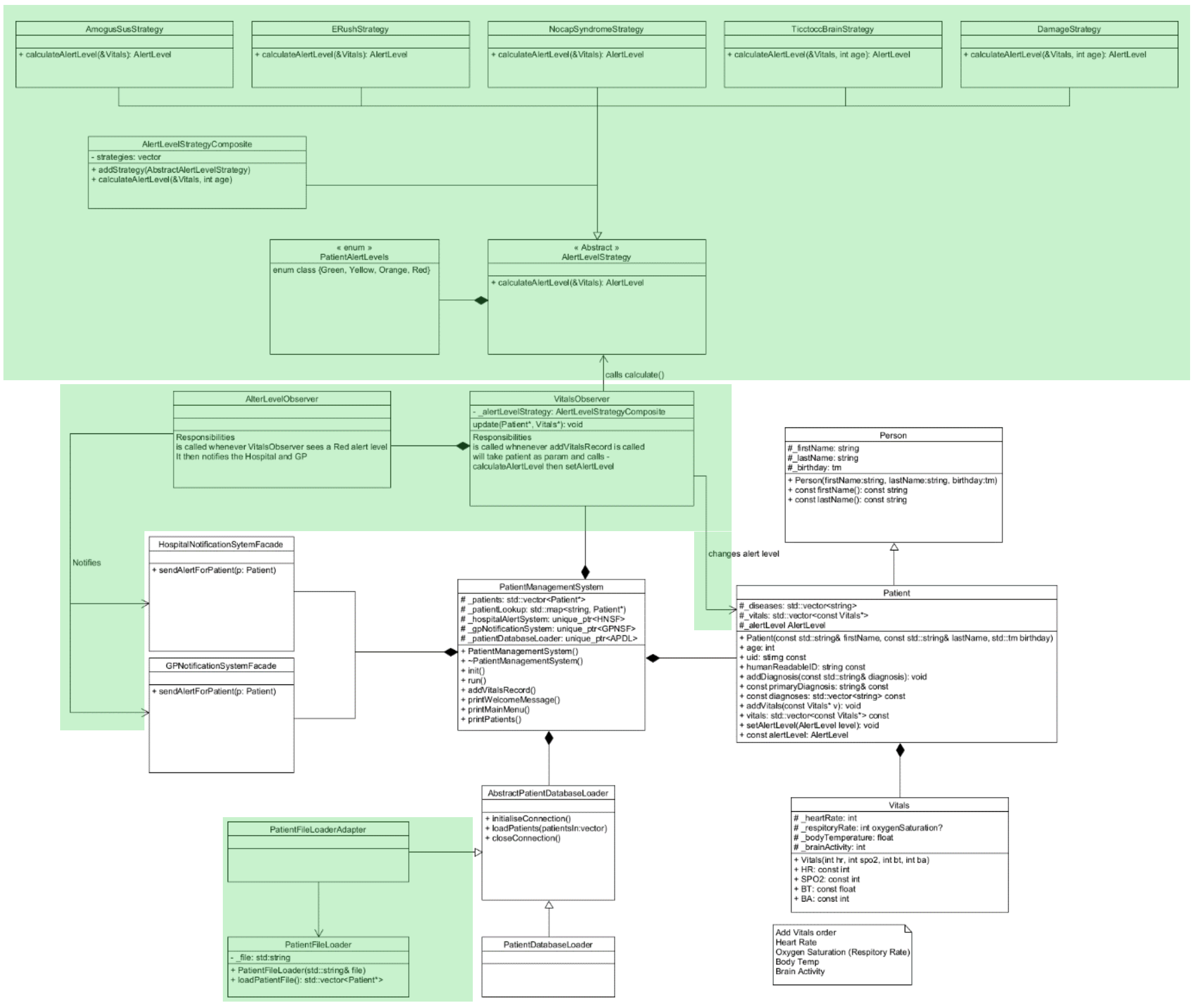
Miguel Angelo Bautista C++ A2 Design Document

