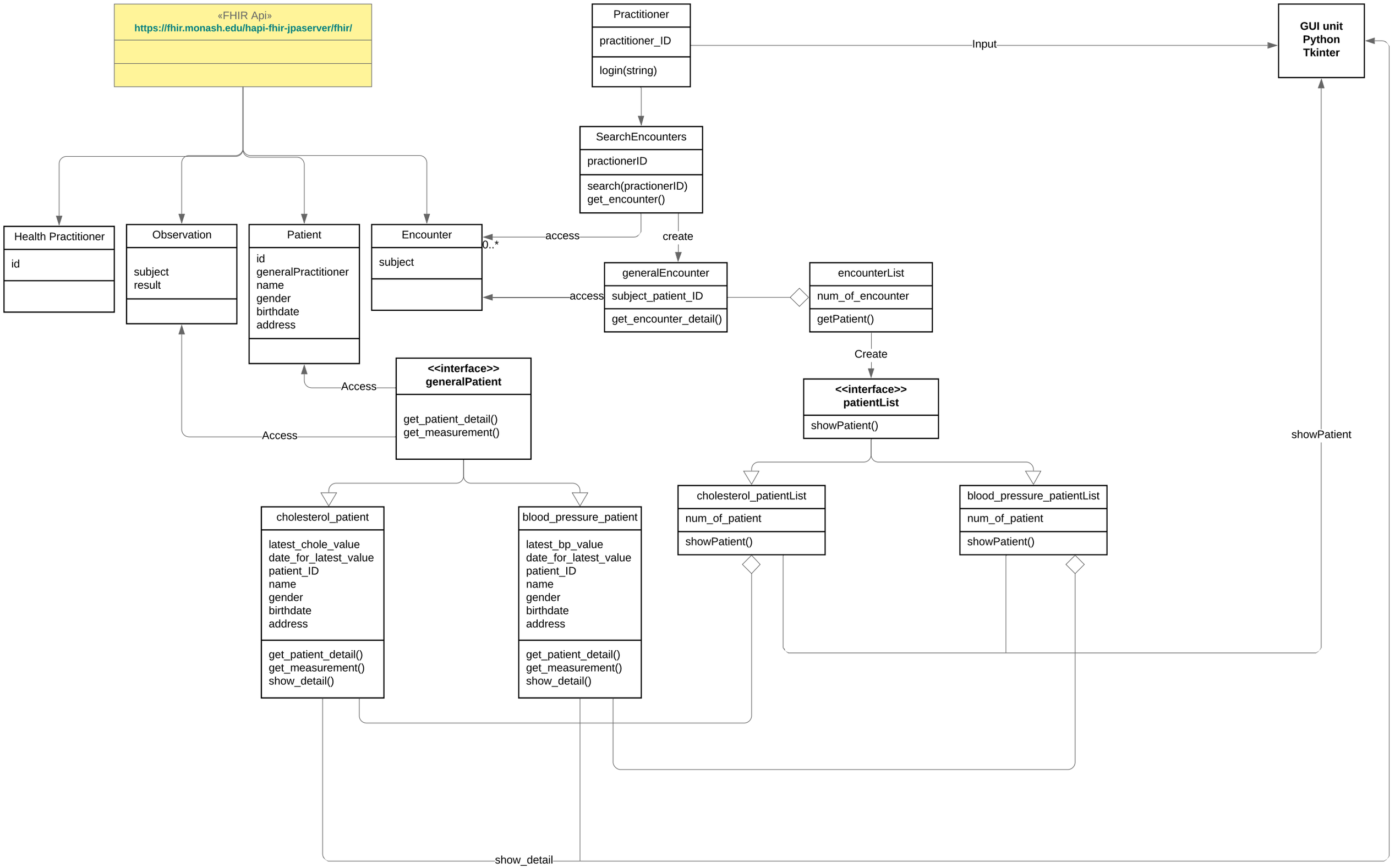
FIT3077 Assignment 3

Junxuan Liang | May 29, 2020



Design principle

- Open/Close principle
    - The health practitioner system is divided into a few different modules and classes. When extension is needed the source code is not changing therefore the original functionalities are not going to be affected. Only the extension code will be needed to implement the extension requirements.
- Liskov Substitutability Principle
    - In the health practitioner system, there is a patient class which include all the patient details needed by the system and there is a practitioner class which include an array of patient monitoring. Due to LSP, extension on monitoring patients with blood pressure data can be achievable by only making extension on the patient class without any changes on the practitioner class or the main program.
- Package cohesion principle
    - The practitioner class act as a stand-alone package in the system. When extension on monitoring patient with blood pressure is required, the remove and add patients to monitor function can be easily maintained as we can just call a method in the practitioner class to achieve that.
- The Acyclic Dependencies Principle
    - There are a few files included in the health practitioner system, and there is an acyclic direction relationship. Due to that, implementing extension on the system will be easier as if there was a cyclical direction formed between the files any extension or modification on one file will leads to modification on all files in the system.
- Stable dependencies principle
    - Other than the main UI file in the system, the other file uses no more than two other files in the system for the project. Therefore, when there is extension code add on the program there is no need to make a huge modification on files other than the main program.

Design pattern

- Iterator
    - The iterator allows the program to loop through all the encounters under a specific practitioners and grab patients from it. Using the iterator for reusing can make the adding functionality of monitoring patient with blood pressure easier as the system is already getting patients based on the cholesterol measurement.

Refactoring methods

- Moving features between classes
    - Before start implementing the new requirements, refactoring is done to move some of the functions into different files and classes. In that way, developers can easily find out where the extensions are needed without dealing with a large piece of code in one single file.
- Organizing data
    - Before implementing new requirements, practitioner and patient class objects are used for the system instead of unorganized raw data. In that way, extension can be easily done as these classes are reusable.