String>
~ childTypes: ArrayList<Constants.Enum>
~ minimumValues: ArrayList<Integer>
~ type: Enum
- measurementAverage: Double
- measurementTotal: double
~ numberStoredRecordings: int
+ updateAverage(): void
+ getValidMonitored(): int
0..*
0..*
Cholesterol
BloodPressure
MeasurementRecording
- type: MeasurementType
- value: BigDecimal
- date: Date
- childValues: ArrayList<BigDecimal>
+ cloneRecording(newRecording: MeasurementRecording): void
0..*
<<abstract>> Observer
update( ): void
0..*
1
<<abstract>> Subject
- observers: ArrayList<Observers>
+ attach(Observer): void
+ detach(Observer): void
+ notifyObservers( ): void
PatientRecord
- id: String
- firstName: String
- lastName: String
- gender: String
- birthdate: String
- address: String
- measurementRecordings, listLastRecordings: ArrrayList<MeasurementRecording>
+ initialiseMeasurements(newList: ArrayList<MeasurementType>): void
+ addMeasurementObject(listTypes: ArrayList<MeasurementType>): void
+ getMeasurement(type: MeasurementType): MeasurementRecording
+ setMeasurementRecording(value: BigDecimal, newDate: Date, newType: MeasurementType)
+ resetRecording(newType: MeasurementType): void
+ pushNewRecordingHistory(newType: MeasurementType): void
0..*
TextualObserver
- observerSubject: PatientSubject
- lastState: Date
- monitorredData: HistoricalTableModel
- type: MeasurementType
MeasurementObserver
- observingSubject: PatientSubject
- lastState : Date
- monitorredData: MonitorTableModel
- type: MeasurementType
PatientSubject
- state : PatientRecord
- server: Server
- active: boolean
+ updateMeasurementValue(type: MeasurementType): void
0..*
0..*
0..*
<<abstract>> TimerTask
PeriodicMeasurementCall
+ frequency: int
- type: MeasurementType
- patientSubjectList: ArrayList<PatientSubject>
+ run(): void
Timer
HistoricalTableModel
- historicalType: MeasurementType
- childType: Enum
- monitoredLastRecordings: ArrayList<String>
- subjects: ArrayList<PatientSubject>
- recordingChartData: XYSeriesCollection
- graphMonitor: boolean
+ addPatient(patient: PatientSubject): void
+ removePatient(patientId: String): void
+ updateHistory(lastRecordings: ArrayList<MeasurementRecord>, newRecord: PatientRecord): boolean
+ createDataSet(): XYSeriesCollection
+ lastRecordingsToString(lastRecordings: ArrayList<MeasurementRecording>): String
+ addChart(chartData: XYSeriesCollection): void
FhirContext
<<abstract>> AbstractTableModel
<<interface>> IGenericClient
Server
- client: IGenericClient
- context: FhirContext
- serverBase: String
- entriesPerPage: int
+ retrievePractitionerIDs(identifier: String): ArrayList<String>
+ addPracIDsToList(newBundle: Bundle, newList: ArrayList<String>
+ retrievePractitionerPatients(identifier: String): ArrayList<PatientRecord>
+ retrievePatient(id: String): PatientRecord
+ retrieveMeasurment(patient: PatientRecord, type: MeasurementType): MeasurementRecording
+ addPatientToList(identifierList: ArrayList<String>, patientList: ArrayList<PatientRecord>, bundle: Bundle): void
PractitionerRecord
- practitionerIdentifier: String
- practitionerIDs: ArrayList<String>
- practitionerPatients: ArrayList<PatientRecord>
- server: Server
+ retrievePractitionerPatients(): void
<<abstract>> PatientTableModel
~ monitorredData: ArrayList<ArrayList<String>>
~ columnNames: ArrayList<String>
~ monitoredPatientID: ArrayList<String>
~ monitoredPatientNames: ArrayList<String>
+ createTable(): void
+ clearDataValues(): void
Controller
- myModel: Model
- myTimer: Timer
- myView: View
- server: Server
- myPeriodicCholesterol, myPeriodicBP: PeriodicMeasurementCall
- allTypes, cholType, bpType: ArrayList<MeasurementType>
+ initController(): void
+ initModel(): void
+ storePracIdentifier(): void
+ updatePatientList(retrievePatientsFromServer: boolean): void
+ monitorSelectedPatients(newType: MeasurementType): void
+ stopMonitorSelectedPatients(newType: MeasurementType): void
+ scheduleMonitor(): void
+ updateFrequency(): void
+ displaySelectedPatients(patientIndex: int, newType: MeasurementType): void
+ updateMinValue(newColumnIndex: int, newType: MeasurementType): void
+ displayXYgraph(chartType: MeasurementType): void
- displayChart(chartType: MeasurementType): void
- displayHighPatients(textualType: MeasurementType): void
XYSeriesCollection
DefaultTableCellRenderer
MonitorTableModel
~ monitoredMeasurementSubjects: ArrayList<ArrayList<PatientSubject>>
~ measurmentMinWatcher: MeasurementCellRenderer
~ observedCellColour: Color
~ measurementData: DefaultCategoryDataset
+ addMonitorPatient(patientID: String, patientName: String): boolean
~ addComponentNames(): void
+ removePatientFromTable(newPatientIndex: int): void
+ addMeasurementType(newType: MeasurementType): void
+ updateMeasurements(newPatient: PatientRecord, newMeasurement: MeasurementRecording): void
+ setMinColouredValue(oldValue: double, column: int): void
+ addChart(chartData: DefaultCategoryDataset): void
Model
- cholMonitorTableModel: MonitorTableModel
- bpMonitorTableModel: MonitorTableModel
- systolicHistTableModel: HistoricalTableModel
- patientListModel: DefaultListModel
- server: Server
- loggedInPractitioner: PractitionerRecord
- storedIdentifiers: ArrayList<String>
- storedPractitioners: ArrayList<PractitionerRecord>
- allTypes, cholType, bpType: ArrayList<MeasurementType>
+ createPractitioner(newIdentifier: String): PractitionerRecord
+ updatePatientNameList(): void
+ addTypeToSystem(newType: MeasurementType): void
MeasurementCellRenderer
- measurementAverage: int
- column: int
+ updateMinColouredValue(newValue: double): void
+ updateMinColouredValue(newValue: double, column: int): void
+ first(): void
+ next(): void
+ hasNextColumn(): boolean
DefaultCategoryDataset
DefaultListModel
Main
View
JFrame
java.swing
JFreeChart
MultipleTypeCellRenderer
- minValList: ArrayList<Double>
- listColumns: ArrayList<Integer>
- currentIndex: int
+ addColumnIndex(numberColumns: int): void
- checkValue(): void
MultipleMonitorTableModel
- type: MeasurementType
+ addMeasurementType(newType: MeasurementType): void
~ addComponentNames(): void
+ updateMeasurements(newPatient: PatientRecord, newMeasurement: MeasurementRecording)
+ setMinColouredValue(newValue: double, column: int): void