THOMPSON RIVERS UNIVERSITY

Software Design Patterns (Fall 2021) - SENG 4130

Health Spy

Cem Doganay (T00582940)

James H. Howe (T00562601)

December 6, 2021

Table of Contents

List of Figures

1 Introduction

Being able to track your health information in real time is essential for people with chronic health issues. By sharing this information on the Cloud, doctors and family members of the patient can also access the health information of the patient in real time. In case of an emergency, all users or viewers will be alerted instantly, which gives the family of the patient peace of mind and enables emergency services to be notified immediately. Also, the doctor responsible for the patient can analyze if the patient's health is getting better or worse by tracking the data.

This project intends to provide a system for tracking patient's health information and displays them onto three separate screens. One of them is the User Screen which is the display shown to the patient. The second screen is the Home Screen allowing family members of the patient to monitor the condition of their loved one. The third screen is the Hospital Screen which shows all hospitalized patient's information at a central location to be able to easily monitor every patient.

Several design patterns are considered when developing the Health Spy system to achieve expandability, robustness, flexibility, and consistency. The advantages of using design patterns is it provides a common vocabulary for communication and reduces confusion of the object interactions while writing the code.

Section 2 establishes the problem definitions and the design requirements. Two potential solutions are discussed in section 3 followed by the detailed analysis of the selected final solution for Health Spy. The teamwork distribution and project management are shown in sections 4 and 5 respectively. Finally the concluding remarks and future improvements are discussed in section 6 followed by section 7 including the appendix.

2 Design Problem

2.1 Problem Definition

Health of those who have chronic conditions and are not hospitalized can be at great risk. Their health can rapidly deteriorate and some have become victims of not being able to report to the emergency authorities quickly enough to get the correct treatment. The family members or friends of the patient might not be able to notify emergency services depending on the situation as well. Health Spy can help reduce those unfortunate situations with a Cloud solution. The aim of Health Spy is to notify all users that are monitoring the patient and track their health conditions in real time. Health Spy takes into account object oriented design patterns for the software development. The design patterns are used to aid and simplify the coding process.

2.2 Design Requirements

The main responsibility of the Health Spy software is to relay crucial health information to the Cloud for doctors, patients, and the patients' families to have access to the information. Health Spy has three different screens to display the health information: the Patient Conditions screen,

the Patient Statistics screen, and the Warning screen. Patient's health data accumulates in the Cloud for analyzing and prediction of changes in health conditions.

2.2.1 Functions

- Health Spy must communicate with the physical base station to receive the patient's health information.
- Health Spy must display the current conditions onto three separate View screens.
- Health Spy shall display the statistics of the patient's health information onto the Patient Statistics screen.
- Health Spy must calculate the statistics of the patient's health information based on the patient data archive stored in the physical base station.
- Health Spy must display dangerous conditions of patients that need urgent attention onto the View screen.
- Health Spy must analyze signs of moderate or rapid decline in the patient's health conditions.
- Health Spy must send patient health data to the central hub of the hospital and the family of the patient.

2.2.2 Objectives

- Accurately track the health of the patient by constantly getting data from the sensors.
- Displaying the tracked health data on the user's device.
- Sharing the real time health information with the patient, hospital, and family of the patient.
- Alerts the family and the hospital staff in case of an emergency.

2.2.3 Constraints

- The Health Spy system updates its displays only when there is new data available in the physical base station.
- Health Spy cannot replace the existing hardware, it improves the software aspect of the system.
- Health Spy cannot send and receive data to other users or devices other than the patient when there is no internet connection.
- Health Spy cannot operate when there is no power supplied.

3 Solution

There are a total of three solutions considered for Health Spy. Each solution is an improvement from the previous one with a new design pattern added to meet the constraints while implementing all features. The three solutions will be described in detail, followed by the discussion of features. The environmental, societal, safety, and economic considerations will be mentioned thereafter and the limitations will conclude this section.

