Individual Work for all: Java Program, Report and Sample Report

Report for Java Coursework

Name: Yeo Yi Xin

Student ID: 20414841

1. Abstract

For this Java Individual Coursework, students were required to program a Congestion Management and Crowd Control System for a fictitious hospital in Malaysia. The purpose of the coursework was to manage the number of visitors currently at different restricted spots in the hospital. There are four spots which are included in this program to control the number of visitors going in, and making sure the visitors were always in a safe distance (1 metre) between one another. This system is created to lower the risk of infecting Covid-19 as well as to prevent overcrowding in the area in the hospital such as the Intensive Care Unit (ICU).

Using Visual Studio Code, the program was implemented with its main in display.java. The information about the restricted spots is done in an external inheritance where RestrictedSpots.java is the superclass to four subclasses, each representing one location in the hospital and store information about the spot such as the number of visitors allowed, allocated time for each visitor and so on. The program output and the input that the program got from the user will be printed and prompted from the terminal.

For this coursework, it is assumed that any input information and the current details is stored in the database that is already available in the hospital database system. This means that visitor information is inserted into the already available database system. [change?]

All the sizes of the restricted spots and maximum capacity of visitors a restricted spot can hold is also estimated. There are four restricted spots, which are Intensive Care Unit Visiting Area are four, and ten for Out-Patient Visitors' Main Waiting Area, Out-Patient Visitors' Sub-Waiting Area, and In-Patient Visitors' Waiting Area. The Intensive Care Unit Visiting Area (ICU) is assumed to be $40m^2$ wide while the other three restricted spots which are Out-Patient Visitors' Main Waiting Area, Out-Patient Visitors' Sub-Waiting Area, and In-Patient Visitors' Waiting Area are assumed to be $80m^2$ wide. The safe social distance is assumed to be 1m from all side. The maximum capacity for Intensive Care Unit Visiting Area is four, while the other three restricted spots are ten. This means that the chances of visitors in close distance (less than 1.0m) with one another is very low because the space is relatively larger compared to the maximum number of visitors allowed. As such, the contact status for this program is based on the distance of the user from other people because it is more effective. For example, if the visitor, A, is close to another person, B, the chances of the visitor A bumping into another person, C, after moving away from B in the same spot is highly unlikely to occur. The current number of visitors is generated by the Random Number Generator (RNG) using the random class. It is also assumed that even if the number of visitors is 0, there can be people in the restricted spot that is close to the user because that person can be a patient, hospital staffs such as nurses, doctors, and cleaners. User can opt to reapply the same spot if they were not allowed entrance to the spot because the program assume that there is a possibility that someone might have walked out when user reapply for same restricted spot because all visitors have allocated time that they are allowed to stay in the spot and the estimated waiting time for the user is just the worst case if the maximum number of visitors just entered the spot.

2. Brief Program Description

The main goal of the program is to control the number of visitors in each restricted spot and obtaining the contact status of the user in the spot. The current number of visitors in each restricted spot are generated by RNG which will randomly select a number between 0 and the maximum capacity that the spot can hold. The maximum visitor capacity is based on the size of the restricted spot and other factors such as the estimated number of hospital staffs in the spot. For example, the area in Intensive Care Unit was assumed to be $40m^2$ wide and only four visitors are allowed at a time as there are high number of hospital staffs. Low visitor number allow easy access for the hospital staffs like nurses and doctors to move around during emergency. The allowed time for all four restricted spots is 30 minutes. All the information regarding the place can be retrieved via the RestrictedSpots class, which is the superclass for the four subclasses. Decision-making activities such as giving permission to user on entering the restricted spot, setting allocated time user when user is allowed to enter, displaying the estimated waiting time, estimated time user will enter and allocated time can be in the restricted spot is done when they are not allowed entrance initially is done StaticDistancing class. From display class, the program can call StaticDistancing class to call DynamicDistancing class to check the distance of user from other visitors to determine the contact status of the user. This is because one of the methods in StaticDistancing called the socialdistancing() method contains function call to checkdistance() method in DynamicDistancing class which will determine the contact status of the user and whether the user needs to move away from another visitor.

