University of
South Australia

UniSA STEM

COMP 3023– Design Patterns with C++

Assignment 2: Patient Vitals Management System.

Weighting: 20%,
Due Date: See course website

Version	Date	Notes
1.0	14 May 25	Initial release

Problem statement

In this individual assignment, you have been tasked by your client to complete a patient management system. An increase in the occurrences of diseases once thought fictional (Kepral's Syndrome, Andromeda Strain, and the dreaded Cordyceps Brain Infection) have made this project of utmost urgency. The patient management system should record patient vitals (body temperature, blood pressure, heart rate, and respiratory rate) and report alerts for concerning results dependent on the disease they have.

The system should load a list of patients. Currently this occurs from a database (mocked for this assignment), but you will change it to support loading from a file. Patients have vitals that are periodically measured. The system should raise an alert level on the patient when vital are recorded that exceed certain limits dependent on the disease they have. Use the following table to determine how to calculate the alert level:

Table 1: alert level calculations

Disease	Vitals	Alert level
Cordyceps Brain Infection	Respiratory rate (RR) > 40	RED
	Respiratory rate (RR) > 30	ORANGE
	Respiratory rate (RR) > 20	YELLOW
Kepral's Syndrome	Age < 12 and heart rate (HR) > 120	RED
	Age >= 12 and heart rate (HR) > 100	RED
Andromeda Strain	Blood pressure (BP) > 140	RED
	Blood pressure (BP) > 130	ORANGE
	Blood pressure (BP) > 110	YELLOW

The alert levels are ranked ordered as Green, Yellow, Orange, Red, with Green being the lowest alert level. A patient defaults Green alert level unless their vitals raise their level.

Assignment overview

Part of the project has already been developed. You must adapt and build upon the provided project. This requires not only understanding the existing codebase and writing new code, but also understanding the **existing design and being able to explain your design changes**. Read through this entire assignment specification before you begin, as you are required to explain many aspects of what you have done later in the assignment.