3.1 Solution 1

The first solution considered is to implement the Observer design pattern. The Observer pattern has two major components: Subject, and Observer. We have also made some modifications with the observer pattern by adding a "Sensor" interface that creates new sensors like body temperature, heart rate and blood sugar. These sensors will notify the "Base_Station" class when there is a new data reading. "Base_Station", a class that is created from the "Subject" interface, represents the patient and will be responsible for collecting the data, registering/removing observers and notifying the observers. Observers are generated from the "Observer" Interface. These observers are: "Hospital_View", "Home_View" and "User_View". Each observer will get updated when there is a change in the temperature, heart rate and blood sugar sensors. The UML diagram shown below is the implementation of the Health Spy using the Observer Pattern.

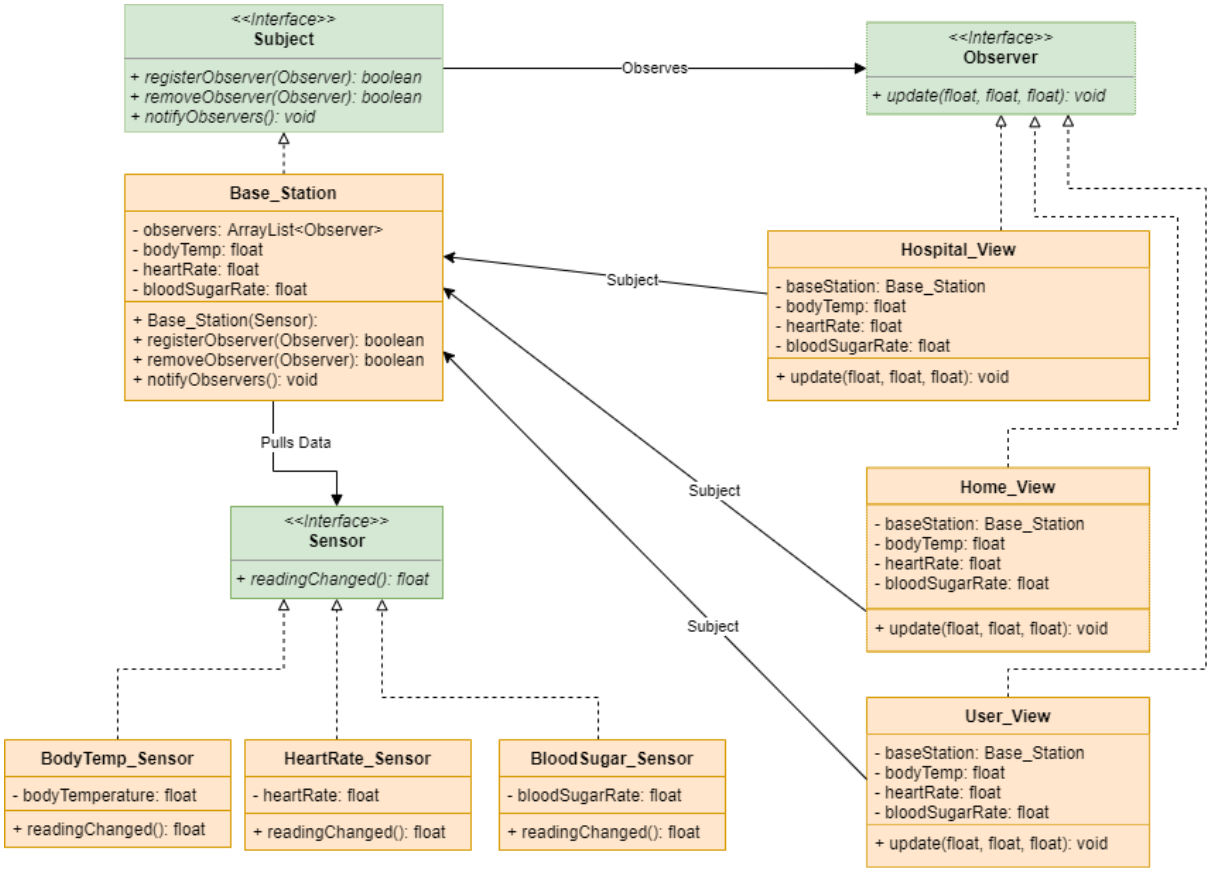

Figure 3.1.1: Solution 1 Observer Pattern UML Diagram

There are some limitations with this approach. First, adding a new sensor will result in modifying the "Base_Station" class, "Observer" Interface and every single view class, which we need to avoid. In addition, we are displaying the data of all sensors on all views, which might not be ideal. For example, "Home_View" might not need to display all sensor information. The user's family might just want to know if the health status of the patient is good. The next solution will try to overcome these limitations.

3.2 Solution 2

The second solution considered is to implement the Iterator design pattern in conjunction with the Observer pattern. The Sensor_Iterator interface is implemented by each of the three views: Hospital_View, Home_View, and User_View objects. The notification of observers follows the same procedure as in Solution 1. A change in any of the Sensor objects will trigger the Base_Station to notify all Observers. When the Base_Station notifies the Observers, each observer will get an up-to-date ArrayList of Sensors. This is the difference from the previous solution where the observers were getting a concrete list of three float values. After the update, each of the observers will iterate through the ArrayList of Sensors and update its values to display. The figure below shows the UML diagram of the proposed solution.