When the details are all filled in and the contact status is confirmed, then the program will print out the current details of the visitor which includes personal information like user's identity number, name, location the spot's number, restricted spot's name, user's allocated time and contact status which are in different colour depending on whether it is DISTANCING, CASUAL or CLOSE. The program also takes into consideration of when user is not allowed into the restricted spots. If user is not allowed entrance to the restricted spot, the program will display the estimated time until user is allowed entrance, prompt user on whether user is willing to wait. If user is willing to wait, then program will assume the estimated time has reached and user is now allowed to enter. User will then be prompted for distance from the socialdistancing() method in StaticDistancing and the process then will be the same which are checking the contact status and then print out the current details. For both instances, after the current detail is printed, the program will prompt whether user wants to apply to the other spots. If user is unwilling to wait and not interested in applying for other restricted spots, then the program will end.

In this program, user will be required to enter the distance twice when the contact status is CASUAL or CLOSE to finalise the contact status of the user. In this program, there are a few restricted requirements that will cause the program to print out error message if the input in invalid which are discussed in additional feature section and can be seen in the sample. It is assumed that the guard will check the contact status of visitor from time-to-time to look out for any high-risk visitor. In this case, visitor with close contact will be considered high risk and may be asked to exit the restricted spot as it is unsafe for both the patients and hospital staffs. Visitors' contact statuses allow hospital management staff to check back on the contact

status of the allowed visitors if there is any disease spreading in the hospital. With this, the hospital can check the visitor's information with decreasing level of high-risk contact status of visitors, which are from CLOSE, CASUAL to DISTANCING. With the list, the hospital management staff can prevent visitor from spreading any viral illness to other places after visiting the hospital.

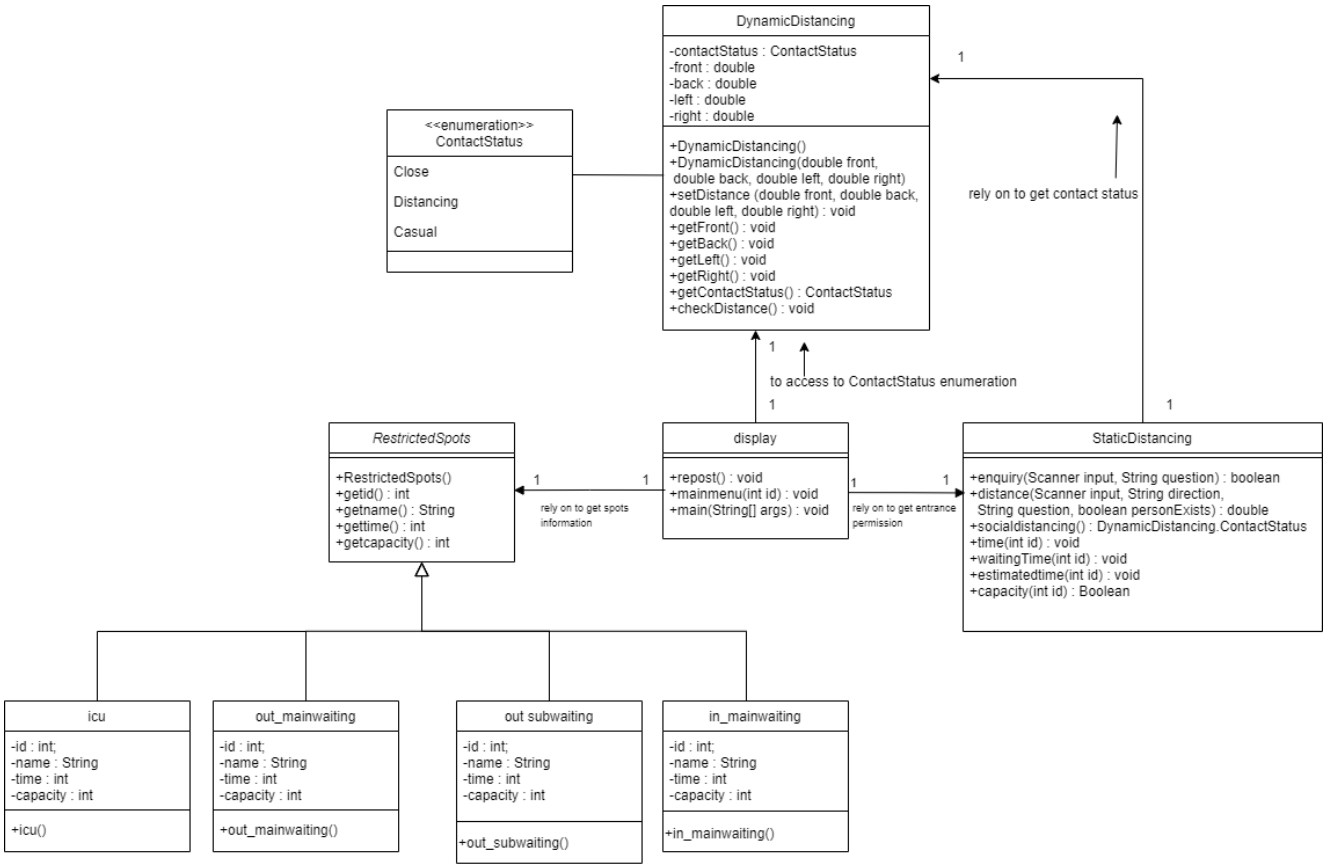
Hence, the Congestion Management and Crowd Control System for a fictitious hospital in Malaysia that is programmed and run in display.java aim to assure that the number of visitors in each restricted spot is not more than the maximum capacity and making sure the visitor is well-aware of their contact status and estimated time that the user is allowed to be in the spot. The opinion is also to make sure there is very low risk of having widespread of disease among patients, hospital staffs and visitors in the hospital due to allowing an unrestricted number of people into every spot in the hospital.

