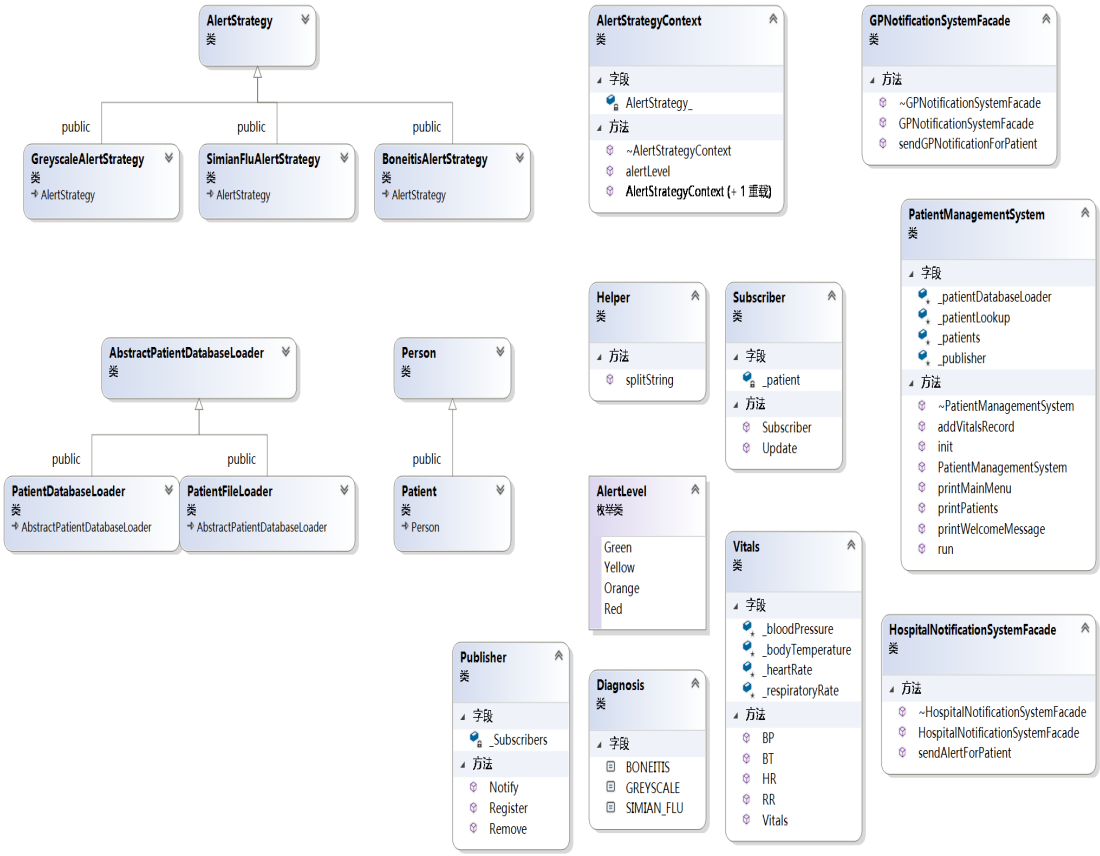
General Idea

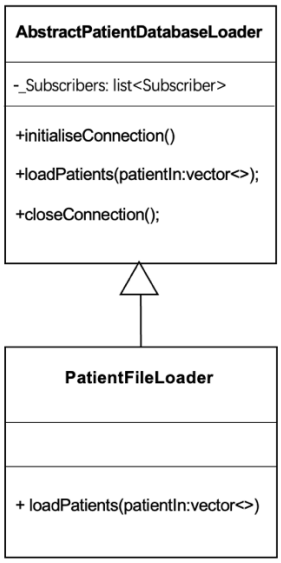
The main function of this system is to load data from databases and files, add vitals, and alert patients of their condition. I have also added an additional helper class to parse each line of the patients.txt file to extract the information we want.

UML



UML diagrams are automatically generated via visual studio 2019.

Functional requirement FR1: Load patients from file



Design Pattern: Composite Pattern

If the system uses PatientFileLoader directly, it is not easy to load files from the database and files at the same time at a later stage. To make it easier to use and manage, we make the PatientFileLoader inherit from the class AbstractPatientDatabaseLoader and override the loadPatients method so that it can load data from a file.

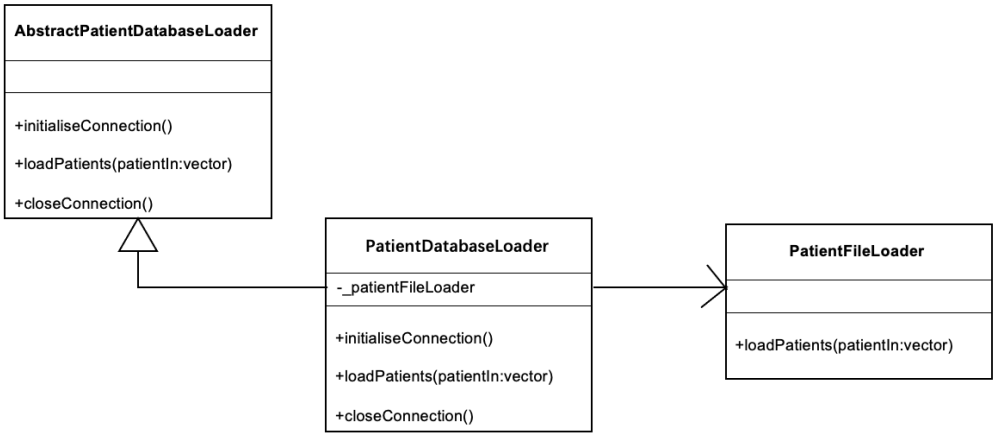
How it works:

