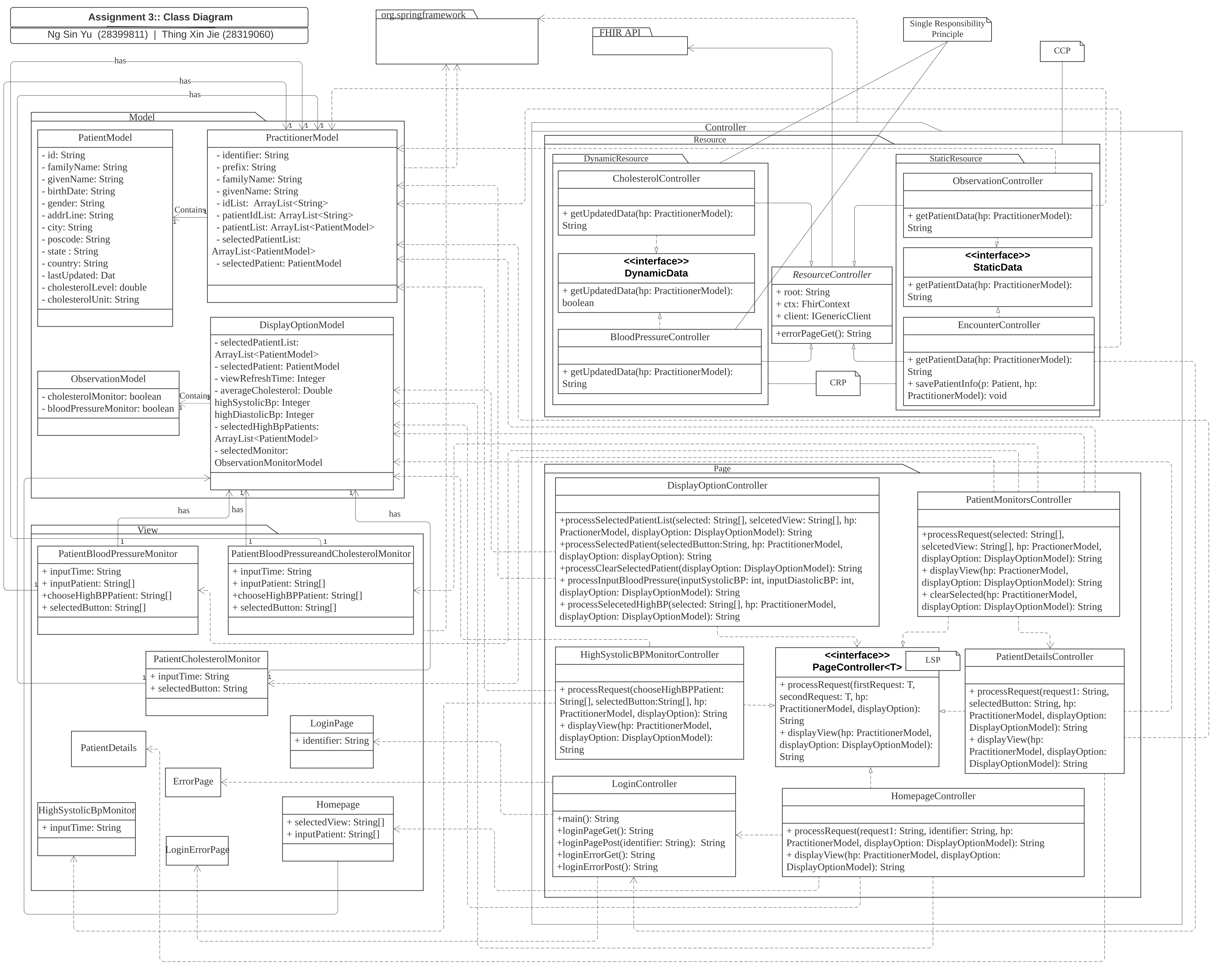
Assignment 3:: Class Diagram

Ng Sin Yu (28399811) | Thing Xin Jie (28319060)


