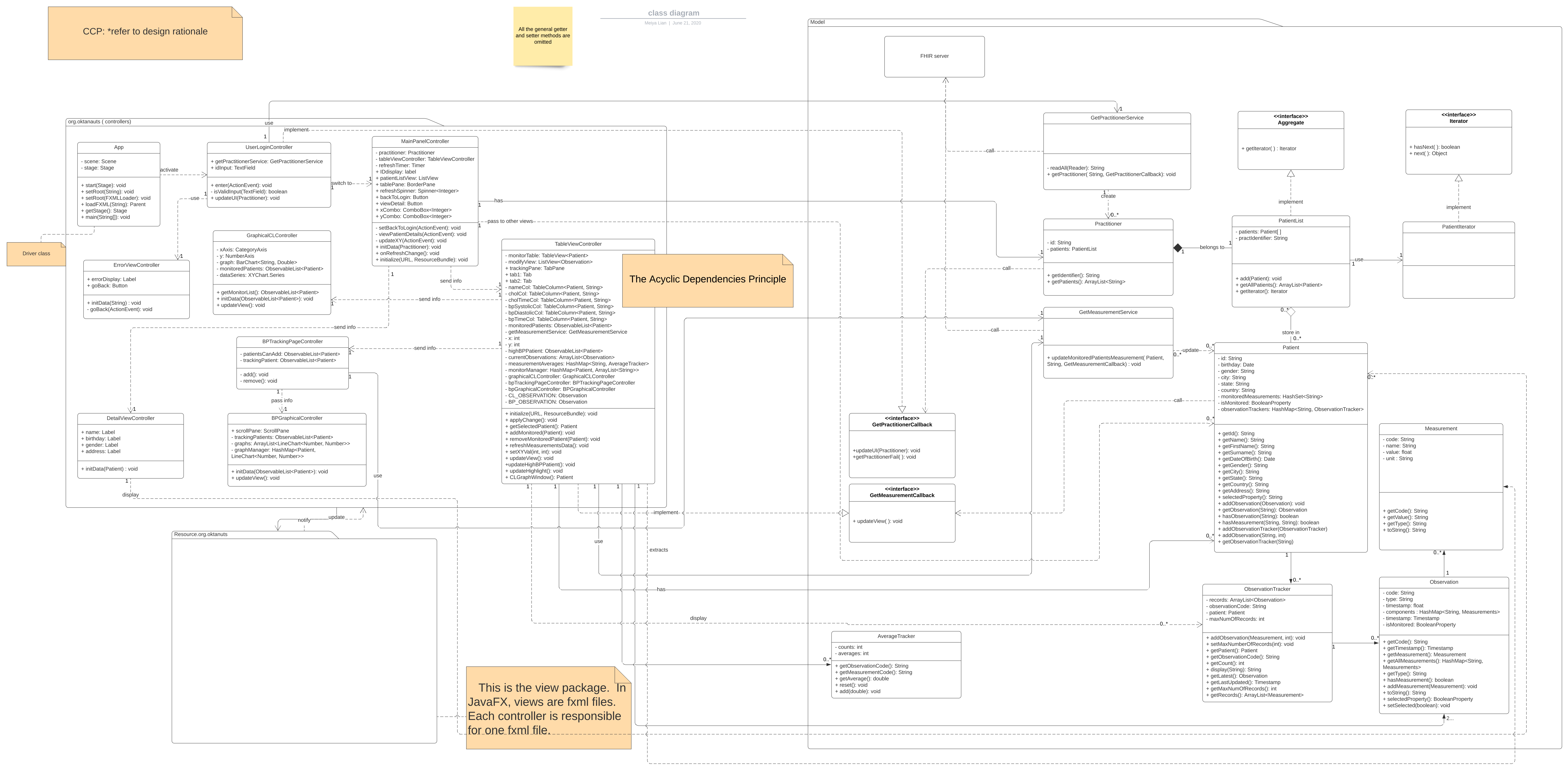
CCP: *refer to design rationale
All the general getter and setter methods are omitted
class diagram
Meiya Lian | June 21, 2020
Driver class
org.oktanauts (controllers)
App
- scene: Scene
- stage: Stage
+ start(Stage): void
+ setRoot(String): void
+ setRoot(FXMLLoader): void
+ loadFXML(String): Parent
+ getStage(): Stage
+ main(String[]): void
activate
UserLoginController
+ getPractitionerService: GetPractitionerService
+ idInput: TextField
+ enter(ActionEvent): void
- isValidInput(TextField): boolean
+ updateUI(Practitioner): void
use
implement
switch to
MainPanelController
- practitioner: Practitioner
- tableViewController: TableViewController
- refreshTimer: Timer
+ IDdisplay: label
+ patientListView: ListView
+ tablePane: BorderPane
+ refreshSpinner: Spinner<Integer>
+ backToLogin: Button
+ viewDetail: Button
+ xCombo: ComboBox<Integer>
+ yCombo: ComboBox<Integer>
- setBackToLogin(ActionEvent): void
- viewPatientDetails(ActionEvent): void
- updateXY(ActionEvent): void
+ initData(Practitioner): void
+ onRefreshChange(): void
+ initialize(URL, ResourceBundle): void
has
pass to other views
send info
ErrorViewController
+ errorDisplay: Label
+ goBack: Button
+ initData(String) : void
- goBack(ActionEvent): void
GraphicalCLController
- xAxis: CategoryAxis
- y: NumberAxis
- graph: BarChart<String, Double>
- monitoredPatients: ObservableList<Patient>
- dataSeries: XYChart.Series
+ getMonitorList(): ObservableList<Patient>
+ initData(ObservableList<Patient>): void
+ updateView(): void
BPTrackingPageController
- patientsCanAdd: ObservableList<Patient>
- trackingPatient: ObservableList<Patient>
- add(): void
- remove(): void
pass info
DetailViewController
+ name: Label
+ birthday: Label
+ gender: Label
+ address: Label
+ initData(Patient) : void
display
BPGraphicalController
+ scrollPane: ScrollPane
- trackingPatients: ObservableList<Patient>
- graphs: ArrayList<LineChart<Number, Number>>
- graphManager: HashMap<Patient, LineChart<Number, Number>>
+ initData(ObservableList<Patient>): void
+ updateView(): void
TableViewController
- monitorTable: TableView<Patient>
- modifyView: ListView<Observation>
+ trackingPane: TabPane
+ tab1: Tab
+ tab2: Tab
- nameCol: TableColumn<Patient, String>
- cholCol: TableColumn<Patient, String>
- cholTimeCol: TableColumn<Patient, String>
- bpSystolicCol: TableColumn<Patient, String>
- bpDiastolicCol: TableColumn<Patient, String>
- bpTimeCol: TableColumn<Patient, String>
- monitoredPatients: ObservableList<Patient>
- getMeasurementService: GetMeasurementService
- x: int
- y: int
- highBPPatient: ObservableList<Patient>
- currentObservations: ArrayList<Observation>
- measurementAverages: HashMap<String, AverageTracker>
- monitorManager: HashMap<Patient, ArrayList<String>>
- graphicalCLController: GraphicalCLController
- bpTrackingPageController: BPTrackingPageController
- bpGraphicalController: BPGraphicalController
- CL_OBSERVATION: Observation
- BP_OBSERVATION: Observation
+ initialize(URL, ResourceBundle): void
+ applyChange(): void
+ getSelectedPatient(): Patient
+ addMonitored(Patient): void
+ removeMonitoredPatient(Patient): void
