FIT3077 Assignment 3

Quarantine Coders Design Rationale

Extension from Assignment 2

Observer Pattern

- Thanks to the Observer design pattern previously implemented, we were able to easily extend our design to satisfy the new requirements of adding new monitor types to our applications. We created new subclasses from the Observer parent class for the new monitors including the **Blood Pressure Monitor**, **Historical Systolic Monitor** as well as **Graphical Monitors**. Each of these monitors observe the subject **PatientList** and provide their own implementation of the abstract **update**() method, which is called each time new data is requested from the server.

Open/Closed Principle (OCP)

- Applying the OCP allowed us to extend the functionalities of our app without modifying the behaviour of the existing system. An example can be found with our **BloodPressureDataClient** class – we were able to subclass this from the base class **FHIRClient** and have it provide a separate concrete implementation of the *get_patient_data()* method, without needing to modify the existing **CholesterolDataClient** class.
- Another similar example can be seen with the **BloodPressureData** class, which is extended directly from the abstract **PatientData** class which was in our design previously.

Refactoring

- **Rename Method**: several classes and functions to be more coherent with addition of new monitor type. For example, the **CholesterolMonitor** class was previously called **MonitorTreeview**, as there was only one type of data to be monitored then.
- Some functions needed to be modified to ensure the app works as intended with new monitor type. For example, the *highlight_patient()* method previously only had to work with the cholesterol monitor, but it needed to be changed to also highlight the blood pressure side of the GUI. The *format_data()* method was also modified to return formatted blood pressure data to be displayed in GUI.

Package-level Design Principles

The classes in our application have been separated into different modules (.py files) with the following principles in mind:

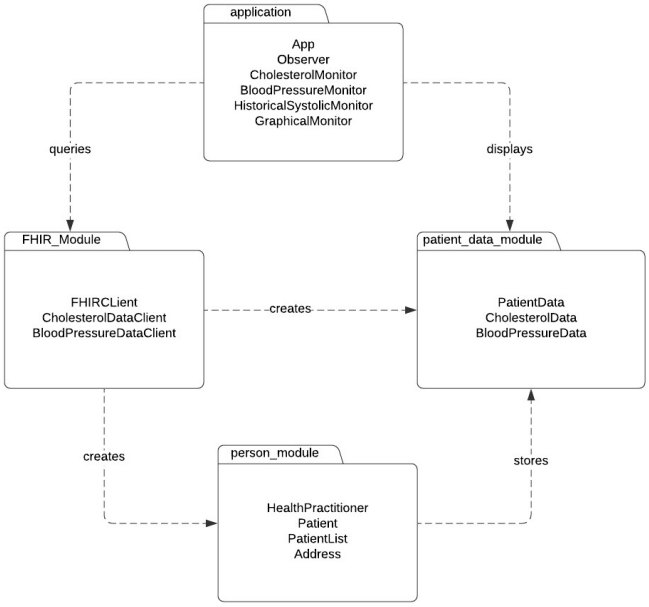
Package Cohesion

- Common Reuse Principle (CRP): A dependency on a module should be a dependency on everything in the module. Classes that are likely to be reused together, such as patient and patient list, are grouped into the same module.
- Common Closure Principle (CCP): Classes that change together belong together. In order to make the system easier to maintain, classes that are likely to change together are grouped into the same modules. An example is the classes **CholesterolData** and **BloodPressureData** – a change to one of these classes is likely to lead to a change in the other. Thus, they are

grouped into the same "**patientdata**" module. Similarly, **CholesterolDataClient** and **BloodPressureDataClient** are grouped onto the "**fhir_client**" module.

Package Coupling

- Acyclic Dependency Principle (ADP): Our design ensured that there is no cyclic dependency between our packages. This can be seen from the following package diagram:



Class Diagram

[Click here](#) to view the diagram on Lucidchart (Assignment 3 tab)