Assignment 2

Patient Vitals Management System

Habib, Malachi William - habmw001

2023

Contents

[V1 – FR1 – Adapter Pattern 2](#_Toc137070136)

[Justification 2](#_Toc137070137)

[How It Works 3](#_Toc137070138)

[Git Commits 3](#_Toc137070139)

[V2 – FR2 – Strategy Pattern 3](#_Toc137070140)

[Justification 3](#_Toc137070141)

[How It Works 4](#_Toc137070142)

[Git Commits 4](#_Toc137070143)

[V3 – FR3 – Composite Pattern 5](#_Toc137070144)

[Justification 5](#_Toc137070145)

[How It Works 5](#_Toc137070146)

[Git Commits 5](#_Toc137070147)

[5](#_Toc137070148)

[V4 – FR4 – Observer Pattern 6](#_Toc137070149)

[Justification 6](#_Toc137070150)

[How It Works 6](#_Toc137070151)

[Git Commits 6](#_Toc137070152)

[Completed Diagram 0](#_Toc137070153)

## V1 – FR1 – Adapter Pattern

The pattern chosen was the Adapter pattern.

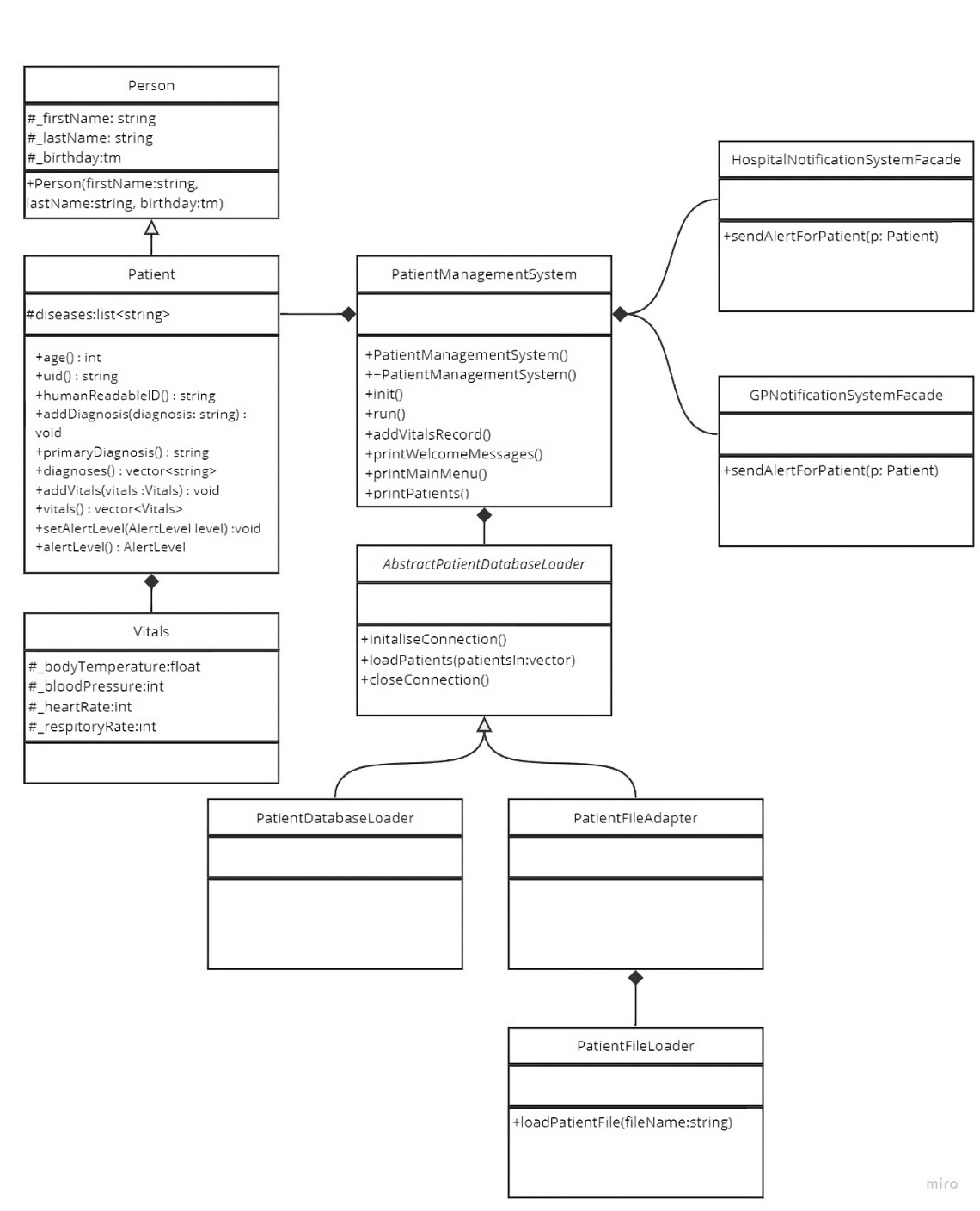
### Justification

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

The 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.

A picture containing text, diagram, plan, technical drawing

Description automatically generatedMerge of the FR2 branch into main: 9dd976b.

Added doc with FR2 details: 2bb4864.

## V3 – FR3 – Composite Pattern

### Justification

The Composite pattern allows for treating individual objects and compositions of objects uniformly. What this means in the example of the management system is that we can manage all the various strategy classes from the one composite strategy class. This greatly simplifies the interactions between the patient and the strategies, allowing the composite strategy class to handle the delegation per disease internally.

### How It Works

1. The CompositeAlertLevelStrategy class is a subclass of the AlertLevelStrategy abstract base class, it is responsible for the management of instances of the AlertLevelStrategy abstract base class.
2. When adding new Vitals to a Patient, the system calls the CompositeAlertLevelStrategy determineAlertLevel method. This method will iterate through all the relevant strategies relevant and call their version of the determineAlertLevel implementation.
3. Each of these calculateAlertLevel method calls will return an AlertLevel. The CompositeAlertLevelStrategy will determine the highest AlertLevel returned and assign it to the patient’s current AlertLevel. As mentioned, this simplifies the logic, and encapsulates it, overall, it greatly reduces the complexity of interactions between the Patient and the Strategy classes.