+ refreshMeasurementsData(): void
+ setXYVal(int, int): void
+ updateView(): void
+updateHighBPPatient(): void
+ updateHighlight(): void
+ CLGraphWindow(): Patient
The Acyclic Dependencies Principle
use
extracts
notify
update
Resource.org.oktanauts
This is the view package. In JavaFX, views are fxml files. Each controller is responsible for one fxml file.
Model
FHIR server
GetPractitionerService
- readAll(Reader): String
+ getPractitioner(String, GetPractitionerCallback): void
call
create
Practitioner
- id: String
- patients: PatientList
+ getIdentifier(): String
+ getPatients(): ArrayList<String>
belongs to
<<interface>> Aggregate
+ getIterator( ) : Iterator
<<interface>> Iterator
+ hasNext( ): boolean
+ next( ): Object
implement
PatientList
- patients: Patient[ ]
- practIdentifier: String
+ add(Patient): void
+ getAllPatients(): ArrayList<Patient>
+ getIterator(): Iterator
PatientIterator
store in
GetMeasurementService
+ updateMonitoredPatientsMeasurement(Patient, String, GetMeasurementCallback) : void
update
Patient
- id: String
- birthday: Date
- gender: String
- city: String
- state: String
- country: String
- monitoredMeasurements: HashSet<String>
- isMonitored: BooleanProperty
- observationTrackers: HashMap<String, ObservationTracker>
+ getId(): String
+ getName(): String
+ getFirstName(): String
+ getSurname(): String
+ getDateOfBirth(): Date
+ getGender(): String
+ getCity(): String
+ getState(): String
+ getCountry(): String
+ getAddress(): String
+ selectedProperty(): String
+ addObservation(Observation): void
+ getObservation(String): Observation
+ hasObservation(String): boolean
+ hasMeasurement(String, String): boolean
+ addObservationTracker(ObservationTracker)
+ addObservation(String, int)
+ getObservationTracker(String)
<<interface>> GetPractitionerCallback
+updateUI(Practitioner): void
+getPractitionerFail( ): void
<<interface>> GetMeasurementCallback
+ updateView( ): void
Measurement
- code: String
- name: String
- value: float
- unit : String
+ getCode(): String
+ getValue(): String
+ getType(): String
+ toString(): String
ObservationTracker
- records: ArrayList<Observation>
- observationCode: String
- patient: Patient
- maxNumOfRecords: int
+ addObservation(Measurement, int): void
+ setMaxNumberOfRecords(int): void
+ getPatient(): Patient
+ getObservationCode(): String
+ getCount(): int
+ display(String): String
+ getLatest(): Observation
+ getLastUpdated(): Timestamp
+ getMaxNumOfRecords(): int
+ getRecords(): ArrayList<Measurement>
Observation
- code: String
- type: String
- timestamp: float
- components : HashMap<String, Measurements>
- timestamp: Timestamp
- isMonitored: BooleanProperty
+ getCode(): String
+ getTimestamp(): Timestamp
+ getMeasurement(): Measurement
+ getAllMeasurements(): HashMap<String, Measurements>
+ getType(): String
+ hasMeasurement(): boolean
+ addMeasurement(Measurement): void
+ toString(): String
+ selectedProperty(): BooleanProperty
+ setSelected(boolean): void
AverageTracker
- counts: int
- averages: int
+ getObservationCode(): String
+ getMeasurementCode(): String
+ getAverage(): double
+ reset(): void
+ add(double): void