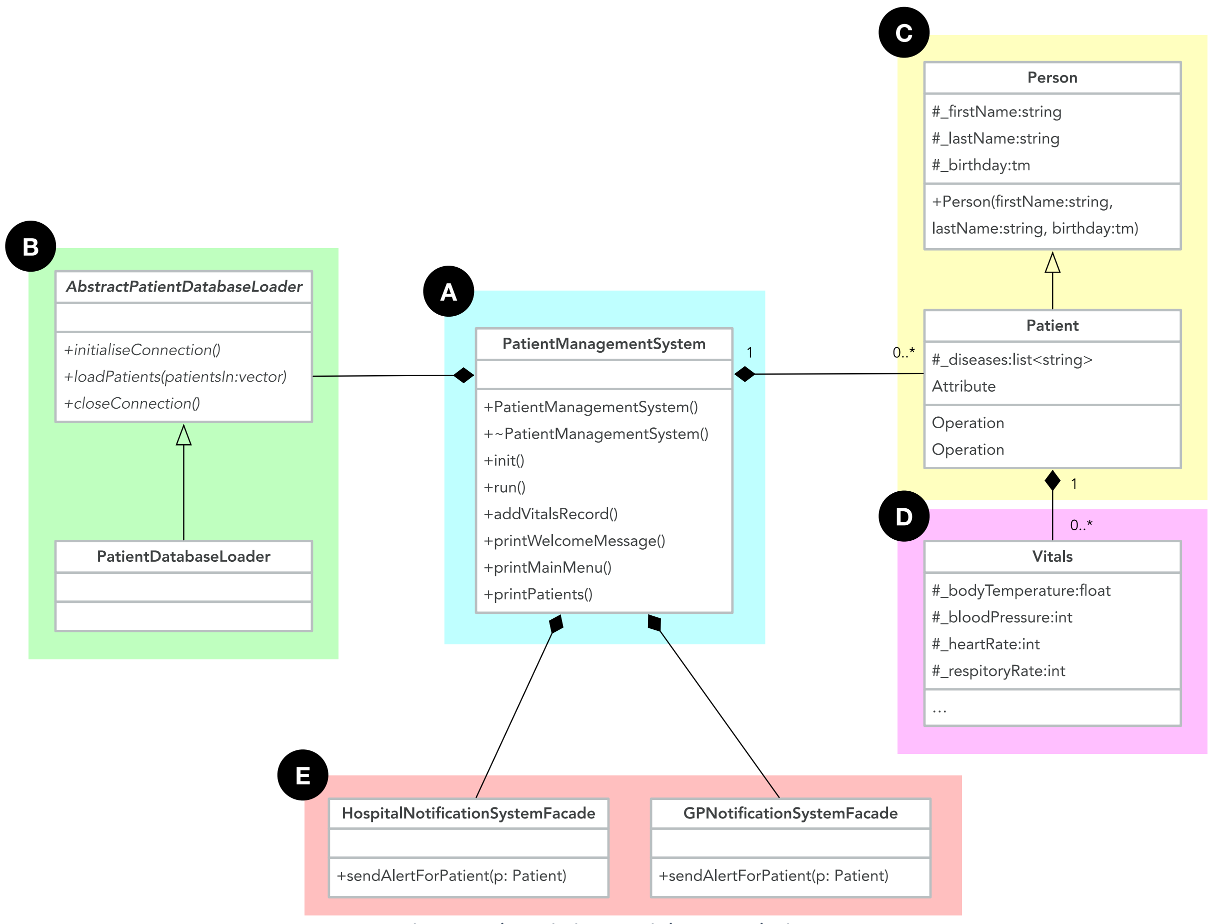

Figure 1: the existing partial system design

The existing system in presented in Figure 1. The main driver of the program is the Patient Management System class Ⓐ. The system can currently connect to a patient database (mocked for this assignment) using Ⓑ and load patients into the system. Patients are represented by the Patient class Ⓒ. The system can currently record patient Vitals Ⓓ through a command line interface. There is little error checking on this interface—we can presume the user will always enter the correct input. Lastly, Façade interfaces have been provided to notify Hospitals and GPs of any patient Alert Levels of concern Ⓔ, though these still need to be integrated into the system process.

In this assignment, you will address functional requirements, non-functional requirements, and document your design and implementation. There are four major functional requirements you must address (FR1–FR4). These requirements will require you to write C++ code that will have you load patients from a file (FR1), load patients from the file and database (FR2), write correct algorithms to identify patient alert levels when they record new Vitals measures (FR3), and alert the hospitals and GPs when a patient has a Red alert level (FR4). Then you will present your design and implementation for each of these requirements in the form of a design document.

You must implement four design patterns throughout this assignment:

- Adapter pattern
- Composite pattern
- Observer pattern
- Strategy pattern

Each of these patterns will map to *one* of the major functional requirements of this assignment.

Each of the functional requirements are presented in the following Assignment Requirements Tasks section, along with a description of the required documentation. The section General requirements then describes the general requires that must be adhered to throughout the assignment.

Learning outcomes

After completing this assignment, you will have learnt to:

- Apply design patterns to solve design problem (CO2, CO2).
- Modify and expand a system in C++ using an object-oriented design (CO2, CO3).
- Write C++ code using modern C++ memory management (CO4).
- Write safe C++ code (CO4).

Assignment Requirements Tasks

This section presents the specific requirements you must address to complete the assignment.

Functional requirement FR1: Load patients from file

The system should load patient data from a file:

- You must use the `PatientFileLoader` class to load patient data from the file.
- The `PatientFileLoader` class is incomplete; you must complete it to correctly load from the file.
- Do not change the interface of the `PatientFileLoader` (i.e. the header file).
- Implementing this into the design must use the `AbstractPatientDatabaseLoader` interface.
- A file is provided (patients.txt) that contains the patients to load.
- The new loader can replace the existing loader specified in the `PatientManagementSystem`, but it should be trivial to switch back to the original method (e.g. one line of code change).

Complete these steps:

1. Consider what design pattern can be used to address this problem.
2. Modify the design class diagram to factor in the pattern.
3. Complete the `PatientFileLoader` so that it loads the patient data from file.
4. Implement the design changes to address the problem.

Functional requirement FR2: load patients from file *and* database

The system should load patients from both the database *and* file:

- When the system runs, patients in the database *and* in the text file should appear in the patient list.
- The database should be loaded first and then the file.
- It should be trivial to change between loaders: database, file, or database+file — where a trivial change is considered one line of code.
- Your code repository must default to database+file.

Complete these steps:

1. Consider what design pattern can be used to address this problem.
2. Modify the design class diagram to factor in the pattern.
3. Implement the design changes to address the problem.