3. Structure Chart

The class diagram below shows the relationship between the eight classes. The main class that is used to run the program is in display.java which is associated to all the other classes. Display class main purposes are to display the menu and user current details, prompt for user input and store the input into a variable before passing it to other methods of the same class or different class as arguments in function call, check user input, display error messages and exit the program. StaticDistancing is to generate current number of visitor and check whether the user is allowed to enter to the restricted spot, current number of visitors after user is allowed to enter spot, allocated time for user in the spot using current time, estimated waiting time for user if user is not allowed to enter the restricted spots, estimated time until user's turn to enter spot, prompt user to enter whether there is a person in the direction, check the validity of user's input, prompt for distance between user and the person and store the distance input into different variables for each distance before passing as arguments to DynamicDistancing to obtain the contact status. Once contact status is obtained, it will return the contact status to main method in display.java. Since the contact status is stored in an enumeration, then the contact status has is initialised to be DynamicDistancing.ContactStatus, where ContactStatus is the name of the enumeration that stores the three-contact status which are CLOSE, CASUAL and DISTANCING. DynamicDistancing main purpose is to get the distance obtained from user input in StaticDistancing, check whether user must move in a certain direction whenever the distance entered is less than 1.0m, determine the contact status and return the contact status to the code that call the function. RestrictedSpots is an abstract class whose main purpose is to allow overriding of the information from and return the information to function call in main method of display.java.

The main purpose while coding out the program is to call minimal new instances, so it is more readable and easier to refer to when looking back at the program. Most of the methods called in main method in display class are from StaticDistancing class. The information for restricted spots is obtained by calling the abstract class RestrictedSpots that can access and return the override information of its four subclasses, which are, icu, out_mainwaiting,

out_subwaiting and in_mainwaiting which extends RestrictedSpots, because the abstract class can access the id, name, time, and capacity of any of the four subclasses through getter method. This shows that RestrictedSpots class is a superclass of the four subclasses. All the subclasses must have the same number and name of getter method as the RestrictedSpots class so that inheritance does not produce runtime error when retrieving information. For example, the getter method, getid() in RestrictedSpots is an abstract public that is empty but getid() in any subclass such as icu class contains value to override the information for getid() and returns the ID of the location. To get the contact status of the user, the function call to the method in DynamicDistancing check and return the contact status after determining the number of people who are close to the user and send warning messages in yellow highlights to ask user to move away (if required). If all distance is more than or equal to 1.0m, then contact status returns as DISTANCING in green. If there is one person less than 1.0m, then contact status returns as CASUAL in orange and if there are more than one less than 1.0m, then contact status returns as CLOSE in red. StaticDistancing returns the contact status to main method in display class after receiving the contact status from the getter method, which is getContactStatus() in DynamicDistancing.