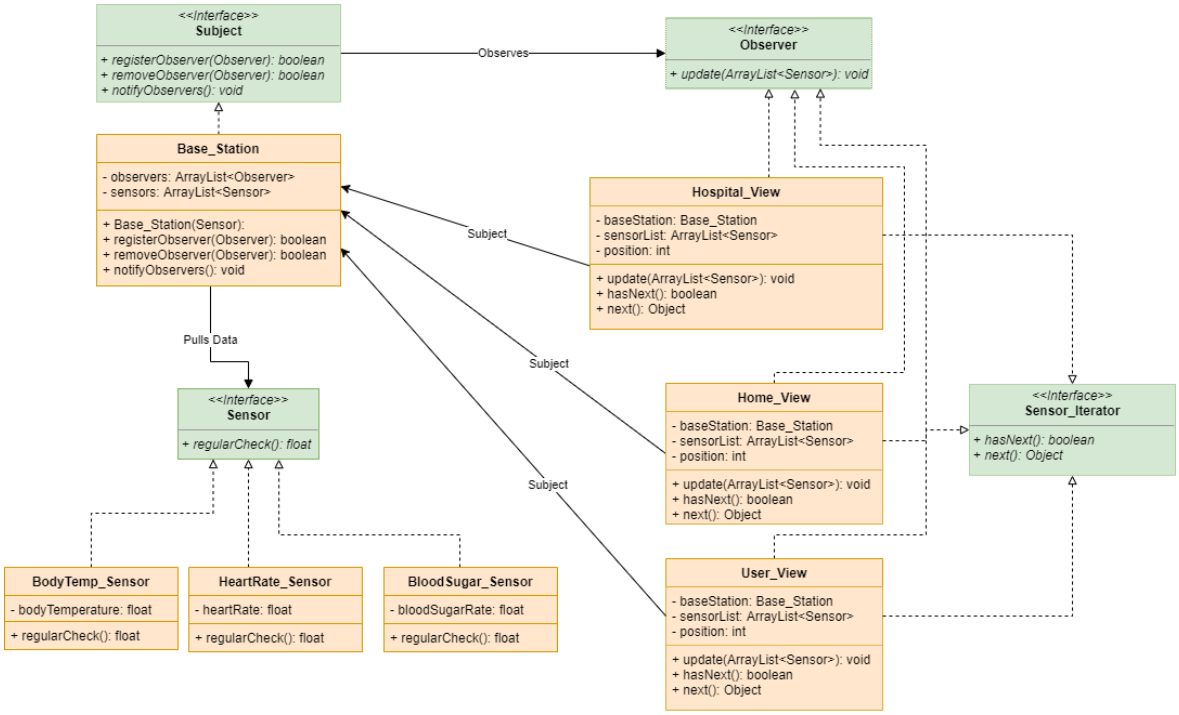

Figure 3.2.1: Solution 2 Observer & Iterator Patterns UML Diagram

Although an improvement to the previous solution, this solution also has a drawback. Every time there is a new patient with this system, the entire structure shown in Figure 2 is instantiated with the current setup. It makes sense that a new Home_View and User_View objects are created with each patient as a separate view is needed for different patients. However the Hospital_View is meant to be a central hub for the hospital workers to view every patient's data. Every time a new patient is created in the system a new Hospital_View object will also be instantiated with the solution given here. This is unnecessary and in fact undesirable since the hospital staff will have to find a specific patient from a pool of objects just to get their health data. It does not make sense that there are multiple Hospital_View objects instantiated when only one is wanted. The final solution will address that issue and discuss how the improved design is better than the combined Observer and Iterator patterns.

3.3 Final Solution

The final solution considers to implement the Singleton design pattern on top of the Observer and Iterator patterns combined. The Singleton pattern ensures that a certain object does not have more than one instantiation during runtime. Since there is no need to create a new Hospital_View for each patient, Hospital_View implements a Singleton pattern. This is accomplished by having a reference static object of Hospital_View declared within the class, and making the constructor private. The only way to instantiate the Hospital_View is to call the getInstance method which will only instantiate a new Hospital_View if the static reference is null. If it is null, a new instance is created. However, the method will return the static reference which is the variable named uniqueInstance if the method has been called before. Figure 3.3.1 shown below is the implementation of Health Spy using the Observer, Iterator, and Singleton patterns.