Functional requirement FR3: calculate the patient alert levels

The system should use the appropriate algorithms for calculating a patient's Alert Level dependant on the patient's disease:

- Each patient will have a different algorithm dependant on their diseases. Reference Table 1 when writing your algorithms.
- The alert level must be calculated whenever **new** `Vitals` is recorded for a patient.
- The alert level must *not* be calculated for **historical** patient data loaded from a database or file.

Complete these steps:

4. Run the current system and record a Vitals for a patient. This will help you understand the current functionality.
5. Consider what design pattern can be used to address this problem.
6. Modify the design class diagram to factor in the pattern.
7. Implement the design changes to address the problem.

Functional requirement FR4: alert the hospitals and GPs

The system should notify stakeholders when a patient's alert level reaches Red:

- Façade interfaces (`HospitalNotificationSystemFacade` and `GPNotificationSystemFacade`) are provided that can alert hospitals and GPs of concerning patients. You need to connect these facades to changes in patient alert levels.
- This notification should happen immediately when a patient's Alert Level changes.

Complete these steps:

1. Consider what design pattern can be used to address this problem.
2. Modify the design class diagram to factor in the pattern.
3. Implement the design changes to address the problem.

Non-functional requirement NFR1: document your design

This last non-functional requirement requires you to document your design decisions and implementation changes. The aim of this is to communicate what you have done to a reader. Your document should look professional, have clear sections, and be written clearly.

Complete these steps:

1. Create a document that will present your design.
2. Add a section that presents a complete design class diagram of the final design.
    a. Highlight every aspect (classes, relationships, etc) that you have added to the design.

3. For each major Functional Requirement (FR1, FR2, FR3, FR4), add a named section that presents how you addressed that requirement from a design and implementation perspective:
    a. **Name** the design pattern you chose.
    b. Explain **why** you applied that particular design pattern (one paragraph).
    c. **Show** a class diagram that highlights the specific classes in the system and relationships that address that design pattern.
    d. Explain, step by step, **how** your design works. A numbered list may work best here. Reference your diagrams as necessary.
    e. Reference the git commits by commit ID where you addressed this aspect of the design.

    Note: an example of a reasonable design description for this section is presented in Appendix 2.

General requirements

This section presents general requirements for the assignment that must be adhered to.

Design constraints

The following design constraints must be adhered to in your code:

DC1. The Visual Studio 2022 solution you submit must compile and run.

DC2. C++ standard library collections must be used for storing collections of objects.

DC3. You must use dynamic memory where appropriate (e.g. new and delete or smart pointers).

DC4. Dynamically allocated memory must be cleaned up correctly.

DC5. Function parameters must use references appropriately.

DC6. Collections of pointers must be cleaned up correctly.

DC7. Class member functions should be declared const unless they cannot be.

DC8. Initialiser lists must be used for constructors where initialisation is required.

DC9. The code style guide available on the course website[1] must be followed with the following exception: you can decide whether to place opening braces on the statement line or the following line.

DC10. Your system must not use external libraries (e.g. BOOST).

Assignment and submission requirements

The assignment has the following general requirements. Failing to address any of these requirements will result in deducted marks.

1. The assignment must be submitted via LearnOnline.
2. You must submit three files: 1) a zipped GIT repository, 2) a functioning executable (.EXE), and 3) the system design document.
3. These files must be submitted as three separate files to LearnOnline.
4. The GIT repository must:
    a. Include the full GIT history;
    b. Contain your Visual Studio 2022 Solution and project files.
    c. The repository must be compressed using the common ZIP compression.
5. The functioning executable must:
    a. Run from command line under Windows 10 or higher.
6. The system design document must:
    a. Be submitted as PDF format.
    b. Note: it will also exist as Word document or similar in your repository.

[1] http://codetips.dpwlabs.com/style-guide

Workplan

The following workplan is suggestion. First, do not wait until the due date to start the assignment; under the “GIT” marking criteria, we are expecting to see consistent commits each week. You can progressively build the system as your understanding of design patterns develops.

The functional requirements have been presented in a logical order for implementation. As you implement each requirement, document the design and implementation. When you have finished, review your design document and update to correctly reflect your code.

Constantly commit to GIT. Use it as a safety net so that you can always revert to a working version of your project.