4. Implementation and Description of Each Methods for each Classes in the Program

There are in eight classes involved in executing the Congestion Management and Crowd Control System, where four of the eight classes are subclasses to one superclass as observed in the class diagram. The class are listed as the following:

a) display

```
import java.util.Scanner;
import java.lang.NumberFormatException;
```

The public static main class where the program runs. It consists of two Java libraries which are java.util.Scanner and java.lang.NumberFormatException that is imported into the program using import declaration. Import statement is used to bring in classes or packages that are required for a particular program to run. The Scanner class in java.util package is used to prompt for user input. NumberFormatException class imported from java.lang package is used in all of the try-catch block to check if the input consists of numbers only. The NumberFormatException class will be discussed in more details in the main method of display class.

```
private static int spotsid = 0;
```

The program initialise a variable named spotsid of int datatype that store spots identification number (spot ID). It is a static method which means the value will not change throughout when the program runs, and it has a private access modifier which means this method is not visible to all other classes.

i. repost

```
public static void repost(int id, String name, int spots, String
spot_name)
```

The method, repost, receive four arguments which are of int datatype to receive user's identification number (user ID) and spot ID, String datatype to receive user's name, and String datatype to receive the name of the spots that user has chosen based on ID. All the arguments are received from main method. This method returns nothing, which is void. It is a static method which means the value in the method will not change when the program runs. It is of public access modifier which means this method is visible to all the other classes and methods in the program. This method is used to print out the current details of the user.

```
System.out.println("\n\033[1mYour current detail\033[0m");
System.out.println("-----------");
```

The first line of this method prints the String "Your current detail" in bold to catch user's attention to read their current details through System.out.println(). It is followed by printing dashed line.

```
System.out.println("ID: " + id);
```

```
System.out.println("Name: " + name);
System.out.println("Spot: " + spots);
System.out.println("Name of spot: " + spot_name);
```

After this line, it is followed by printing user ID, name, and spot ID that is based on the ID user has entered. Additional information like the spot's name is also printed. In the current details, there is also the allocated time based on current time and contact status, but these details are not printed from this method.

ii. mainmenu

```
public static void mainmenu(int id)
```

The method, mainmenu, receive one parameter which is of int datatype to receive the spot ID. The function is called, and argument is passed from main method. This method returns nothing so return type is void and the method is static.

```
System.out.printf("\n%2s %22s %50s %19s\n", "ID", "Accessible
places", "Permitted waiting time (min)", "Maximum capacity");
```

The method began with printing the table header, which is the attribute for each column, mainly restricted spot's identity number, the name of the spots, permitted waiting time or known as the allocated time for each spot and the maximum capacity for each spot using System.out.printf() to format the output.

```
if (id!=1){
        RestrictedSpots first = new icu();
        System.out.println(first.getid() + "\t" + first.getname()
+ "\t" + first.gettime() + "\t\t\t\t" + first.getcapacity());}
```

The method contains four if statements. One of the if statement is shown above which prints information for ICU. Four if-statement is used instead of one if-else if-else statement because the program needs to check whether all the condition is true or false instead of just checking one condition and escape from the if-else if-else statement after one of the conditions is fulfilled. The condition used in each of the if-statement is to check inequality between spot ID stored in variable id and the number. The system will execute the codes if the inequality is true. This is an additional feature that is used when the user checks other spots after obtaining permission to enter the first spot or when user is not allowed to enter the spot. The first line of code is a constructor which is a form of implicit inheritance that creates a new instance of the spot using new keyword and assigned to a variable named first of type RestrictedSpots. Then, it prints the details of the spot, which are spots ID, name, allocated time, and maximum capacity, using System.out.println(). Between each variable calling the method in RestrictedSpots using dot operator, there is different number of escape sequence to insert tab so everything will be aligned. In the println of fourth if-statement, there is an escape sequence to insert a new line.

iii. main

```
public static void main(String[] args)
```

The method, main, receive one parameter which is String argument. This method returns nothing, so it is void, it is a static method and has public access modifier.

```
Scanner input = new Scanner(System.in);
mainmenu(spotsid);
```