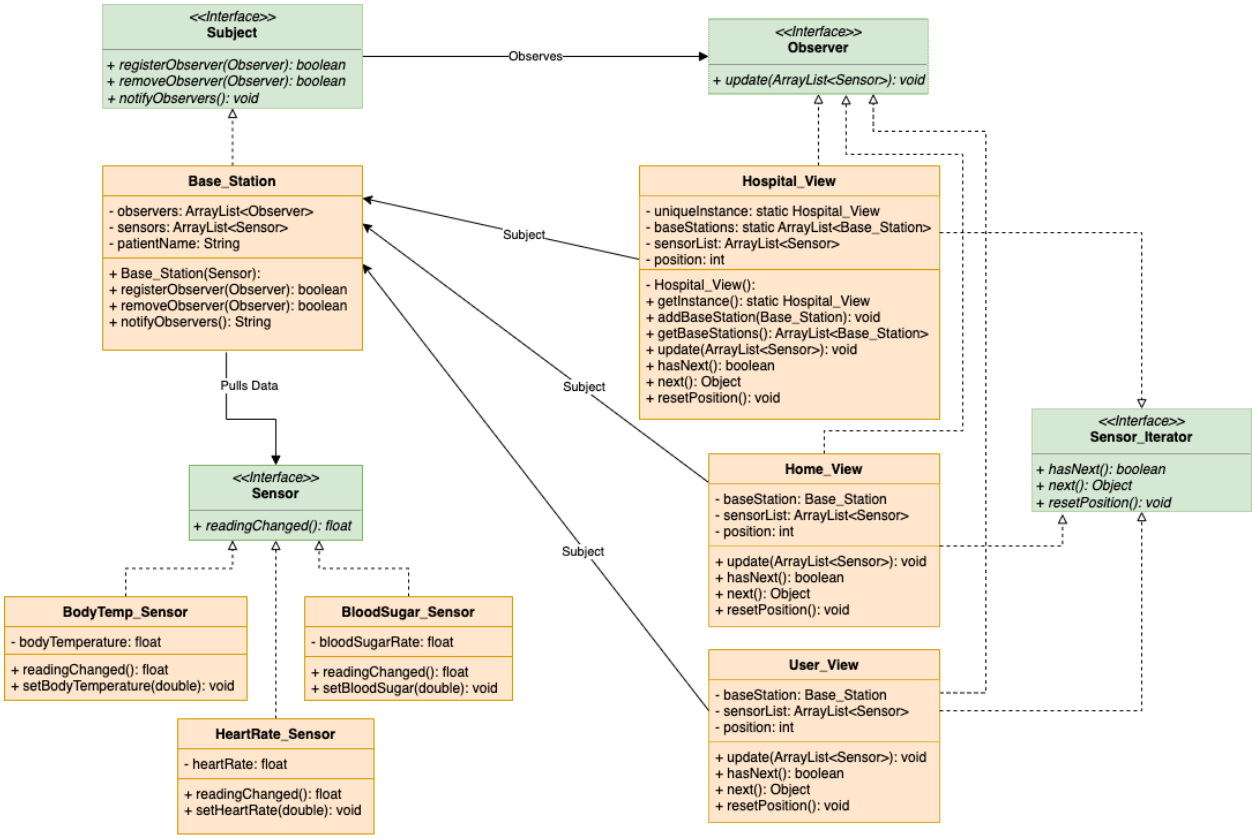

Figure 3.3.1: Final Solution Observer & Iterator & Singleton Patterns UML Diagram

This solution addresses the previous problems and implements all of the aforementioned requirements, therefore the combination of Observer, Iterator, and Singleton patterns were chosen as the final solution.

Additionally, Health Spy requires a Graphical User Interface (GUI) for each of the View objects to display data. Java Swing has been chosen to host the GUI as the libraries provided are powerful and allows detailed customization for Health Spy's needs. Including the Java Swing libraries in fact is implementing the Template design pattern into the design. Java Swing library

has a large structure where many levels of object extensions are made. The structure follows the Template pattern where the methods of the superclass are either simply used by the subclasses or overridden and newly defined. Below is a sample UML diagram showing how the JFrame object is utilizing the templates of the superclasses, especially with a focus on the paint() method.

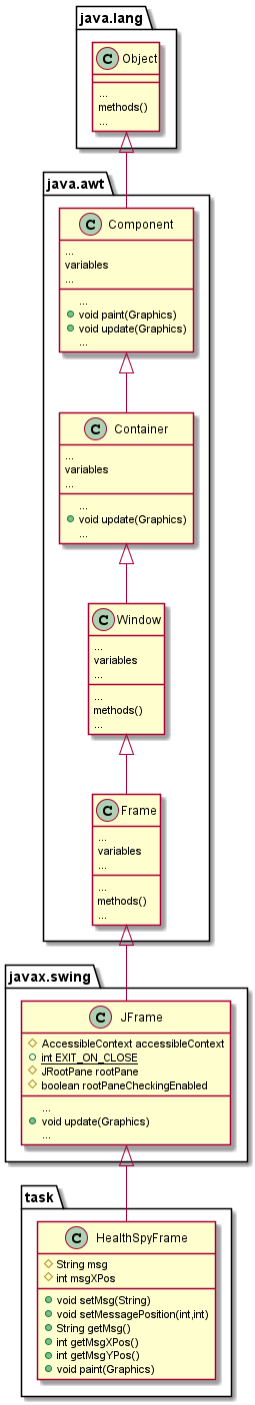

Figure 3.3.2: Template Design Pattern UML Diagram on JFrame

The graphical user interface is organized in a simple design. Figure 3.3.3 shows a screenshot of an example result of running Health Spy. The User_View window has the patient's name included in the window title on the top left corner. Inside of the window there are three rows with six text boxes for each row. The rows correspond to the particular user's heart rate, temperature, and blood sugar rate readings coming from the sensors. The first column denotes the readings of that row along with the units of the measurements, and can change colours of either red or green. A green box indicates that the reading for that health metric is within normal bounds whereas a red box indicates that the reading for that health metric is abnormal and some medical attention may be needed. The following five boxes are where the past five measurements are recorded. The readings are newest on the far left, and move to the right as time passes with new measurements to display. The Home_View window looks very similar to the User_View window with one exception. The blood sugar rate reading is replaced with a single textbox instead of five most recent readings. The textbox only shows whether the recorded measurement is a good one within the threshold or not. This is to easily identify if the patient is healthy or not without having to know the knowledge of what is considered to be normal. The Hospital_View combines all patient's User_View windows into one screen and displays all the health information. The addition made after combining all User_View windows is another row before the measurement rows to denote which reading sets belong to which patient.

