MVC + Observer

In terms of the design used for the front end implementation of the system, nothing much has changed, the use of the Model-View-Controller design pattern along with the Observer implementation has proved greatly useful as additional Views (BPObservationView, LineChartView) can be added easily into the current system.

Since the current design pattern of the frontend is deemed as robust and easily extensible, the design is kept the same, however refactoring of the view and controller classes were done whereby the dependencies of the view classes on additional model classes were removed as the controller classes could simply pass the data to be displayed. This is to ease testing of functionalities as we can narrow the scope of testing to mainly controller classes if the data passed is wrong or view class if the data is not being displayed properly. Views classes for line graph and bar chart are also designed to be capable of being reused to display any types of patient information without major refactoring. This greatly improve the extensibility and flexibility of the display functionality.

As for the Backend package, nothing much has changed as well because of the ease of extensibility resulting from previous design patterns and principles implementation. The layer of abstraction we had implemented in assignment 2 for this package allows us to easily extend and use the base classes (AbstractMonitor, PatientInfo) to achieve the new requirements without the need to refactor any other classes. For example, to represent and manage the blood pressure values of patients (or any new type of patient information), a new class (PatientInfoBP) is created just by extending PatientInfo and overriding the base method updateData() to retrieve the blood pressure values from the FHIR server. These new classes created also adhere to the design patterns and principles we had implemented earlier. Furthermore, to adhere to good coding practice, every class' attributes are kept private.

We did refactor some class and method names to improve readability as well as changing some global variables to local or even removing them, to minimise any unnecessary dependencies. For example, previously AbstractMonitor had a dependency on Practitioner. We removed it and instead, let AbstractController manage monitors and have dependency on them. We also created two new monitor classes (CholesterolMonitor and BloodPressureMonitor) to allow MonitoredController to differentiate the different types of monitor. This allows users to start or stop observing either of these values during runtime and also increase the ease of implementation of any potential new type of monitors in the future.

To address some of the comments in assignment two results, User class exists as a base class for any potential new users that might interact with the system. Each new type of user will have a unique encounter relationship with the patients, for example, a nurse. The reason why we did not merge the Backend package with the Model package is because we do not want the Model package to have a dependency on Server package. We grouped the classes that interact with the FHIR server and JSON data into Backend (will also be reused together) to ensure that only limited classes can communicate with FHIR server to achieve data hiding and data integrity, while also ensuring Common Reuse Principle is practiced.

Reference

Assignment 3 Source Code

Assignment 3 Class Diagram