It starts with creating a new instance of Scanner class from java.util package and assigned it to a variable named input which is used to prompt user input. It is followed by function call calling the mainmenu method which pass the argument spotsid whose value is still 0 to print the menu. Hence, the program will print every line in the mainmenu method.

```
while (true)
```

The rest of the program is enclosed in an infinite while loop that remains true as its loop-continuation condition. The program will break out of the while loop only after encountering the break keyword, which is when user does not want to apply to other location.

```
System.out.println("\033[1mPut in your details\nUser
input\033[0m");
System.out.println("----------");
int userid;
int repeat = spotsid;
```

The program starts with printing two lines which indicate that the following section require user to type in their details through System.out.println(). The first string is printed in bold. Then, a variable named userid of int datatype is initialised. Another variable named repeat is initialised with int datatype and assigned with value stored in spotsid using an assignment operator (‘=’).

```
while (true) {
    System.out.print("Your ID: ");
    String userID = input.nextLine();
    try {
         userid = Integer.parseInt(userID);
         break;
    } catch (NumberFormatException e) {
         System.out.println("\033[91;1mPlease enter number only\n \033[0m");
    }
}
```

This nested while loop structure is used to prompt user ID but will also be used to prompt for spot ID as well. The difference is only the variable name in which the user input is stored into. This nested while loop will only break if the user enters valid input of integer datatype. The first line of code in the nested while loop is the

System.out.print that prompts for user ID through input.nextLine() which will read user input until the program encounters new line character (‘\n’). The result of call to nextLine is given to variable userID using assignment operator (‘=’). Inside the try block, the program will convert user input from String datatype to Integer datatype using Integer.parseInt() that will pass userID as argument. The result to Integer.parseInt() is given to variable userid using assignment operator (‘=’). If the input is invalid, which means the result after conversion does not match integer datatype, then the NumberFormatException is thrown and caught by the catch block. The program will execute the System.out.println and display the error message that in red and bold informing user to enter only numbers. The program will run the loop again until the input is valid. If user input is valid, then the program will encounter the break keyword in the try block and escape from the nested while loop.

The program continues by prompting for user’s name using System.out.print through input.nextLine() in which the result of call to nextLine is given to variable name.

The program is followed by prompting spot ID and when the value stored in spotsid is valid, the program checks spotsid for the condition in while loop to determine whether integer entered is between 1 and 4.

```
while ((spotsid < 1) || (spotsid > 4) || (repeat==spotsid)){
   if ((spotsid < 1) || (spotsid > 4)){
       System.out.println("\033[91;1mPlease enter only the given location
id!\n\033[0m");
    }
   else{
       System.out.println("\033[91;1mYou have applied for this location
before\n\033[0m");
}
    while (true) {
        System.out.println("Location ID:");
        String locationID = input.nextLine();
         try {
              spotsid = Integer.parseInt(locationID);
              break;
         } catch (NumberFormatException e){
               System.out.println("\033[91;1mPlease enter number only\n\033[0m");
         }
    }
}
```

The loop-continuation condition in this while loop is comparison on whether value in spotsid is less than 1, more than 4 or equal to the value stored in repeat. If any of the condition returns true, the nested while loop will run. In the while loop, the program checks the expression in if-else statement where codes in if-statement will be executed if spotsid is less than 1 or more than 4. If true, System.out.println is exected and program display the error message in red and bold informing user to enter only the given spot ID. Codes in else statement will be executed if user

input is equal to the value stored in spotsid and an error message in red and bold informing user on repeated request. The nested while loop within the current nested while loop will prompt for spot ID until user enters a valid input which is like the nested while loop described when program first started prompting for user ID. After escaping from the nested while loop which is in a nested while loop, the program will check whether it is an integer between 1 and 4. If the integer entered is between 1 and 4, then the system will break out of the while loop. If the integer is not between 1 and 4, then the while loop will run until the input is between 1 and 4.
The program has finished prompting for user input.

```
StaticDistancing permission = new StaticDistancing();
boolean entrance = permission.capacity(spotsid);
```