Figure 3.3.3: Example Screenshot of Running the Health Spy App

3.3.1 Features

The features of Health Spy can be derived from the final solution shown above in the previous section. The design patterns used enable the system to implement object-oriented principles. Below is a list of features of Health Spy on how the design can realize them.

3.3.1.1 Base Station

The base station is the hub responsible for acquiring each patient's health data and notifying the Observer objects about the change in data. It maintains an ArrayList of observers and sensors which are the physical devices that will display and acquire the health data respectively. The Base_Station object is able to add or remove observers and sensors after instantiation if necessary as the observer pattern does so. When a change in value is measured in one of the Sensor objects, a method will be executed that will notify all the Observer objects of the change.

3.3.1.2 Sensors

The physical sensors will be attached to the patient's body to monitor them and to gather information related to their health. The Sensor interface provides the basis method for each physical sensor to implement in order to be able to send the information into the system. Each concrete Sensor implementation must correspond to one metric. As an example, the sensors included for the current report are BodyTemp_Sensor, HeartRate_Sensor, and BloodSugar_Sensor. Each of these sensors report on one particular value: the body temperature, the rate of heartbeat, and the blood sugar levels. Any sensors that are added to the system must follow this characteristic in order for Health Spy to function. For the purpose of testing, a simulation of changing values will be fed to each of the Sensor classes instead of hooking it up to a physical device.

3.3.1.3 Displays

The data acquired from the physical sensors will be displayed onto the Observer objects via the Base_Station object. It will update the on-screen values whenever the Base Station detects a change in the Sensors. These Observer objects also are Sensor_Iterator objects as well. When an update is triggered, the View objects will iterate through the Sensor ArrayList passed by the Base Station and acquire all data necessary for display. The iteration will only store the data that is required to display for as each view requires different health data to show. There will be multiple Home_View and User_View objects for each patient, but only one Hospital_View object for the entire system. The Singleton pattern ensures that there is only one Hospital_View instantiated during the runtime. This is because the hospital only needs one central hub to view from the main office. In addition, the User_View display will be in each patient's room for doctors or nurses to monitor as well. For the purpose of testing, a GUI will be used to display the data on a computer screen. However, all three views will show on the same screen instead of each screen showing on separate devices.

3.3.2 Environmental, Societal, Safety, and Economic Considerations

Using interfaces will be beneficial when there will be a need to add a new sensor and new observers. New sensors and interfaces can be created from their interfaces without modifying the other parts of the code depending on the situation. Loosely coupled classes will reduce the time and cost of the modifications when there is a need. Also, creating a new user will be very easy thanks to the object oriented implementation. This will increase the scalability of the software.

Health Spy software will only do observation and will not take any action that will directly affect the health of the patient so there will not be a significant safety concern. Only possible scenario can be displaying wrong information on the observers due to a bug or a sensor malfunction. However, healthcare workers are expected to examine the patient with their own tools in case of an emergency and a malfunction won't cause any wrong diagnosis.

Observer pattern eliminates the polling by only updating when there is a new reading. Eliminating polling can significantly reduce the computational power usage as well as the electrical power usage. Our software is designed not to consume any resources when there are no readings from the sensors.

Health Spy respects and protects the health privacy of its customers. Our software’s views are not connected to each other thus it is not possible to access a view from a different view. It is also not possible to access a patient's health information from a view that is not connected to that patient. Furthermore, “Sensor” ArrayList is a private variable. It will not be possible to access the information stored into this variable directly.

