## FR1: load patients from file

Design Pattern: Adapter 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 eeddbd.**
* **I created the adapter PatientFileLoaderAdapter in commit 611cbd**
* **The adapter was implemented in 42df08**
* **Other related commits:**

## FR2: load patients from file and database

Design Pattern: Composite 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 from the client code.

3. Strategy Pattern

4. Observer Pattern