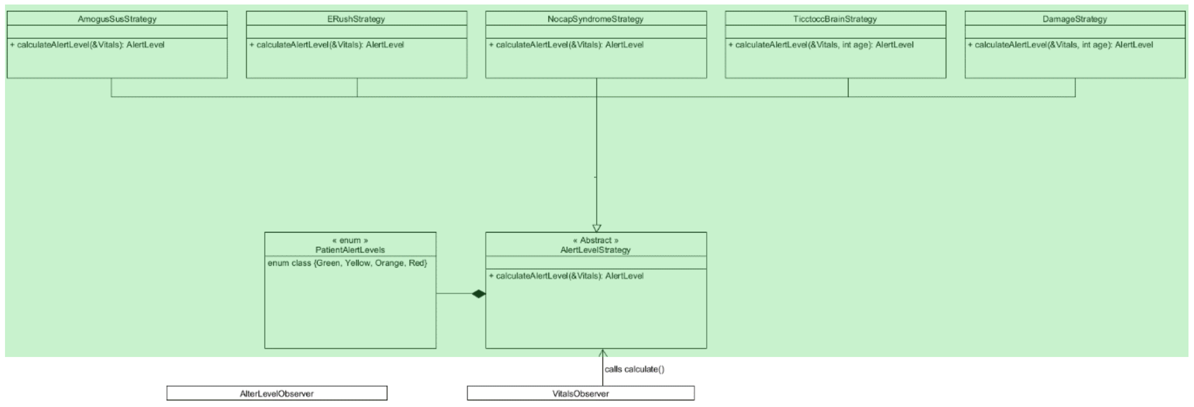
Baumy007

Design Diagram

Note: Those in green are the additional parts to the system


AmogusSusStrategy
+ calculateAlertLevel(&Vitals): AlertLevel
ERushStrategy
+ calculateAlertLevel(&Vitals): AlertLevel
NocapSyndromeStrategy
+ calculateAlertLevel(&Vitals): AlertLevel
TicctoccBrainStrategy
+ calculateAlertLevel(&Vitals, int age): AlertLevel
DamageStrategy
+ calculateAlertLevel(&Vitals, int age): AlertLevel
AlertLevelStrategyComposite
- strategies: vector
+ addStrategy(AbstractAlertLevelStrategy)
+ calculateAlertLevel(&Vitals, int age)
« enum »
PatientAlertLevels
enum class {Green, Yellow, Orange, Red}
« Abstract »
AlertLevelStrategy
+ calculateAlertLevel(&Vitals): AlertLevel
calls calculate()
AlterLevelObserver
Responsibilities
is called whenever VitalsObserver sees a Red alert level
It then notifies the Hospital and GP
VitalsObserver
- _alertLevelStrategy: AlertLevelStrategyComposite
update(Patient*, Vitals*): void
Responsibilities
is called whnenever addVitalsRecord is called
will take patient as param and calls -
calculateAlertLevel then setAlertLevel
Person
#_firstName: string
#_lastName: string
#_birthday: tm
+ Person(firstName:string, lastName:string, birthday:tm)
+ const firstName(): const string
+ const lastName(): const string
Notifies
HospitalNotificationSytemFacade
+ sendAlertForPatient(p: Patient)
GPNotificationSystemFacade
+ sendAlertForPatient(p: Patient)
changes alert level
PatientManagementSystem
# _patients: std::vector<Patient*>
# _patientLookup: std::map<string, Patient*)
# _hospitalAlertSystem: unique_ptr<HNSF>
# _gpNotificationSystem: unique_ptr<GPNSF>
# _patientDatabaseLoader: unique_ptr<APDL>
+ PatientManagementSystem()
+ ~PatientManagementSystem()
+ init()
+ run()
+ addVitalsRecord()
+ printWelcomeMessage()
+ printMainMenu()
+ printPatients()
Patient
#_diseases: std::vector<string>
#_vitals: std::vector<const Vitals*>
#_alertLevel AlertLevel
+ Patient(const std::string& firstName, const std::string& lastName, std::tm birthday)
+ age: int
+ uid: stirng const
+ humanReadableID: string const
+ addDiagnosis(const std::string& diagnosis): void
+ const primaryDiagnosis: string& const
+ const diagnoses: std::vector<string> const
+ addVitals(const Vitals* v): void
+ vitals: std::vector<const Vitals*> const
+ setAlertLevel(AlertLevel level): void
+ const alertLevel: AlertLevel
AbstractPatientDatabaseLoader
+ initialiseConnection()
+ loadPatients(patientsIn:vector)
+ closeConnection()
PatientFileLoaderAdapter
PatientFileLoader
- _file: std:string
+ PatientFileLoader(std::string& file)
+ loadPatientFile(): std::vector<Patient*>
PatientDatabaseLoader
Vitals
# _heartRate: int
# _respitoryRate: int oxygenSaturation?
# _bodyTemperature: float
# _brainActivity: int
+ Vitals(int hr, int spo2, int bt, int ba)
+ HR: const int
+ SPO2: const int
+ BT: const float
+ BA: const int
Add Vitals order
Heart Rate
Oxygen Saturation (Respitory Rate)
Body Temp
Brain Activity

