COMP1029 Programming Paradigms Java Coursework Report

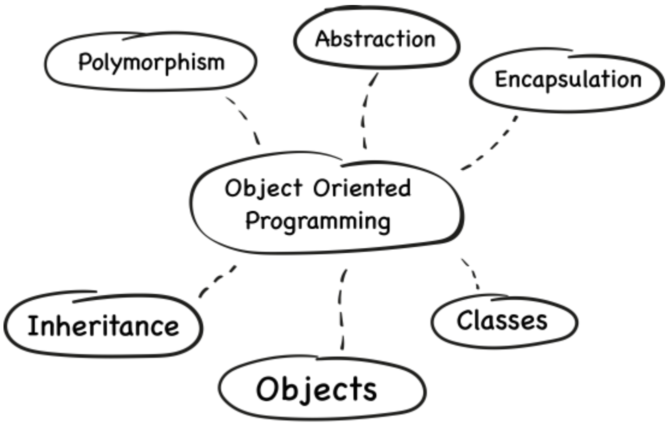
Name	: Lim Bing Qian
Student ID	: 20408309
OWA	: hfybl3
Course	: Computer Science with Artificial Intelligence
Semester	: 2 (Spring)
Module Convenor	: Doreen Ying Ying Sim

Table of content

No.	**Content**	**Page**
1	Introduction	2
2	Description - RestrictedSpots.java class - StaticDistancing.java class - DynamicDistancing.java class - Main.java class	2 - 5
3	Overall Experience	5
4	Conclusion	5 - 6
5	References	6

Introduction
In this programming paradigms java coursework, the main objective is to develop a human congestion management and crowd control system for a hospital using java programming to ensure that every spot in the hospital is not overcrowded by people as the Covid Pandemic just ended not long ago. Therefore, it is our job to make sure that the environment is controlled and safe so everyone gets to return to their pre-pandemic life without being much affected by the virus.

Java Programming is a popular programming language that focuses on object-oriented programming; therefore, the elements and concepts of OOP such as inheritance, polymorphism, encapsulation, and abstraction will be used in our programming coursework. For this coursework, all java coding content are done on Eclipse IDE and all classes are saved in a package called coursework, Main.java will be the class (file) that combines all other classes created in the package and it will execute the program.



In this report, an overview of the human congestion management and crowd control system, including its functionality, idea and structure will be talked about in the description section. It will also discuss the challenges faced when developing such a system and how I overcame them. The report will conclude with a discussion of the system's performance and limitations.

Description
Before starting with the code, what we need to implement in the program should first be identified. In this congestion management and crowd control system, there are two main features/functions to include and implement into the program. Firstly, it is the maximum capacity of people allowed in a spot at one time should be calculated and known, then check whether the current number of visitors inside the spot has reached its maximum before allowing the user to enter, else they will have to wait for a certain amount of time (spot permitted average time). Once the user is allowed to enter, the second main feature is to check the user’s distance with the people standing at the front, back, left and right of them, then determine the user’s health status.

RestrictedSpots.java class
To start off with the program, a class called RestrictedSpots.java is first created to create the details associated with each spot, they are the spot's id, name, area, permitted average time and maximum capacity. These variables are all declared and created as private variables to ensure that the RestrictedSpots class maintains control over the data and its manipulation. Then, there is a constructor to initialise the object's data members, the maximum capacity of each spot will also be calculated with the formula written in the constructor and the formula is based on the hospital's distancing rules and restrictions.

Below the constructor, a utility method called "static int visitorno()" is created to generate a random number of current visitors inside each spot, a java.util.Random class is imported on top of the code for the function of generating random numbers. Also, the "static int visitorno()" method is created with static so that it can be called without creating an object inside the RestrictedSpots class.

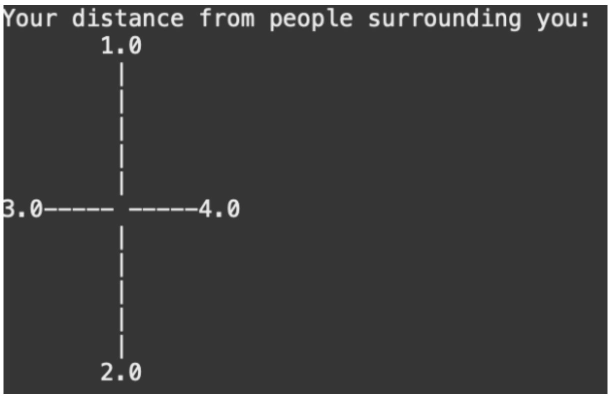
At the bottom part of the RestrictedSpots class code, there are few public methods and they are used as getter for the private data variables declared above, so they can be accessed by other classes in a controlled manner.

