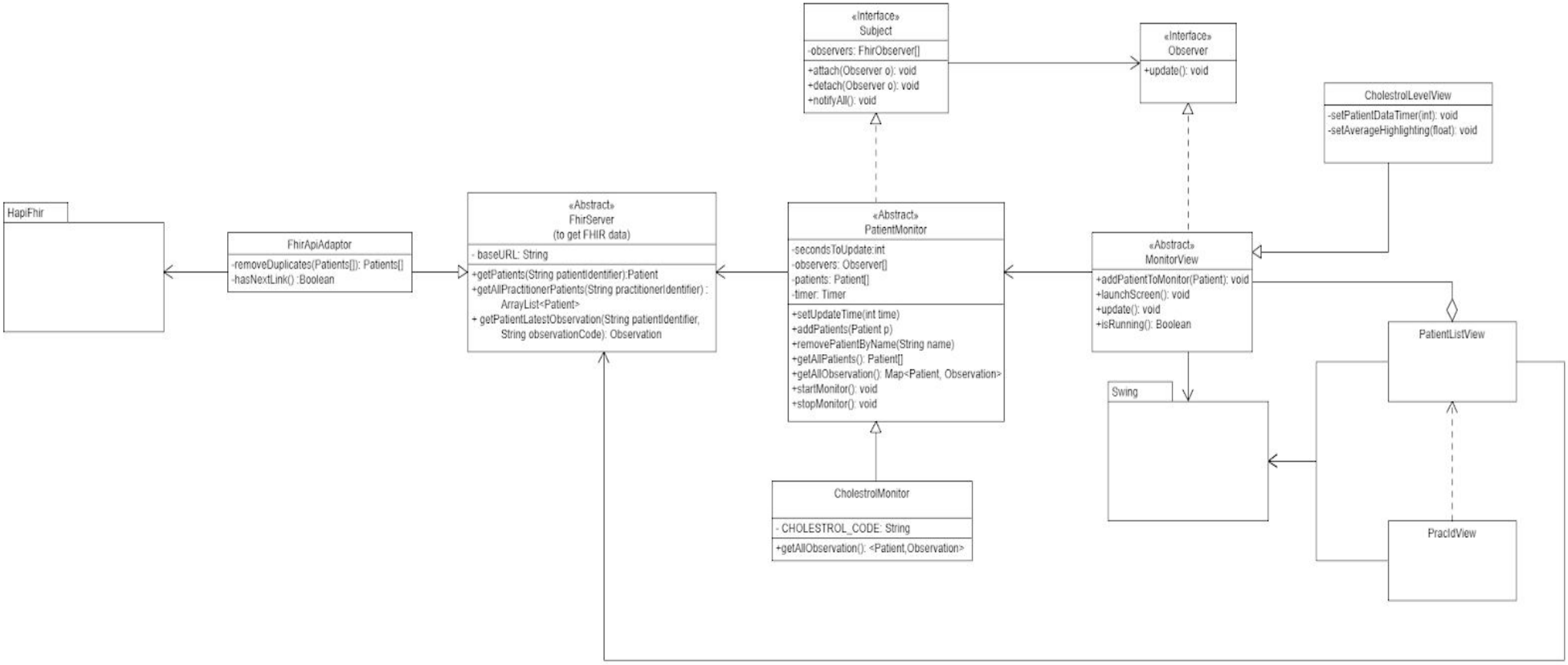
FIT3077 Assignment 2 : UML Class Diagram

Team Indomie: Richard Pranjatno and Jason Agustino



FIT 3077 A2 Design Rationale

We implemented the Observer design pattern for our FHIR app system because we wanted to “observe” the data provided by the FHIR server, without the server knowing any implementation of our Observer object. This means that the server is not dependent on our observers, and all it does is notify each observer of an update. This makes the subject and observers loosely coupled. Both don't know each other's implementation, and an observer can be detached and attached to a subject independently.

This design allows multiple observers to be updated by the Subject whenever there is change in the Subject state, and having the abstract Observer class allows extension of different types of views. In the case of our system, currently we are only monitoring cholesterol level, however with the abstract observer class we can then extend to add more parameters such as blood pressure, blood sugar level, etc.

The following are some examples of how we used the Observer design pattern in the implementation of our FHIR app system.

1. The abstract Subject class we have is the FhirServer class which is responsible for getting all the patients of a practitioner and their cholesterol levels.
2. The concrete Subject class we have implemented is the PatientMonitor class, which implements the abstract Subject class FhirServer. This concrete Subject is responsible for adding and removing observers, as well as updating/notifying the observers.
3. We extended the class PatientMonitor, to monitor a patient's cholesterol level by creating the class CholesterolMonitor, it has the implementation to gather the patient’s cholesterol data.
4. PatientListView class is a concrete Observer class, which updates a list of the practitioner’s patients by attaching itself to a concrete Subject class.
5. CholesterolLevelView class is also another concrete Observer class, which updates the list of the patient’s cholesterol level by attaching itself to a concrete Subject class.

In addition to the Observer design pattern, we also implemented the Adaptor design pattern for our FhirServer abstract class. The FhirApiAdapter class solves the problem of the HapiFhirApi interface not matching with the interface of the PatientMonitor class. The concrete FhirSubject class which is our PatientMonitor class, does not need to know the implementation of the HapiFhirApi. All it needs to know is what data to obtain, and not how to obtain the data through api calls. The FhirApiAdapter class reduces dependency between the PatientMonitor class and the actual API server calls. The introduction of an adapter class allows us to create more ConcreteSubject classes that request data to the HapiFhirApi through the adapter class. This means that we can have more data about a particular practitioner and not just based on Encounter or Observations, without the Subject class knowing the actual API implementation.