Marking Scheme

Criteria	Mark
Design document (5%)	
Overall presentation – Presentation, consistent visual style, spelling and grammar, diagrams legibility.	5%
Design (55%)	
Each of these criteria are marked from the design document. They will be marked on 1) completeness of diagram and syntax, 2) correctly selected and applied design pattern, 3) clarity of explanation, 4) referenced git commits, and 5) correctly reflecting the code.	
FR1: Load patients from file	15%
FR2: load patients from file *and* database	10%
FR3: calculate the patient alert levels	15%
FR4: alert the hospitals and GPs	15%
Functionality and implementation (20%)	
Functionality – The program functions as expected. Patient alert levels are correctly reported. Patient data is correctly loaded. **Implementation –** The design patterns are implemented correctly.	20%
Non-functional requirements (20%)	
GIT – GIT has been used. Optimal marks are rewarded for 1) at least one commit each in Week 10, Week 11, Week 12 and 2) at least one commit per functionality in the system. Commit messages should use the imperative mood.	10%
Design constraints – The design constraints have been addressed in the code implementation.	10%

- You will be required to explain your design and code during week 13. Not being able to explain any of these elements can result in losing marks.

Extensions

Late submissions will be penalised according to the course outline unless an extension has been approved by the course coordinator. Refer to the course outline for further information regarding extensions.

Academic Misconduct

This is an individual assignment. Your submitted files will be checked against other student's submissions, and other sources, for instances of plagiarism. Students are reminded that they should be aware of the academic misconduct guidelines available from the University of South Australia website.

Deliberate academic misconduct such as plagiarism is subject to penalties. Information about Academic integrity can be found in Section 9 of the Assessment policies and procedures manual at: http://www.unisa.edu.au/policies/manual/

Appendix 1 — Example output

```
WELCOME TO HEALTHCO 3000
------------------------

Select an option:
1. List patients
2. Add vitals record
3. Quit
> 1
bj0280|Bloggs,Joe|18-02-1980|Cordyceps Brain Infection|37.5,80,60,16
jh1074|Jones,Henry|02-10-1974|Cordyceps Brain
Infection|37.5,80,60,16;37.5,82,15,16
mm05115|McCartney,Martha|24-05-2015|Kepral’s Syndrome|
tg0548|Taylor,George|15-05-1948|SimianFlu|37.2,80,60,12
zd01100|Zaius,Dr|01-01-2000|SimianFlu|
kk0890|Kroker,Kif|19-08-1990|Cordyceps Brain Infection|40.2,70,60,12;39.7,80,40,12

Select an option:
1. List patients
2. Add vitals record
3. Quit
> 2
Patients
bj0280|Bloggs,Joe|18-02-1980|Cordyceps Brain Infection|37.5,80,60,16
jh1074|Jones,Henry|02-10-1974|Cordyceps Brain
Infection|37.5,80,60,16;37.5,82,15,16
mm05115|McCartney,Martha|24-05-2015|Kepral’s Syndrome|
tg0548|Taylor,George|15-05-1948|SimianFlu|37.2,80,60,12
zd01100|Zaius,Dr|01-01-2000|SimianFlu|
kk0890|Kroker,Kif|19-08-1990|Cordyceps Brain Infection|40.2,70,60,12;39.7,80,40,12

Enter the patient ID to declare vitals for > jh1074
enter body temperature: 36.3
enter blood pressure: 88
enter heart rate: 60
enter respitory rate: 55
Patient: Jones, Henry (jh1074)has an alert level: Red

This is an alert to the hospital:
Patient: Jones, Henry (jh1074) has a critical alert level

This is an notification to the GPs:
Patient: Jones, Henry (jh1074) should be followed up

Select an option:
1. List patients
2. Add vitals record
3. Quit
> 2
Patients
bj0280|Bloggs,Joe|18-02-1980|Cordyceps Brain Infection|37.5,80,60,16
jh1074|Jones,Henry|02-10-1974|Cordyceps Brain
Infection|37.5,80,60,16;37.5,82,15,16;36.3,88,60,55
mm05115|McCartney,Martha|24-05-2015|Kepral’s Syndrome|
tg0548|Taylor,George|15-05-1948|SimianFlu|37.2,80,60,12
zd01100|Zaius,Dr|01-01-2000|SimianFlu|
kk0890|Kroker,Kif|19-08-1990|Cordyceps Brain Infection|40.2,70,60,12;39.7,80,40,12
```

```
Enter the patient ID to declare vitals for > bj0280
enter body temperature: 36
enter blood pressure: 88
enter heart rate: 66
enter respitory rate: 35
Patient: Bloggs, Joe (bj0280)has an alert level: Orange

Select an option:
1. List patients
2. Add vitals record
3. Quit
> 2
Patients
bj0280|Bloggs,Joe|18-02-1980|Cordyceps Brain Infection|37.5,80,60,16;36,88,66,35
jh1074|Jones,Henry|02-10-1974|Cordyceps Brain
Infection|37.5,80,60,16;37.5,82,15,16;36.3,88,60,55
mm05115|McCartney,Martha|24-05-2015|Kepral's Syndrome|
tg0548|Taylor,George|15-05-1948|SimianFlu|37.2,80,60,12
zd01100|Zaius,Dr|01-01-2000|SimianFlu|
kk0890|Kroker,Kif|19-08-1990|Cordyceps Brain Infection|40.2,70,60,12;39.7,80,40,12

Enter the patient ID to declare vitals for > mm05115
enter body temperature: 39
enter blood pressure: 80
enter heart rate: 130
enter respitory rate: 12
Patient: McCartney, Martha (mm05115)has an alert level: Red

This is an alert to the hospital:
Patient: McCartney, Martha (mm05115) has a critical alert level

This is an notification to the GPs:
Patient: McCartney, Martha (mm05115) should be followed up
```

