## FR1: load patients from file

**Design Pattern:** Adapter Pattern

**Why I am using this pattern:** The System is required to take data from the text file. The PatientFileLoader class is already given to me to read the patient data from a local file (patients.txt in this case). However, it is not compatible with the interface expected from the class (AbstractPatientDatabaseLoader), since it was designed for the database. Adapter pattern is used to solve this problem; it allows incompatible interfaces to work together by creating a class which convert one interfaces into another. In this case, the adapter is (FilePatientDatabaseLoader), it adapts the PatientFileLoader to conform to AbstractPatientDatabaseLoader interface.

A diagram of a patient database

AI-generated content may be incorrect.

**How it works:**

1. Creates an adapter that implements AbstractPatientDatabaseLoader

2. Uses the existing PatientFileLoader internally

3. Converts between the database-style interface and file loading

4. Allows easy switching between database and file loading with one line change

5. Maintains the existing PatientFileLoader implementation

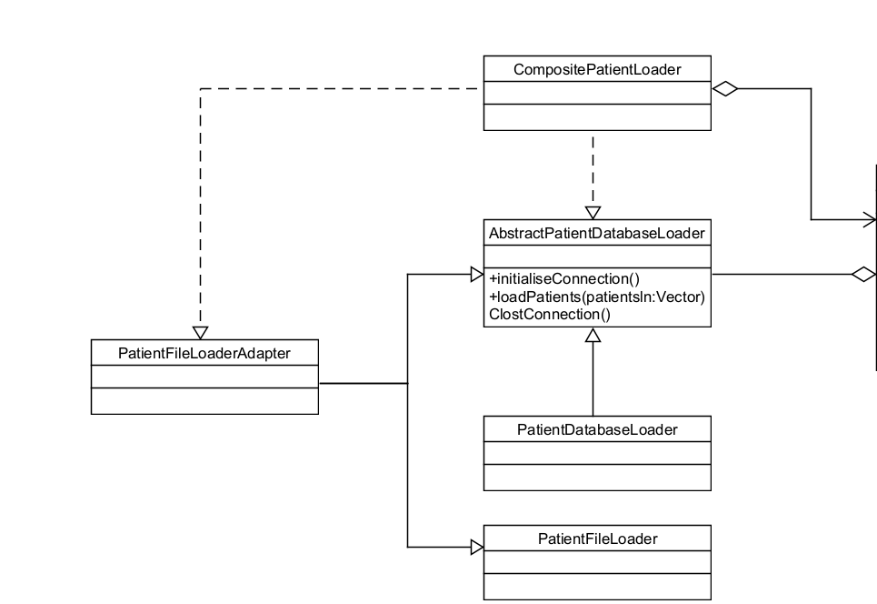
**Git Commits:**

* **I first implemented the PatientFileLoader in commit f52438.**
* **I created the adapter PatientFileLoaderAdapter in commit 42df08.**
* **The adapter was implemented in 4f473**
* **Other related commits: /**

## FR2: load patients from file and database

**Design Pattern:** Composite Pattern

**Why I am using this pattern:** The Composite pattern is used to implement this function requirement. The system is required to load patients from three ways: from database, from text file and from both. And is required to switch among these configurations easily with a line of code change. Composite is ideal for dealing this case because it can help treat different loaders (PatientDatabaseLoader and PatientFileLoader) and the combination (CompositePatientLoader) evenly through a common interface which is AbstractPatientDatabaseLoader. This enables us to create a tree-like structure where the composite can contain individual loaders or other composites, executing them in a specific order while hiding the complexity of managing multiple data sources.

****

**How it works:**

1.Creates a CompositePatientLoader that implements AbstractPatientDatabaseLoader

2. Allows multiple loaders to be added to a single composite loader

3. Maintains order of execution (database first, then file) by order of addition

4. Delegates all operations (initialize, load, close) to contained loaders in sequence

5. Uses existing PatientDatabaseLoader and PatientFileLoaderAdapter implementations

6. Allows switching between configurations by adding/removing loaders:

- Database only: Add just PatientDatabaseLoader

- File only: Add just PatientFileLoaderAdapter

- Both (default): Add both loaders in order

7. Provides single-line configuration changes through the addLoader method

8. Maintains the existing database and file loader implementations unchanged

**Git Commits:**

* **I created the composite PatientFileLoader in commit 53fb4e.**
* **The adapter was implemented in 486abb.**
* **Other related commits: /**

## FR3 calculate the patient alert levels

**Design Pattern:** Strategy Pattern

**Why I am using this pattern:** The system must calculate a patient’s Alert Level using disease-specific algorithms that vary depending on the patient's diagnosed diseases. Each patient’s Alert Level is recalculated whenever new vital signs data are input, but **not** during initial data loading from historical sources such as databases or files. I choose strategy pattern for this requirement because it allows the system to treat each disease-specific alert calculation algorithm independently. This approach enables flexible, extensible alert level calculations without the need to modify patients’ code. The system can also dynamically choose the appropriate algorithm at runtime based on the patient’s disease.

A diagram of a company

AI-generated content may be incorrect.

**How it works:**

1. Creates an interface AlertLevelCalculator for calculating alert levels for different diseases.
2. Create the three calculator classes, each of them is corresponding to one disease.
3. The new calculator classes will inherit from the abstract class AlertLevelCalculator.
4. Implement each calculator based on the methods of determining alert level.

e.g. Patient(s) diagnosis with Andromeda Strain will have alert level of red if blood pressure is over 140.

1. Assign calculators based on the patient’s diagnosis which allows switching between configurations automatically based on the patient’s diagnosis
2. When new vitals for patient are added, the calculator will automatically detect the diseases with diagnosis and choose the correct calculator.
3. More diseases can be monitored by adding the corresponding calculator.

**Git Commits:**

* The base interface AlertLevelCalculator is created in commit d61ff1.
* The specific calculators are created and implemented as follows:

Cordyceps Brain Infection created & implemented in 47431f.

Kepral’s Syndrome Created in 72a2ba, implemented in e9f8a8

Andromeda Strain Created and implemented in 72a2ba

* Other related commits: /

## FR4: alert the hospitals and GPs

Design Pattern: Observer Pattern