FR1: Calculate the patient alert levels


AmogusSusStrategy
+ calculateAlertLevel(&Vitals): AlertLevel
ERushStrategy
+ calculateAlertLevel(&Vitals): AlertLevel
NocapSyndromeStrategy
+ calculateAlertLevel(&Vitals): AlertLevel
TicctoccBrainStrategy
+ calculateAlertLevel(&Vitals, int age): AlertLevel
DamageStrategy
+ calculateAlertLevel(&Vitals, int age): AlertLevel
« enum »
PatientAlertLevels
enum class {Green, Yellow, Orange, Red}
« Abstract »
AlertLevelStrategy
+ calculateAlertLevel(&Vitals): AlertLevel
calls calculate()
AlterLevelObserver
VitalsObserver

Design Pattern: Strategy

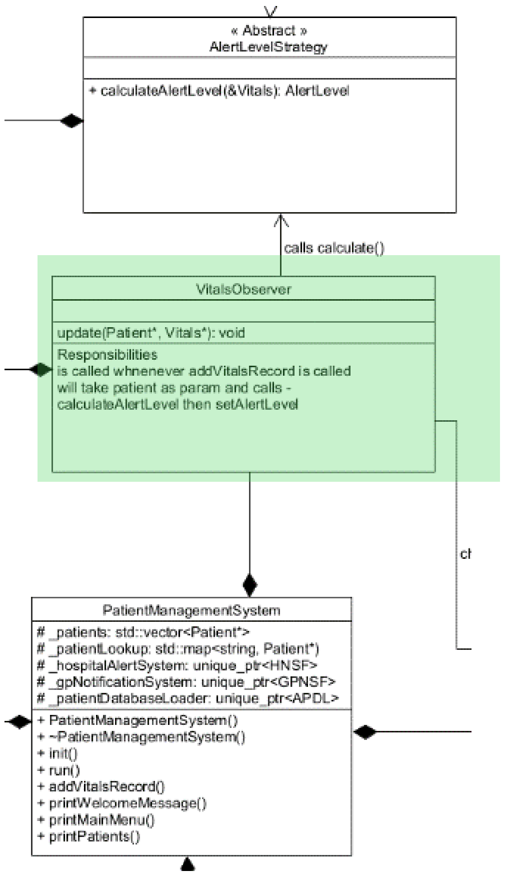
The system must calculate the alert level of the patient for their primary disease. To help implement this, I used a strategy pattern where the class AlertLevelStrategy acts as the Abstract strategy class. This creates a common interface to be used by all diseases for calculating Alert Levels, allowing easy extensibility for future diseases, as well as better programming experience by encapsulating each disease class into one common parent class. The AlertLevelStrategy is called by the VitalsObserver so that whenever new vitals have been recorded, The strategy is called.

How it works:

1. VitalsObserver initialises and calls a specific AlertLevelStrategy based on the patient's primary disease
2. The specific strategy is then run and will calculate what alert level to give the patient based on the disease
3. Finally, the strategy returns the appropriate alert level back to VitalsObserver

Git commits:

- I first added the strategies in commit 2829ec.
- I then refactored the strategies to have two implementations of calculateAlertLevel as two diseases require the patient's age in commit 6da76d, and 993653


« Abstract »
AlertLevelStrategy
+ calculateAlertLevel(&Vitals): AlertLevel
calls calculate()
VitalsObserver
update(Patient*, Vitals*): void
Responsibilities
is called whnenever addVitalsRecord is called
will take patient as param and calls -
calculateAlertLevel then setAlertLevel
cl
PatientManagementSystem
# _patients: std::vector<Patient*>
# _patientLookup: std::map<string, Patient*)
# _hospitalAlertSystem: unique_ptr<HNSF>
# _gpNotificationSystem: unique_ptr<GPNSF>
# _patientDatabaseLoader: unique_ptr<APDL>
+ PatientManagementSystem()
+ ~PatientManagementSystem()
+ init()
+ run()
+ addVitalsRecord()
+ printWelcomeMessage()
+ printMainMenu()
+ printPatients()

Design Pattern: Observer

In order to know when to calculate the patient's alert level, A new, non-historical vitals data needs to be created. In order to determine when new vitals data has been recorded, The VitalsObserver is notified by the PatientManagementSystem whenever the user selects 'record new vitals'. This releases PatientManagementSystem from further unrelated functionality, creating a separation of concerns.