Appendix 2 — Example output

This appendix presents an example of what is expected for the design document.

FR0: The system should load patients from the database

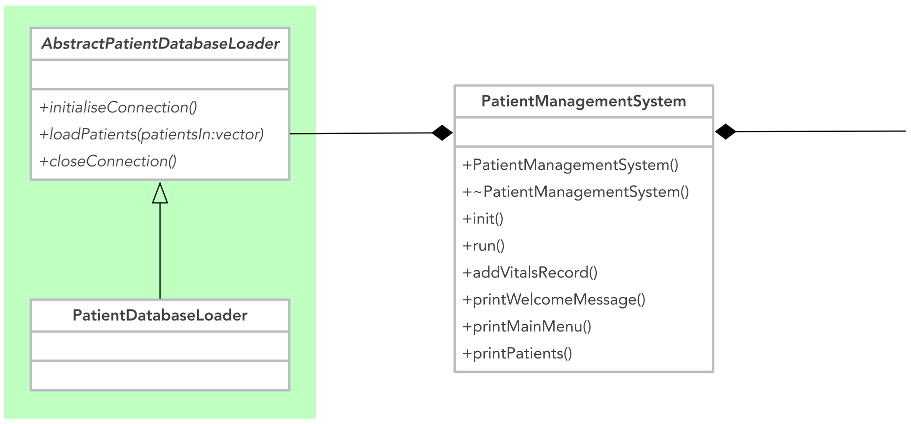

Figure 2: Bridge design pattern

Design pattern: Facade design pattern

The system must load patients from a database. The underlying database system is a complex set of interfaces[2] and classes that can be difficult to use. To address this, I apply the Façade design pattern which wraps a complicated subsystem in a simpler interface that exposes only certain aspects of the system.

The Facade is represented in Figure 2. The primary Façade class is the `PatientDatabaseLoader`. This class interfaces between the database and the `PatientManagementSystem`. To facilitate extensibility, I have created an abstract interface (`AbstractPatientDatabaseLoader`) that could be used to create new loaders.

How it works:

1. The `PatientManagementSystem` will initialise the `PatientDatabaseLoader` by calling `initialiseConnection()` – this handshakes with the database.
2. The `PatientDatabaseLoader` will call `loadPatient(…)` passing in a vector which will be loaded with patients.
3. The concrete `PatientDatabaseLoader` will retrieve the patient details and construct patient objects which are added to the vector.
4. The `PatientManagementSystem` is then responsible for release Patient memory allocated by the `PatientDatabaseLoader`.
5. Finally, the `PatientManagementSystem` must explicitly call `closeConnection()` to close the database connection.

Git commits:

- I first added the `PatientDatabaseLoader` in commit 98fe20.
- I refactored and added the abstract interface in commit 38edd0.
- Other commits include: 40edd0, 222d0.

[2] Though this is mocked for assignment purposes