1. The PatientManagementSystem will connect to the PatientDatabaseLoader by calling initialiseConnection ()
2. The PatientDatabaseLoader will call loadPatient(...) passing in a vector which will be loaded with patients.
3. The concrete PatientFileLoader will retrieve the patient details and construct patient objects which are added to the vector.
4. The PatientManagementSystem will release Patient memory allocated by the PatientFileLoader.
5. Finally, the PatientManagementSystem use closeConnection() to close the database connection.

Git commits:

- commit 90b33c.

Functional requirement FR2: load patients from file and database



Design Pattern: Adapter pattern

We want the system to be able to load data from databases and files, but to be as simple as possible if called from outside, so that external users only need to use one interface and can read data from different ways. With the adapter pattern we can achieve the functionality we need. With this pattern, the user can switch between database and file loading.

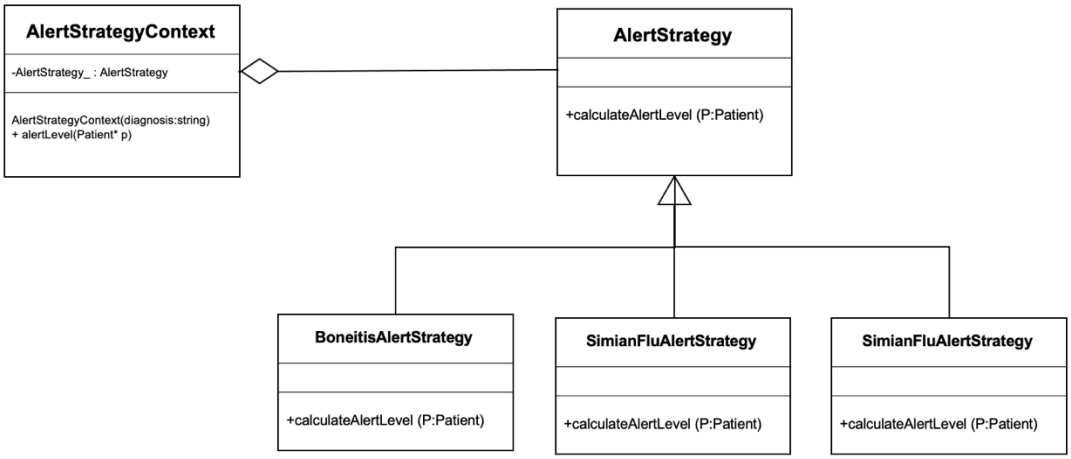
How it works:

From the diagram, we can see that the PatientDatabaseLoader inherits from the unified interface AbstractPatientDatabaseLoader, and adds the private property _patientFileLoader to the PatientDatabseLoader, finally we will call the PatientFileLoader's loadPatients method in the PatientDatabseLoader's loadPatients method

Git commits:

- in commit: Functional requirement FR2: load patients from file and database (0d0df18).

Functional requirement FR3: calculate the patient alert levels



Design Pattern: Strategy pattern

The alert level in the system is determined according to the different diagnosis. If we write the code intuitively, we have the problem of using if...else would be complicated and difficult to maintain. It is therefore preferable to use a policy model to differentiate the calculation of different diagnosis levels.

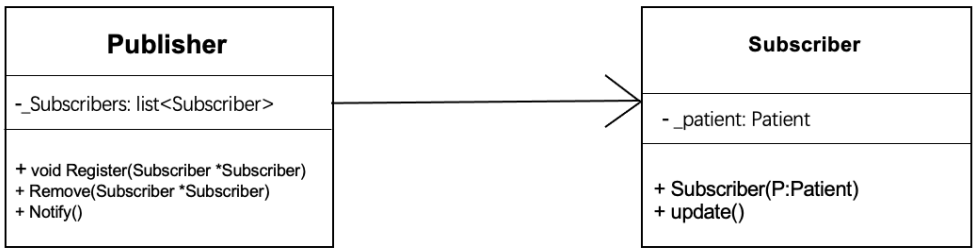
How it works:

The alert level is calculated by means of the patient's diagnosis, and different AlertStrategy classes can be created depending on the diagnosis, i.e. the different calculation methods are encapsulated in a single class, which all follow a uniform precept.

Git commits:

- in commit: 461b4ba09944646e5d3c7a90ff800722254bf0da.

Functional requirement FR4: alert the hospitals and GPs



Design Pattern: Observer Pattern

The Observer pattern defines a one-to-many dependency between objects, so that when the state of an object changes, all its dependents are notified and automatically updated with the relevant content. This means that a (Publisher class) to many (Subscriber class) relationships is established, enabling multiple instances of objects that depend on the Publisher to be synchronized with the corresponding changes when the Publisher's object changes.

How it works:

We encapsulate all the patients in the database into a subscriber, then put all the subscribers into the Publisher's list, and each time notify() is called, it will loop through the subscriber members, i.e., call the corresponding update method of the subscriber.

Git commits:

- in commit: db396afe5dcaa390344a5c312c4c3784025785ca