How it works:

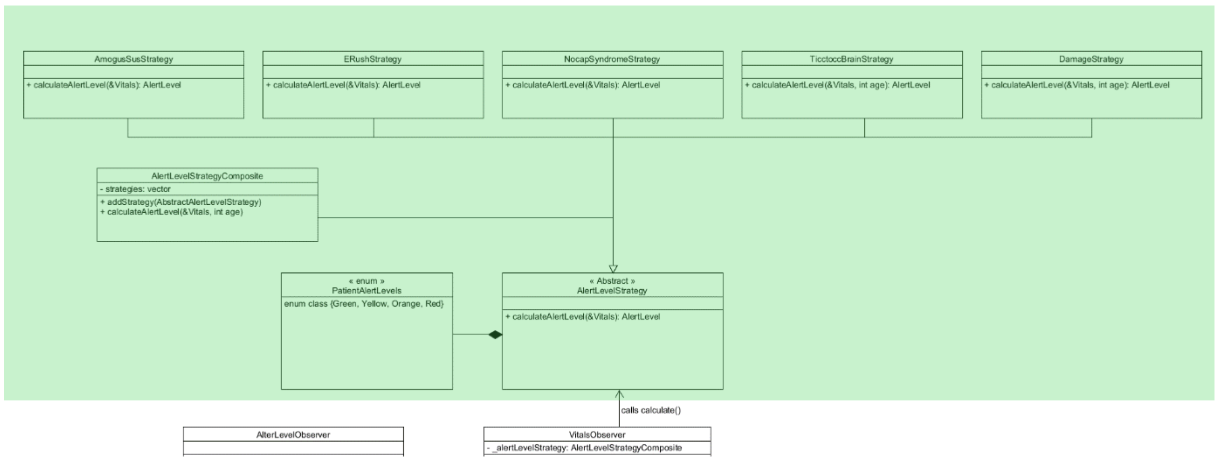
1. User selects the "Record new Vitals" option and creates new vitals for a certain patient
2. PatientManagementSystem then notifies VitalsObserver that a new set of vitals has been created for the patient and will run the patient's primary disease's strategy to calculate the alert level
3. The VitalsObserver then receives the new alert level of the patient from the strategy, then sets that new Alert Level for the patient

The job of the VitalsObserver is to be notified when the 'record new vitals' command in the UI is called by the user, which then subsequently calls the calculateAlertLevel on the strategy pattern that was discussed earlier.

Git Commits:

- The VitalsObserver was implemented in commit 88b15a
- Then in commit 993653, I refactored VitalsObserver to allow the new change for the strategies in commit 6da76d, and 993653, as well as removed some debugging code

FR2: Calculate the alert level for all diseases a patient has


AmogusSusStrategy
+ calculateAlertLevel(&Vitals): AlertLevel
ERushStrategy
+ calculateAlertLevel(&Vitals): AlertLevel
NocapSyndromeStrategy
+ calculateAlertLevel(&Vitals): AlertLevel
TicctoccBrainStrategy
+ calculateAlertLevel(&Vitals, int age): AlertLevel
DamageStrategy
+ calculateAlertLevel(&Vitals, int age): AlertLevel
AlertLevelStrategyComposite
- strategies: vector
+ addStrategy(AbstractAlertLevelStrategy)
+ calculateAlertLevel(&Vitals, int age)
« enum »
PatientAlertLevels
enum class {Green, Yellow, Orange, Red}
« Abstract »
AlertLevelStrategy
+ calculateAlertLevel(&Vitals): AlertLevel
calls calculate()
AlterLevelObserver
VitalsObserver
- _alertLevelStrategy: AlertLevelStrategyComposite

Design Pattern: Strategy + Composite Pattern

Because the calculation is now being done for each disease a patient has instead of just the primary disease, I implemented a Composite Pattern with the name AlertLevelStrategyComposite to help ease the calculation of each strategy. Each strategy is still working the same way as before, just that it is now being done through the composite class.

