Assignment 2

Patient Vitals Management System

Habib, Malachi William - habmw001

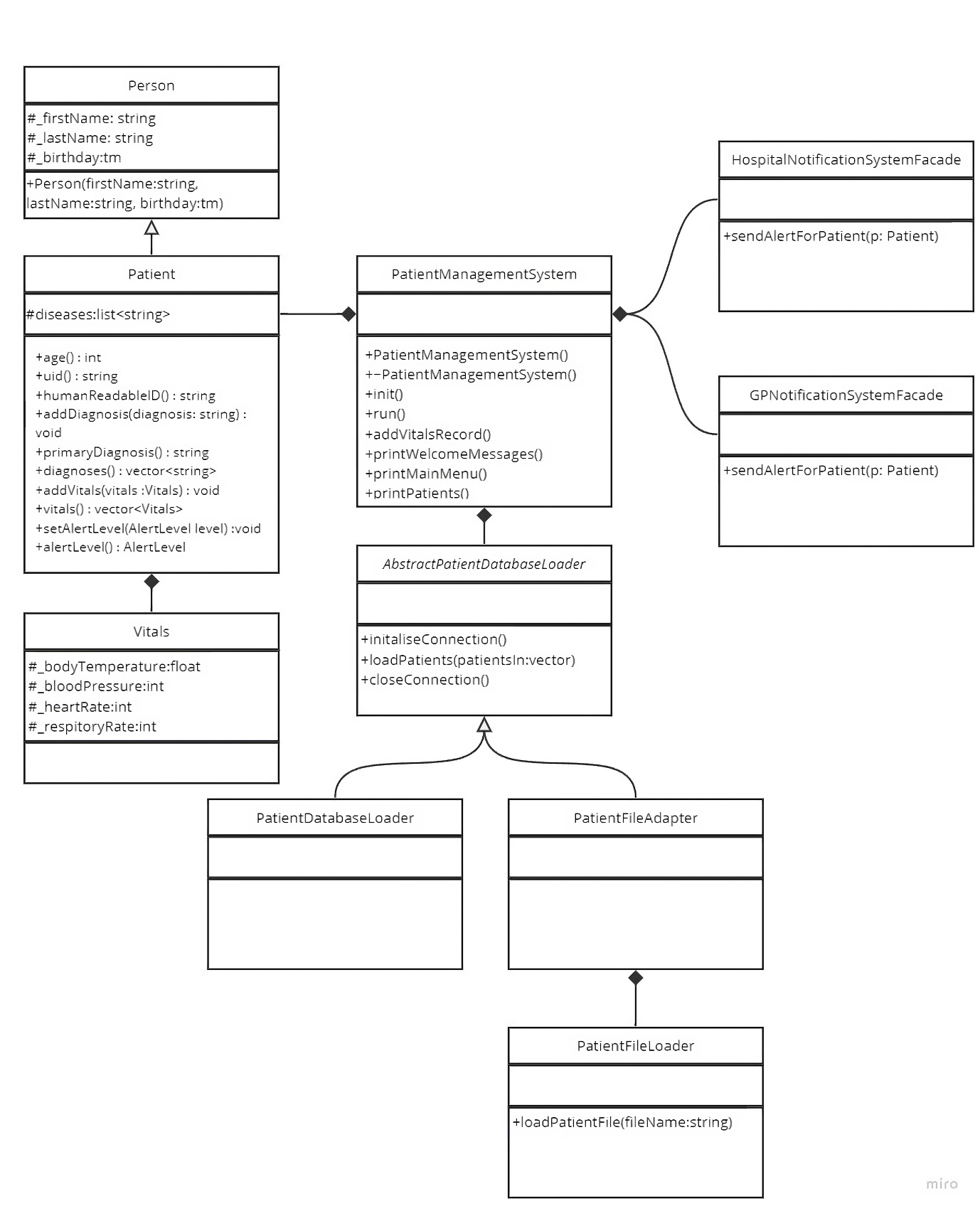
2023

## V1 – FR1 – Adapter Pattern

The pattern chosen was the Adapter pattern.

### Justification

We need to make the PatientFileLoader compatible with the existing AbstractPatientDatabaseLoader abstract base class. This means we must adapt the functionality of the PatientFileLoader to fit the method signatures declared in the AbstractPatientDatabaseLoader abstract base class. Essentially, the PatientFileLoader needs a 'wrapper' that allows the uses of its methods within the expected interface of the PatientDatabaseLoader. This scenario makes a great use case for the Adapter pattern.



### How It Works

1. The PatientFileAdapter class is a subclass of the AbstractPatientDatabaseLoader abstract base class, within the adapter class, it has an instance of the PatientFileLoader.
2. As the PatientFileAdapter is a subclass of the AbstractPatientDatabaseLoader abstract base class, it must implement all the methods within the parent class, this allows us to implement custom logic into the methods. Therefore, we will use the local instance of the PatientFileLoader to implement our chosen functionality within the loadPatients method of the AbstractPatientDatabaseLoader abstract base class.
3. We can easily switch between using the PatientFileAdapter and the PatientDatabaseLoader as they both inherit from the same abstract base class, thereby enforcing a consistent interface.

### Git Commits

Implementation of the adapter pattern: fdb056d.

## V2 – FR2 – Strategy Pattern

The pattern chosen was the Strategy pattern.

### Justification

A picture containing text, diagram, plan, technical drawing

Description automatically generatedThe Strategy pattern was chosen due to its ability to interchange algorithms at runtime. Each Patient has a different primary disease, which each require a different algorithm to calculate the relevant alert level. By using the Strategy pattern, we can encapsulate each algorithm in their own Strategy class, and switch between them dynamically based on the Patients current primary disease.

### How It Works

The AlertLevelStrategy abstract base class is the common interface for all strategy classes, it defines one method, calculateAlertLevel, the method will be implemented by all the concrete strategy classes and will be used to apply the varying logic between the classes.

Each disease has its own strategy class, as mentioned above, each class will implement its own calculateAlertLevel method. Each implementation of the method will apply the relevant logic to the disease.

When the primary disease of a Patient is identified, the local variable AlertLevelStrategy is set to the specific instance of the Strategy class for that disease. For instance, if the Patients primary disease was MadZombieDisease, then the AlertLevelStrategy variable would be set to MadZombieDiseaseStrategy, allowing the correct logic to be applied when adding vitals.

Whenever **new** vitals are added to a Patient, the system will call the addVitals method, in which there is an internal call to the calculateAlertLevel method, this will apply the above logic mentioned, and set the AlertLevelStrategy variable to an instance that correlates to the Patients primary disease.

### Git Commits

Implementation of the Strategy pattern: 9d6621a.

## V3 – FR3 – Composite Pattern

### Justification

### How It Works

### Git Commits

## V4 – FR4 – Observer Pattern

### Justification

### How It Works

### Git Commits