Design Rationale

The Oktanauts - Meiya Lian and Nicholas Chmielewski

“Classes that change together, belong together.”[Mar2000] Controller, view(fxml files) and models are in separate packages. This is because referring to MVC, the change inside controllers and views should not affect the model classes. And, if the model packages are modified, there is a high chance to modify the controller classes. In accordance with simplicity and limited scope of our project, there was no need to further break down into smaller packages.

In our initial design, we had designed our system under an MVC architecture, which allowed us to add multiple views to the system without modifying most of the code. For the functionalities in this assignment, 3 new controllers class had been added. (**GraphicalCLController**, **BPTrackingPageController** and **BPGraphicalController**). Strictly speaking, only the **BPTrackingPageController** is a ‘true’ controller class since it will respond to user input (user selecting patients to track their history blood pressure). The other two classes are view classes as they are purely used to display messages. However, in Javafx, it is a convention to see views as fxml files, and all the code related to that fxml file is said to be the “controller” of that file. Hence, all of these classes are listed as controllers.

In our original design, we implemented a Measurement class which allowed us to adequately fulfil the required functionality. However with the new requirements including blood pressure measurements, which was broken down into systolic and diastolic, but inside of the same observation, we needed to update the design of our system. Hence we decided to implement an Observation class which allowed for the required bundling. Our Measurement class from A2 was left relatively unchanged, but now we have a dynamic container for it which supported a single API call getting the information for both measurements as well as a shared timestamp. This meant that we didn’t need to entirely start over from scratch, keeping the majority of our code the same.

We decided to use the Observation class as in the FHIR API database measurement records are stored as Observations, which can be broken down into multiple different component measurements. The original Cholesterol measurement existed by itself inside of an Observation shell which in fact used the exact same LOINC code. If we had chosen to keep our code the same, multiple calls would need to be done to get the desired data, as the solution originally treated the get measurement services as measurement level.

To fulfil the functionality of monitoring the previous blood pressure of a patient, we added the **ObservationTracker** class to the model class. The tracker will store the latest n records of the observation of a patient. When the process of monitoring a patient 's observation begins, the tracker will be told to only store the single most recent observation available. When the user selects this patient to track their blood pressure, the value of n will be updated to 5 and the system will fetch the data accordingly. This class has applied OCP as the number of monitored records can be changed dynamically. This flexibility supports future extensions to existing functionality without needing to update the base code of the class.

Through the OCP we have left our overall system open to easy extension. For example, the highlighting functionality and table composition have been written in such a way that instead of having to repeat the required code, simple observation blueprints of the observation format can be provided to allow extension when adding new measurements to the table. This is also reflected in the fact that the entire solution has been designed to allow for dynamic changes in terms of measurements, highlighting, updating, etc, rather than explicity hard-coding in the specific measurements that are required. This means that making additions to the scope of the system is relatively simple. For example, changing from Cholesterol level to another measurement in terms of the table view consists of only changing a few lines of code, rather than an underlying system which would break unless it exists in its original state.

An average tracker class has also been included to simplify the code present in the table controller, as well as to reduce the need for outside dependencies. This also means that the behaviour of how highlighting/averages are calculated can be changing without necessarily having to update the table controller.

The Iterator pattern is used for an aggregate class **PatientList** in the system. Although it had not been used in any functionality in the current system, we decide to keep this pattern by following OCP. With this pattern, future functionalities such as filtering patients with age/gender/other attributes can be implemented easily. The use of the **PatientIterator** followed LSP to ensure that when through polymorphism it is treated as an iterator that it does not impede on the behaviours on the Iterator class that it inherits.

DIP and ISP are used in the system referring to this principle. Interfaces such as GetPractitionerCallback and GetMeasurementCallbak are implemented to invert the dependency between the Controller package and the Model package. In this way, any changes to the controller classes will not affect the model package, as the model classes interact with the controller classes via interfaces.

Reference

[Mar2000] Robert C. Martin, “Design Principles and Design Patterns”, 2000. Available on-line from Object Mentor: http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf

Docs.oracle.com. 2020. *Mastering FXML: Why Use FXML | Javafx 2 Tutorials And Documentation*. [online] Available at: <https://docs.oracle.com/javafx/2/fxml_get_started/why_use_fxml.htm> [Accessed 14 June 2020].