3.3.3 Limitations

One of the major limitations of our software comes from “Hospital_View” being a singleton. With this implementation, it won't be possible to use this system for multiple hospitals on a single server since “Hospital_View" is hard coded. Every single “Base_Station” (Patient) will be connected to the same “Hospital_View”. A simple possible solution for this issue is to run Health Spy on separate servers for each hospital.

Another limitation of our software is not being able to create new observers during runtime. In order to add a new observer, we would need to create a new class from the “Observer” interface. Similarly, adding a new sensor during runtime will not be possible.

The developed system is currently hard coded in the main function with the GUI not expandable without rewriting code. The time constraint has rushed to have a working demo and can be improved with more time for development.

4 Team Work

4.1 Meeting 1

Time: September 24, 2021, 10:30 am to 12:00 pm
Agenda: Distribution of tasks for the Introduction and Design Problem sections.

Team Member	Previous Task	Completion State	Next Task
Cem Doganay	N/A	-	T1
James H. Howe	N/A	-	T2

4.2 Meeting 2

Time: September 25, 2021, 2:30 pm to 3:00 pm
Agenda: Review of individual progress and distribution of next tasks.

Team Member	Previous Task	Completion State	Next Task
Cem Doganay	T1	100%	T3
James H. Howe	T2	100%	T3

4.3 Meeting 3

Time: October 29, 2021, 2:30 pm to 5:00 pm
Agenda: Designing the UML diagrams.

Team Member	Previous Task	Completion State	Next Task
Cem Doganay	T3	100%	T4, T6, T7
James H. Howe	T3	100%	T4, T5

4.4 Meeting 4

Time: November 3, 2021, 3:30 pm to 8:00 pm
Agenda: Completion of the document write up for Section 3.

Team Member	Previous Task	Completion State	Next Task
Cem Doganay	T4, T6, T7	100%, 100%, 100%	N/A
James H. Howe	T4, T5	100%, 100%	N/A

4.5 Meeting 5

Time: November 20, 2021, 3:30 pm to 4:00 pm
Agenda: Discussion on how to do the implementation in IntelliJ.

Team Member	Previous Task	Completion State	Next Task
Cem Doganay	N/A	-	T8, T10
James H. Howe	N/A	-	T9, T10

4.6 Meeting 6

Time: November 27, 2021, 1:00 pm to 8:00 pm
Agenda: Write up code implementation for Health Spy.

Team Member	Previous Task	Completion State	Next Task
Cem Doganay	T8, T10	100%, 90%	T10, T11
James H. Howe	T9, T10	100%, 90%	T10, T11

4.7 Meeting 7

Time: November 28, 2021, 1:30 pm to 4:30 pm
Agenda: Completion of code implementation and create GUI for Health Spy.

Team Member	Previous Task	Completion State	Next Task
Cem Doganay	T10, T11	100%, 85%	T11, T12
James H. Howe	T10, T11	100%, 85%	T11, T12

4.8 Meeting 8

Time: December 1, 2021, 12:30 pm to 7:00 pm
Agenda: Completion of GUI specifics and create presentation.

Team Member	Previous Task	Completion State	Next Task
Cem Doganay	T11, T12	100%, 100%	T13
James H. Howe	T11, T12	100%, 100%	T13

4.9 Meeting 9

Time: December 6, 2021, 5:00 pm to 6:00 pm
Agenda: Completion of the report for submission.