StaticDistancing.java class
In StaticDistancing class, one of the main features/functions of this program will be implemented here where current human capacity of each spot will be checked with the maximum capacity allowed before allowing the user to enter. Inside the class, there is a static void method called main which takes in three arguments, current capacity(capa), maximum capacity(maxcap) and average waiting time (waittime) from other class.

After the user selects which room to enter in the Main class (the Main class will be discussed later), it will compare and check the current capacity with the maximum capacity of the room, user is allowed to enter if current capacity is less than maximum capacity and the current capacity of that spot will increase by one. Else if the spot has reached its maximum capacity, it will display the average waiting time and user will be asked whether if they want to wait, if they choose to wait, they will be allowed to enter after the average waiting time for that spot. Else if they choose to not wait, user will be asked if they want to enter another restricted spot in the hospital, they will be redirected back to the main menu (Main class) if their answer is yes and the system will exit if their answer is no.

In this StaticDistancing class, if..else statements were mainly used to make and achieve these conditional statements.

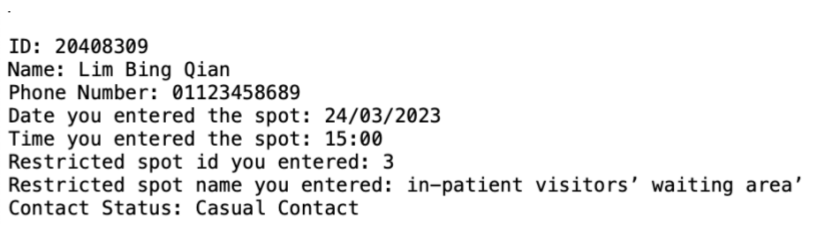
DynamicDistancing.java class
Once user has successfully entered a spot, they will be asked to input their physical distance (in metres) from other people in immediate surrounding in four different directions. An extraordinary feature is applied here, where it prints out and shows user the distances they've inputted in visualised form using the symbols "-" and "|".



```
Your distance from people surrounding you:
       1.0
        |
        |
        |
        |
        |
3.0----- -----4.0
        |
        |
        |
        |
        |
       2.0
```

Then, the system will check if distances of the user from other people in immediate surrounding in four different directions are safe, one by one with if conditional statements. If any one of the direction's safe distancing rule (at least 1 meter) is violated, user will be advised to move to the opposite direction for a calculated amount of safe distance. For example, if the distance between the user and the person on the left of user is 0.4 metre (violated safe distancing restrictions), the user will be advised to move 0.6 meters to the right to ensure safe distancing (1m-0.4m).

There is also an integer variable created earlier called "safevalue", it is initialised as 0 to track and check if the user has violated the safe distancing rules. If the user violates and is advised to move a safe distance away, then safevalue will increase by 1. If safevalue equals to 0, then they are classified as safe in dynamic distance. Else, the user will be classified as casual contact and asked to input their personal details to ensure they are safe in the next few significant days where symptoms of the virus will mostly likely develop and appear. The details required to be inputted are ID number, name, phone number, date and time they entered the spot. There is another extraordinary feature implemented here, where the details of casual contacts will be automatically written into a file for record tracking purpose.



```
ID: 20408309
Name: Lim Bing Qian
Phone Number: 01123458689
Date you entered the spot: 24/03/2023
Time you entered the spot: 15:00
Restricted spot id you entered: 3
Restricted spot name you entered: in-patient visitors' waiting area'
Contact Status: Casual Contact
```

At the end of the DynamicDistancing class, regardless whether they are classified safe or casual contact, user will be asked if they want to enter another spot or quit. User will be redirected back to the menu if they want to enter another spot in the hospital by inputting the number "1" and the system will exit the program if user wants to quit by inputting "2".

