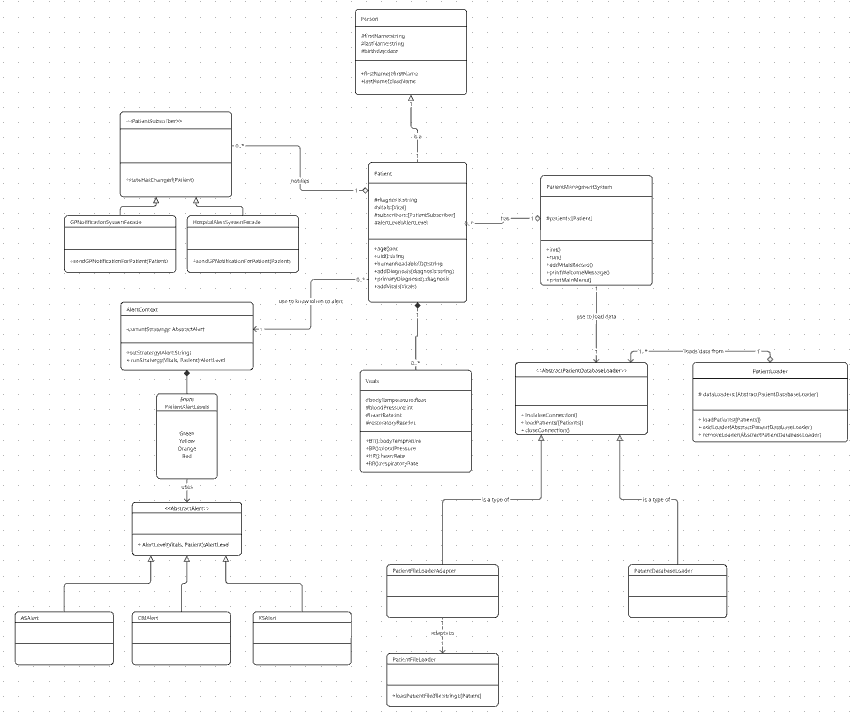
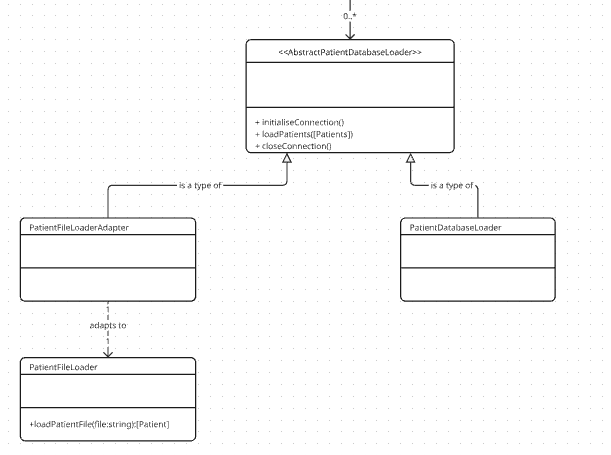
Patient Management System

# Complete class diagram



# FR1: Load data from file

* 1. **Name** the design pattern you chose.
  2. Adapter pattern
  3. Explain **why** you applied that particular design pattern (one paragraph).
  4. The adapter pattern allows the easy access of patient data in lots of different formats by building an adapter for loading data from a file I make it easy to add data from different sources down the line. E.g adding from a database instead of a file.
  5. **Show** a class diagram that highlights the specific classes in the system and relationships that address that design pattern.



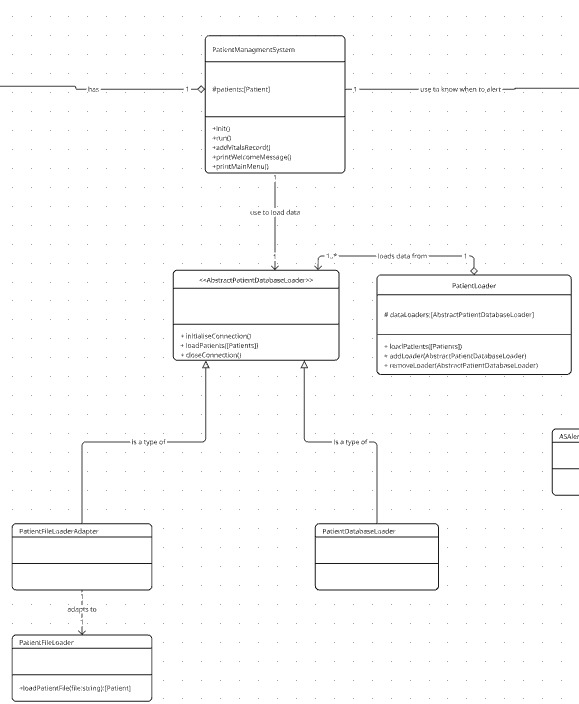
* 1. Explain, step by step, **how** your design works. A numbered list may work best here. Reference your diagrams as necessary.
  2. 1. The PatientLoader (shown in the complete class diagram) calls database loaders using loadPatients
  3. 2. The PatientDatabaseLoader inherits directly from AbstractPatientDatabaseLoader so it just implements the loadPatients method directly
  4. 3. The PatientFileLoaderAdapter class implements the loadPatients method and in this implementation is calls the loadPatientFile method on the PatientFileLoader class
  5. Reference the git commits by commit ID where you addressed this aspect of the design.