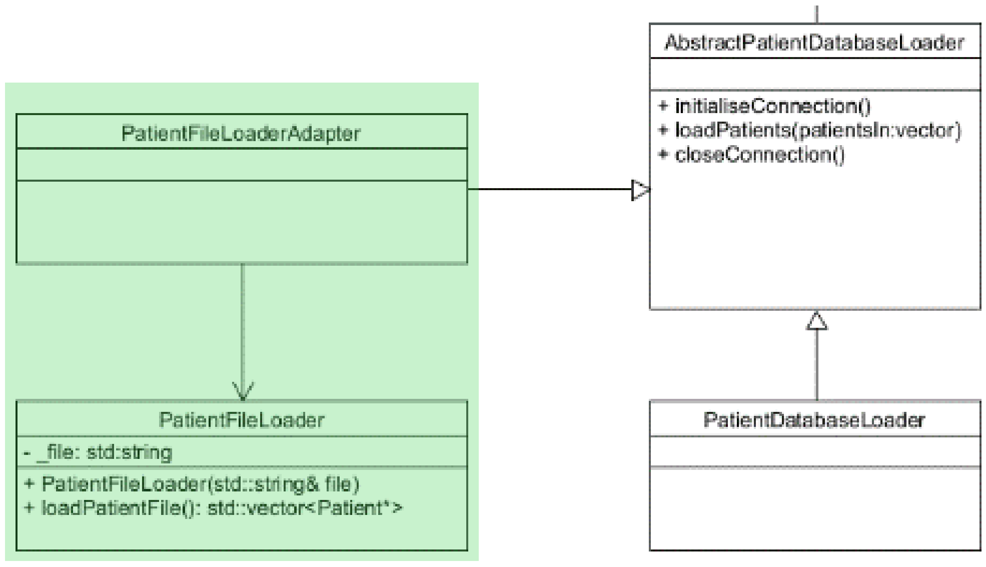
How it works:

1. VitalsObserver initialises the AlertLevelStrategyComposite, then It calls the addStrategy of the class and adds each disease that the patient has.
2. Then, the VitalsObserver calls the composite class' calculateAlertLevel which goes through each of the added strategies' calculateAlertLevel functions.
3. It checks which AlertLevel is the highest and returns that AlertLevel back to the Vitals Observer

Git commits:

- AlertLevelStrategyComposite was first introduced in commit 6da76d
- It was then refactored in commit 993653 to implement the two implementations of calculateAlertLevels

FR3: Load patients from file


AbstractPatientDatabaseLoader
+ initialiseConnection()
+ loadPatients(patientsIn:vector)
+ closeConnection()
PatientFileLoaderAdapter
PatientFileLoader
- _file: std:string
+ PatientFileLoader(std::string& file)
+ loadPatientFile(): std::vector<Patient*>
PatientDatabaseLoader

Design Pattern: Adapter Pattern

Loading patients from a database and loading from a .txt file requires vastly different logic. Loading from text files requires string manipulation with string streams. As well as a different implementation of loadPatients(). The PatientFileLoaderAdapter will take care of added the patients into the system, while the PatientFileLoader is in charge of parsing the .txt file and returning a vector of Patient objects back to the PatientFileLoaderAdapter.