Team Member	Previous Task	Completion State	Next Task
Cem Doganay	T13	100%	N/A
James H. Howe	T13	100%	N/A

5 Project Management

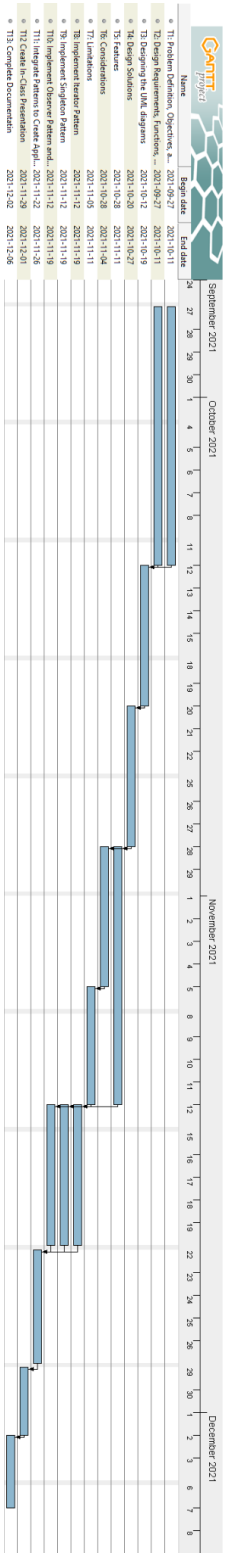

Figure 5.1: Project Gantt Chart (download image for better resolution)

6 Conclusion and Future Work

In conclusion, the final solution demonstrated includes four design patterns compounded into one system. The observer pattern is used for the base station to notify all of the view screens that are monitoring the particular patient's health information. The iterator pattern is used for the view screens to retrieve all health metrics that the base station passes along from the sensors to display on screen for people to monitor the health status of the patient. The singleton pattern is used for the hospital screen to ensure that only one instance of the hospital view exists in the entire runtime as having more than one hospital view does not make sense in Health Spy's system. The graphical user interface was coded using the template pattern provided by the Java Swing library which provided baselines for programming the screen interface.

Health Spy has successfully achieved the following functional requirements: communicate with the physical base station, display the current conditions onto three View screens, display dangerous conditions of patients that need urgent attention onto the View screen, and send patient health data to the central hub of the hospital and the family of the patient. The base station object is able to communicate to the view objects by passing the information retrieved from the sensor objects. The information is then updated on screen for people to see the most up-to-date conditions of the patient in their respective View screens. The dangerous conditions of patients are shown on screen by showing a green background around the health metric that is deemed within tolerance while a red background around the health metric indicating an abnormal reading. The patient's health data is sent to the central hub of the hospital and the family of the patient through the design patterns implemented for the Health Spy system and can be seen on the View objects which display onto physical screens.

The first objective of Health Spy is to track the health of the patient by constantly getting data from the sensors. This is achieved by the View screens receiving the updates from the base station whenever the base station notifies the screens of an update from the sensors. The second objective of Health Spy is to display the tracked health data on the user's device. This is achieved by the interface displaying the past five readings of the sensors on screen. The third objective of Health Spy is to share the real time health information with the user, hospital, and family of the patient. This is achieved by having three separate screens for each party to access the patient's health information. The fourth objective of Health Spy is to alert the family and the hospital staff in case of an emergency. This is achieved by indicating the irregular health metrics of the patient with colours so people can identify immediately of a dangerous condition.

In the future, Health Spy can be improved by displaying the statistics of the patient's health information on a dedicated section of the View. Health Spy can be improved by calculating the statistics of the patient's health information to analyze signs of moderate or rapid decline in the patient's health conditions as well. These features will give doctors and nurses more information about the patient's condition to make a better informed decision. A further recommendation for Health Spy is to include the GUI code into the View objects to promote loosely coupled designs.

7 Appendix

Task ID	Task Description
T1	Write up the Problem Definition, Objectives, and Constraints.
T2	Write up the Design Requirements, Functions, Objectives.
T3	Designing the UML diagrams.
T4	Write up the Design Solutions.
T5	Write up the Features.
T6	Write up the Environmental, Societal, Safety, and Economic Considerations.
T7	Write up the Limitations.
T8	Implement Iterator Pattern Code
T9	Implement Singleton Pattern Code
T10	Implement Observer Pattern Code and Template Pattern Code for GUI
T11	Integrate Patterns to Create Application
T12	Create Health Spy Google Slides for In-Class Presentation
T13	Complete Health Spy Report Document for Submission