### Git Commits

Updated the AlertStrategy to include a CompositeAlertLevelStrategy class: db56b86.

Merge of the FR3 branch into main: 10b46d9.

### 

### A picture containing text, diagram, plan, technical drawing Description automatically generated

## V4 – FR4 – Observer Pattern

### Justification

The Observer pattern is a good fit here because we need to inform the Hospital and the GP when a Patient’s Alert Level becomes critical. The Observer pattern allows us to be alerted about other classes when they change, without having to have them tightly linked or dependant on each other. Not only is this a good software design principle, but it also allows us to simplify the logic and make our code easier to read and understand.

### How It Works

1. The Patient class inherits from the Subject class, this is common in the Observer pattern. The Subject class represents the class(s) that is being observed, this means that when an instance of the Subject abstract base class changes, the Observer classes will be notified (if it is setup correctly). Each Subject class contains a list of observers, in our case, the abstract base class for the Observer pattern is the HospitalAlertObserver.
2. When a Patient’s inputted vitals indicate an alert level of red, the Patient class will notify all observers about the critical condition.
3. From there, each observer will perform a given action, for our scenario we have two observers. The GPNotificationSystemFacade and the HospitalAlertSystemFacade.
4. The design allows for expansion, there is the inclusion of an unused method to remove observers, as this would be required within a real system. The design of this allows for future code changes with minimal impact. Primarily this heaps to obfuscate the code, as the Patient class does not need to know what each notification does, therefore it makes a lot of sense to remove this logic from the Patient class, as we have done.

### Git Commits

Basic implementation of the Observer pattern: d1ca855.

Changed variable location for the observer’s vector: 477e18e.

Merge of the FR4 branch into main: 00a76c6.

(See diagram on the next page)

## Completed Diagram