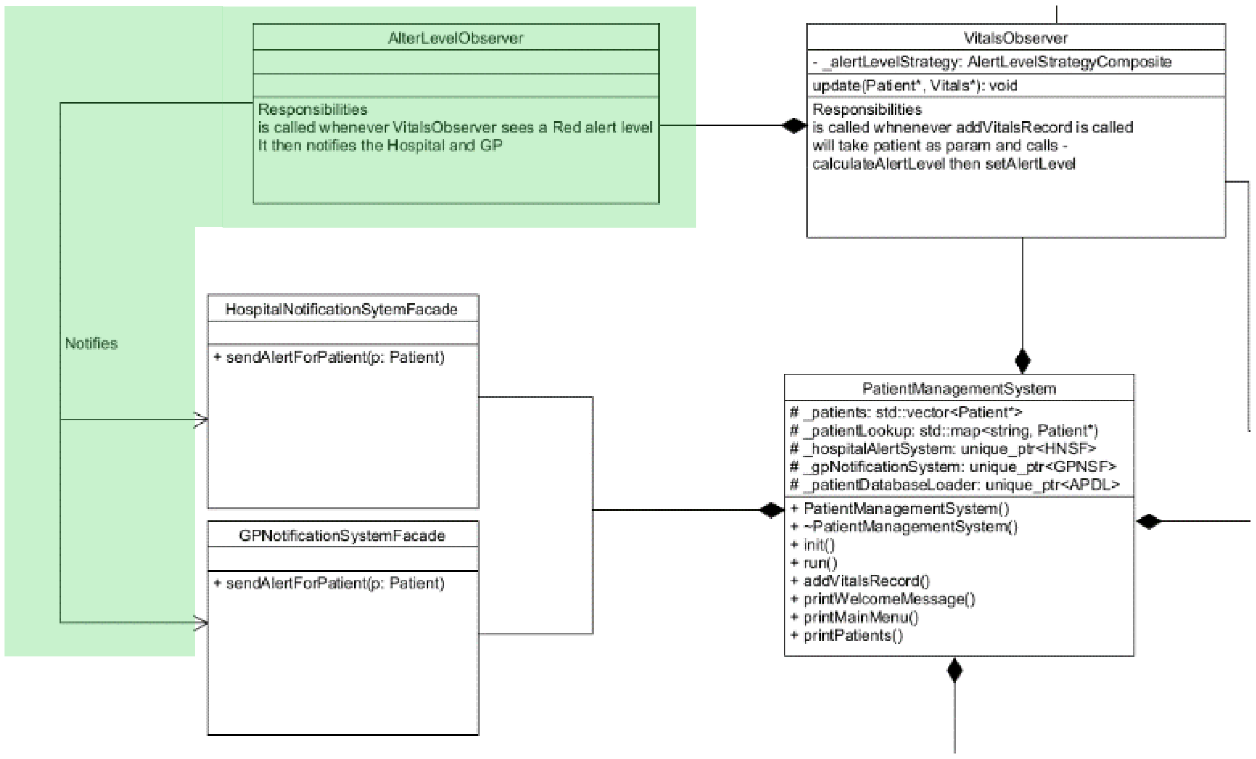
How it Works:

1. PatientManagementSystem calls the loadPatients() function of PatientFileLoaderAdapter.
2. PatientFileLoaderAdapter then initialises PatientFileLoader with the file's name.
3. Next, PatientFileLoaderAdapter calls the loadPatientFile() of PatientFileLoader which parses the contents of the file and creates a vector of patients which will be returned to the Adapter
4. Finally, PatientFileLoaderAdapter takes the returned vector and adds each Patient data in the vector to the _patients (patientsIn) member variable of PatientManagementSystem.

Git Commits:

- The PatientFileLoaderAdapter and PatientFileLoader was implemented in commit babfcb

FR4: Alert the hospitals and GPs


AlterLevelObserver
Responsibilities
is called whenever VitalsObserver sees a Red alert level
It then notifies the Hospital and GP
VitalsObserver
- _alertLevelStrategy: AlertLevelStrategyComposite
update(Patient*, Vitals*): void
Responsibilities
is called whnenever addVitalsRecord is called
will take patient as param and calls -
calculateAlertLevel then setAlertLevel
Notifies
HospitalNotificationSytemFacade
+ sendAlertForPatient(p: Patient)
GPNotificationSystemFacade
+ sendAlertForPatient(p: Patient)
PatientManagementSystem
# _patients: std::vector<Patient*>
# _patientLookup: std::map<string, Patient*)
# _hospitalAlertSystem: unique_ptr<HNSF>
# _gpNotificationSystem: unique_ptr<GPNSF>
# _patientDatabaseLoader: unique_ptr<APDL>
+ PatientManagementSystem()
+ ~PatientManagementSystem()
+ init()
+ run()
+ addVitalsRecord()
+ printWelcomeMessage()
+ printMainMenu()
+ printPatients()

Design Pattern: Observer

The criteria for sending an alert to the hospital and the GP is that the patient must have an Alert Level of Red. To do so, I implemented an observer class named AlertLevelObserver that is notified whenever VitalsObserver (another observer class) was returned an Alert Level of red when it called calculateAlertLevel on the composite strategy

How it works:

1. VitalsObserver is returned a new Alert Level of Red for a patient. It then calls the AlertLevelObserver's notify() function and passes the Patient object which will be used later with the Facades.
2. AlertLevelObserver then calls the HospitalNotificationSystemFacade's as well as the GPNotificationSystemFacade's sendAlert function and passes the Patient object to those functions
3. The facades are then run which send the alert to the Hospital as well as the GP

Git Commits:

- The AlertLevelObserver was fully implemented in commit cf703d
- Vitals observer was also refactored to notify AlertLevelObserver whenever an alert level red was detected in commit cf703d