org.springframework
FHIR API
Single Responsibility Principle
CCP
has
has
has
Model
PatientModel
- id: String
- familyName: String
- givenName: String
- birthDate: String
- gender: String
- addrLine: String
- city: String
- poscode: String
- state : String
- country: String
- lastUpdated: Dat
- cholesterolLevel: double
- cholesterolUnit: String
Contains
PractitionerModel
- identifier: String
- prefix: String
- familyName: String
- givenName: String
- idList: ArrayList<String>
- patientIdList: ArrayList<String>
- patientList: ArrayList<PatientModel>
- selectedPatientList: ArrayList<PatientModel>
- selectedPatient: PatientModel
ObservationModel
- cholesterolMonitor: boolean
- bloodPressureMonitor: boolean
Contains
DisplayOptionModel
- selectedPatientList: ArrayList<PatientModel>
- selectedPatient: PatientModel
- viewRefreshTime: Integer
- averageCholesterol: Double
highSystolicBp: Integer
highDiastolicBp: Integer
- selectedHighBpPatients: ArrayList<PatientModel>
- selectedMonitor: ObservationMonitorModel
has
has
has
View
PatientBloodPressureMonitor
+ inputTime: String
+ inputPatient: String[]
+chooseHighBPPatient: String[]
+ selectedButton: String[]
PatientBloodPressureandCholesterolMonitor
+ inputTime: String
+ inputPatient: String[]
+chooseHighBPPatient: String[]
+ selectedButton: String[]
PatientCholesterolMonitor
+ inputTime: String
+ selectedButton: String
LoginPage
+ identifier: String
PatientDetails
ErrorPage
Homepage
+ selectedView: String[]
+ inputPatient: String[]
HighSystolicBpMonitor
+ inputTime: String
LoginErrorPage
Controller
Resource
DynamicResource
CholesterolController
+ getUpdatedData(hp: PractitionerModel): String
<<interface>> DynamicData
+ getUpdatedData(hp: PractitionerModel): boolean
BloodPressureController
+ getUpdatedData(hp: PractitionerModel): String
ResourceController
+ root: String
+ ctx: FhirContext
+ client: IGenericClient
+errorPageGet(): String
CRP
StaticResource
ObservationController
+ getPatientData(hp: PractitionerModel): String
<<interface>> StaticData
+ getPatientData(hp: PractitionerModel): String
EncounterController
+ getPatientData(hp: PractitionerModel): String
+ savePatientInfo(p: Patient, hp: PractitionerModel): void
Page
DisplayOptionController
+processSelectedPatientList(selected: String[], selcetedView: String[], hp: PractionerModel, displayOption: DisplayOptionModel): String
+processSelectedPatient(selectedButton:String, hp: PractitionerModel, displayOption: displayOption): String
+processClearSelectedPatient(displayOption: DisplayOptionModel): String
+ processInputBloodPressure(inputSystolicBP: int, inputDiastolicBP: int, displayOption: DisplayOptionModel): String
+ processSelecetedHighBP(selected: String[], hp: PractitionerModel, displayOption: DisplayOptionModel): String
PatientMonitorsController
+processRequest(selected: String[], selcetedView: String[], hp: PractionerModel, displayOption: DisplayOptionModel): String
+ displayView(hp: PractionerModel, displayOption: DisplayOptionModel): String
+ clearSelected(hp: PractitionerModel, displayOption: DisplayOptionModel): String
HighSystolicBPMonitorController
+ processRequest(chooseHighBPPatient: String[], selectedButton:String[], hp: PractitionerModel, displayOption): String
+ displayView(hp: PractitionerModel, displayOption: DisplayOptionModel): String
<<interface>> PageController<T>
LSP
+ processRequest(firstRequest: T, secondRequest: T, hp: PractitionerModel, displayOption): String
+ displayView(hp: PractitionerModel, displayOption: DisplayOptionModel): String
PatientDetailsController
+ processRequest(request1: String, selectedButton: String, hp: PractitionerModel, displayOption: DisplayOptionModel): String
+ displayView(hp: PractitionerModel, displayOption: DisplayOptionModel): String
LoginController
+main(): String
+loginPageGet(): String
+loginPagePost(identifier: String): String
+loginErrorGet(): String
+loginErrorPost(): String
HomepageController
+ processRequest(request1: String, identifier: String, hp: PractitionerModel, displayOption: DisplayOptionModel): String
+ displayView(hp: PractitionerModel, displayOption: DisplayOptionModel): String