Main.java class
In the Main.java class where the program will be executed, the RestrictedSpots, StaticDistancing and DynamicDistancing classes are combined inside the static void main method. Firstly, the restricted spots are created by passing values (details of each spot) into the RestrictedSpots class to create the spots (objects), the restricted spots created are ICU, out-patient visitors' main waiting area, in-patient visitors' waiting area, medicine dispensary unit waiting area and administrative office.

Then, a menu of all the available restricted spots is displayed and it will prompt user to select the spot ID they would like to enter. Based on user's selection on which spot they would like to enter, a switch case statement is used to execute instructions for each different spots' case. For all different spots in the switch case statement, what's inside each case is more or less the same, it will calculate the maximum capacity of the spot, display the current number of visitors inside the spot, StaticDistancing and DynamicDistancing classes are also included and combined here in each of the case (each spot).The only differences between each case is that the values(data) of each spot are different. Lastly in the switch case (default), If the spot ID the user inputted is an invalid option, it will redirect back to the menu and require the user to enter again.

Finally, out of the static void main method, the public static void main method will be used to execute the main method.

Overall Experience

In the process of coding the human congestion management and crowd control system program, it was a smooth process in terms of figuring out the algorithm of this program as similar programs like this one were also learned and done in our previous C programming modules. However, because it is my first time learning and coding java, there were two technical difficulties I faced at the start of the coding, and I'll explain them here in this section one by one.

Firstly, it was with the constructor, as I wasn't aware that we need to the initialize objects using a constructor first before we can use them in Java Programming. After more readings about OOP (object-oriented programming) done on online resources, I learned that constructor is one of the essentials in doing OOP, to ensure that objects of a class are initialized properly and behave correctly.

Then, it was with the private variables in the RestrictedSpots class, I had a problem accessing to the variables in other classes because initially I thought that a constructor declared as public will already make the variables accessible in other classes. At first, I did not realize the problem was with the private variables, so I kept changing my code in the Main.java class on the restricted spots creation. This problem stopped me from continuing my program until I consulted and discussed with my course mate who used private variables in his class as well, he figured out and told me that it was the problem with the variables not being accessible in other classes. Then I started looking up online and thanks to this article written by Programiz: https://www.programiz.com/java-programming/examples/access-private-members , I was able to resolve this issue by returning the variables as public one by one.

Conclusion

In conclusion, this human congestion management and crowd control system was designed to ensure that every spot in the hospital is not overcrowded by people, taking into consideration of the COVID-19 pandemic restrictions and guidelines. This system does not only apply to a hospital setting, but it can also be used in other different location settings to minimize the chances of the virus spreading such as government office building, vaccination center and school etc. Besides that, I also managed to achieve the main aims of this system:

checking if the spot has reached its maximum capacity before allowing the user to enter and checking the user's distance with the people standing at the front, back, left and right of them to determine their risk status.

This program was also structured with the concept of object-oriented programming (OOP) and divided into three classes, RestrictedSpots.java, StaticDistancing.java and DynamicDistancing.java, each with its own set of unique features and functionalities. Then, the program also combines these three classes and executes it in Main.java.

During the development of the system, even though several technical difficulties were faced, but I am glad that I was able to overcome them and started adapting to the style of Java Programming better as I moved on further in developing the system. The ways I find solutions to the problems I faced were mainly by discussing with my course mates and finding suitable solutions on the internet.

As for the system's performance, it was tested to be efficient and effective in managing human congestion and crowd control in a hospital setting by minimising the risks of the virus spreading. However, it also has its limitations, such as its inability to detect and manage high-risk individuals who may be carriers of the virus. I will also try to redevelop and improve this system in the future to provide more features and functionalities to meet the evolving needs of hospital and other institutions, even though this system would already be submitted for marking by the time I am free to implement extra features.

References

- Programiz. (n.d.). Access private members in Java. Programiz. https://www.programiz.com/java-programming/examples/access-private-members

- W3Schools. (n.d.). Java Constructors. W3Schools. https://www.w3schools.com/java/java_constructors.asp

- Anna Monus. 2023, February 3. Using OOP objects to write high-performance Java code (2023). https://raygun.com/blog/oop-concepts-java/

- W3Schools. (2021, August 23). Java OOP (Object Oriented Programming). W3Schools. https://www.w3schools.com/java/java_oop.asp

- Trung Tran. (2021, December 17). Top 6 Object-Oriented Programming Languages. https://www.orientsoftware.com/blog/list-of-object-oriented-programming-languages/