I started work on file adding in: 9fde2a2

Finished file loading in: 3724a8f

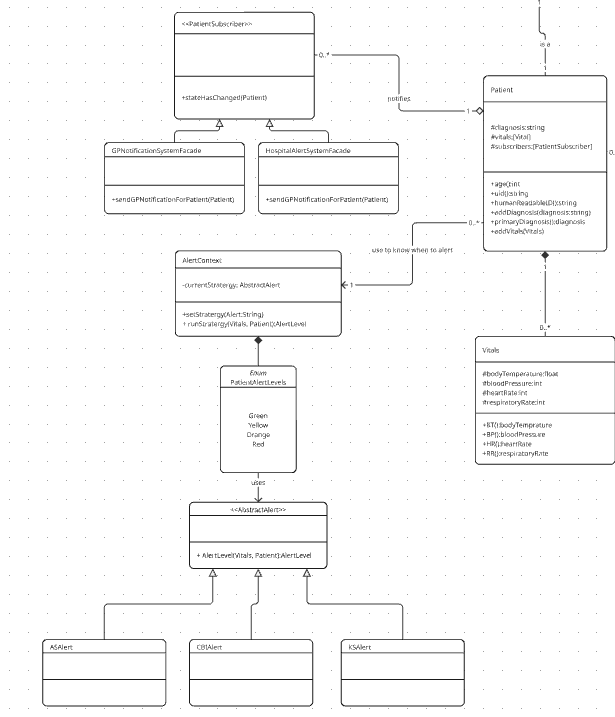
I fully implemented the adapter pattern in:251deebc

# FR2: Load data from database

* 1. **Name** the design pattern you chose.
  2. Composite pattern
  3. Explain **why** you applied that particular design pattern (one paragraph).
  4. The composite pattern helps me to treat different objects as the same which works well in this case because I am loading data from files and databases but I don’t care how the data is loaded I am just looking for the data extracted. By applying the composite pattern I can treat patient data from file and patient data from a database as just patient data.
  5. **Show** a class diagram that highlights the specific classes in the system and relationships that address that design pattern.
  6. 
  7. Explain, step by step, **how** your design works. A numbered list may work best here. Reference your diagrams as necessary.
  8. 1. The PatientManagmentSystem creates PatientDatabaseLoader and PatientFileLoaderAdapter
  9. 2. The PatientManagmentSystem adds these AbstractPatientDatabaseLoader classes to PatientLoader
  10. 3. The PatientManagmentSystem calls loadPatients on PatientLoader
  11. 4. PatientLoader loops through all of it’s data loaders and calls loadPatients on them
  12. 5. Each data loader updates the patients list with their data
  13. Reference the git commits by commit ID where you addressed this aspect of the design.

I worked off file loader commits to start and finish implementation of PatientLoader in:0a9805a

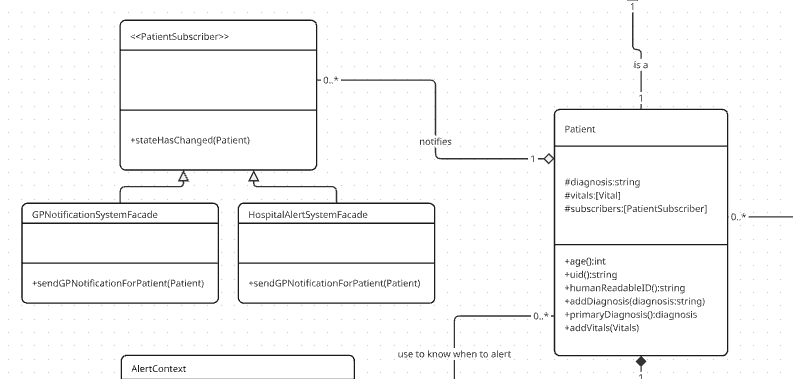
# FR3: Generate patient alerts

* 1. **Name** the design pattern you chose.
  2. Strategy pattern
  3. Explain **why** you applied that particular design pattern (one paragraph).
  4. I have to apply a strategy to generate alerts for different patients based on a long list of criteria. The strategy pattern allows me to abstract away the implementation for different diseases making it easy to add multiple diseases, and be able to define different alert criteria for each disease.
  5. **Show** a class diagram that highlights the specific classes in the system and relationships that address that design pattern.
  6. 
  7. Explain, step by step, **how** your design works. A numbered list may work best here. Reference your diagrams as necessary.
  8. 1. The Patient sets which AbstractAlert to use based upon the diagnosis of the patient by calling setStratergy on AlertContext
  9. 2. The Patient calls runStratergy using the newly entered vitals from the patient
  10. 3. The AlertContext calls the AlertLevel method on the currentStratergy
  11. 4. AbstractAlert returns a AlertLevel enum to AlertContext
  12. 5. AlertContext returns the alert level to the patient
  13. Reference the git commits by commit ID where you addressed this aspect of the design.

I started in: 2569c92

I finished implementation in:1214bf7

# FR4: Alert doctors

* 1. **Name** the design pattern you chose.
  2. Observer pattern
  3. Explain **why** you applied that particular design pattern (one paragraph).
  4. The observer pattern works well for notification systems because the patient doesn’t care what it’s vitals are but their doctor/s might. But we don’t know which doctors care, the observer pattern allows us to setup the system so that the doctors assign which patients vitals they care about and subscribe to their vital alerts.
  5. **Show** a class diagram that highlights the specific classes in the system and relationships that address that design pattern.
  6. 
  7. Explain, step by step, **how** your design works. A numbered list may work best here. Reference your diagrams as necessary.
  8. 1. The PatientManagmentSystem adds the GPNotificationSystemFacade and HospitalAlertSystemFacade to each newly created patient as subscribers
  9. 2. When the patient updates its alertLevel it calls stateHasChanged on each of it’s subscribers
  10. 3. Each type of PatientSubscriber checks the alert level and if it meets the threshold it sends a notification
  11. Reference the git commits by commit ID where you addressed this aspect of the design.

I fully implemented all aspects in: 41aa87e