Model-View-Controller (MVC) Architecture Pattern

- Controllers perform necessary actions, passing the processed results into Models and Views will retrieve data to be displayed from Models, making acyclic dependencies possible to be achieved. Since Models do not need to know anything about the Controller and View and Views only depend on Models, modifications can be easily implemented.
- MVC architecture makes the resources and business logic easy to maintain. The reusability of Controllers for multiple Views makes MVC a great choice in meeting the requirements of having multiple views on similar data.
- For example, users are able to select the type of monitor (cholesterol and/or blood pressure), and different views will be displayed based on their selection. However, the Controller which performs the backend processes (i.e. processSelectedPateintList in DisplayOptionController) is performing similar functionality to update the details of selected patients based on the list of selected patients. MVC eases this case where the same route (function in controller) is called but different views are returned.
- We used MVC architecture in previous assignment, and since MVC allows Model, View and Controller performs their tasks without interfering each other, we can fulfill new requirements easily by adding new models for new requirements such as ObservationMonitorModel which stores the type of monitor selected by users, new controllers to process data accordingly and new Views to render pages based on the type of monitor selected by users. All these were done without changing much on the processes of previous requirements, since we are focusing more on adding functionalities by extending initial design instead of modifying the implementations of previous requirements to cater with new requirements

Single Responsibility Principle (SRP)

- Each controller is focusing their own functionality; the tasks related to certain aspects are written within the corresponding controller.
- We used Interface Segregation Principle (ISP) in previous design, where the ObservationController was implementing both DynamicData and StaticData interface where StaticData is used to retrieve data once to initialise patients' cholesterol level/ blood pressure in the homepage (where practitioners can select patients to monitor) and DynamicData is to retrieve updated data at interval.
- However in new requirements, there are more than one type of data that needs to be updated at interval, which are cholesterol and blood pressure, hence more than one getUpdatedData function needs to be performed. We refactored by moving the getUpdatedData function to new Controllers (CholesterolController and BloodPressureController), which are responsible for their own data. These controllers only implements DynamicData since they are only required to retrieve updated data and ObservationController is refactored to only implementing StaticData which will retrieve both cholesterol and blood pressure readings from the server. The retrieval of data is not written in CholesterolController and BloodPressureController to reduce the traffic of fetching data from the same module in the server.

Liskov Substitution Principle (LSP)

- LSP is used on PageControllers, since Controllers for the pages are performing similar tasks, which are processing the request parameters from the user interface on Views (processRequest function) and rendering the View page (displayView function). The use of LSP allows easy extension of the application, where when there are extra requirements needed to display different pages, we can simply implements the PageController interface and perform similar actions; by overriding processRequest function to process the request retrieved from View and overriding displayView function to render view since both functions are behaviour similarly on every derived classes.

Common Closure Principle (CCP) and Common Reuse Principle(CRP)

- Based on CCP, all controllers are separated into two main packages, page controllers used to render the view and resource package used to fetch relevant data from the database. But in this assignment, we further separate the resource into two different packages which are Dynamic Resource and Static Resource. Since we had adhered to CCP from last assignments, further modification to this package does not affect the functionality and design of the other package. This is because each package has their own responsibility and minor modification will not affect each other.
- The main reason for doing this is because Static Resource's controllers will be only used for the first time data fetching while Dynamic Resource is used for fetching updated data. Since we have multiple controllers that are used to fetch updated data, splitting them into an independence package increases its maintainability. However, CRP may make us have a lot of small packages if we abuse it. Hence, it is a challenge for us to seek a balance in between CCP and CRP if there are any other further new requirements added to it.

Acyclic Dependencies Principle

- Adopting MVC design patterns had brought us to separate our component to three major packages which controllers, model and view. Controllers which used to process data will depend on the model to store all the relevant processed data,then controllers will also depend on View to render all the corresponding webpage. For the dependency between View and Model, the view will depend on the data that is stored in the model to display accurate data.
- So, there will always be a one way dependencies between the Model, View and Controller. These then help us to eliminate the Morning after syndrome as changing the view will not affect the controller or even the model. Hence, when adding multiple graph visualisations in the view for this assignment, it will not affect the controller nor the model.