The above is a constructor which creates a new instance of StaticDistancing class, which is assigned to a variable named permission of type StaticDistancing using new keyword. The next line of code shows function call for capacity which is in StaticDistancing class using permission and a dot operator. The function call passes spotsid as an argument. The return value of the function is stored in variable entrance which is of boolean datatype that only store true or false. The capacity() method is called to check the current capacity based on the spotsid. Based on the value stored in entrance, the program will determine whether user is allowed to enter the restricted spot.

```
DynamicDistancing.ContactStatus contact = permission.socialdistancing();
```

If user is allowed to enter, the program continues by calling the function socialdistancing() from StaticDistancing class to check for contact status from other class and store it in variable contact of datatype enumeration of ContactStatus from DynamicDistancing class.

```
if (spotsid == 1){
     RestrictedSpots first = new icu();
     repost(userid, name, spotsid, first.getname());
}
```

It is continued with if-else if-else statement which expression is checking equality between spotsid with 1, 2 and then 3. The equality between spotsid and 4 is checked in the else statement.
It begins with the constructor calling a new instance of the icu class, which is assigned to a variable named first of type RestrictedSpots. All the three variables which stores user input are pass as arguments into the function call for repost() method. The fourth argument is the name of the chosen spot that is obtained by calling getter method, getname() from RestrictedSpots which obtains the override information from icu. The block of code is the same for the other 3 spots where the difference is the instantiation.

```
permission.time(spotsid);
```

It is followed by function call for time() method from StaticDistancing class that passes spotsid an argument. This function call will print out the allocated time

(based on current time) for visitor after they are allowed entrance to the restricted spot.

```
if (contact == DynamicDistancing.ContactStatus.CLOSE || contact == DynamicDistancing.ContactStatus.CASUAL)
```

The if-else statement checks for equality between user contact status and constant value in enumeration, that are CASUAL, CLOSE or DISTANCING (else).

```
if (contact == DynamicDistancing.ContactStatus.CLOSE
            System.out.println("\033[0;31m");
            System.out.println("Contact Status: " + contact);
            System.out.println("\033[0m");
}
else{
            System.out.println("\u001B[38;5;208m");
            System.out.println("Contact Status: " + contact);
            System.out.println("\033[0m");
}
```

In the if statement, there is a nested if-else statement that checks whether the contact status is CASUAL or CLOSE. If it is CASUAL, then it will print the contact status in orange and red if contact status is CLOSE and green for DISTANCING.

```
System.out.println("Put 'Y' or 'y' as yes and other keys as no.");
System.out.print("\nDid you move away from the person(s) who are close to you? ");
char choice = input.next().charAt(0);
input.nextLine();
```

An additional feature is added in which user is prompted on whether user has moved away from the person(s) if contact status is CLOSE or CASUAL to finalise the contact status of user. User input is stored in variable choice with character datatype. The nextLine() is used because the input.next().charAt(0) will only read the character only and not the new line character. Hence, if there is an absence of input.nextLine(), the next user input may take new line character as input. If user answers ‘y’ or ‘Y’, codes in if statement will be executed and prompt user’s distance by calling socialdistancing() method in StaticDistancing before proceeding with printing current details together with the finalise user’s contact status.
If user did not move away, then the code in else statement is executed. The program prints warning message that they may be guided out from the spot for not being within the safe distance protocol (1.0m). After that, the program continues with function call for socialdistancing() method in StaticDistancing and prints user current details.

```
if (choice == 'Y' || choice == 'y' ){
   System.out.println("");
   mainmenu(spotsid);
```

```
    continue;
}
else{
    System.out.println("Thank you!");
    break;
}
```

The program prompts user to continue applying to other restricted spots. The input is stored in variable choice. Variable choice is used in if-else statement to check if user enters 'y' or 'Y'. If true, then program will print a new line with System.out.prinln and call the function, mainmenu() and pass spotsid as argument so that the menu will not print the row spot's information that has just been applied by user. If user enters another key, program assumes that user does not want to continue anymore and will print "Thank you" and break out of the while loop.

If user is denied permission to enter the spot as it reaches maximum number of visitors allowed, the program will execute codes in else.

```
permission.waitingTime(spotsid);
```

The function call to waitingTime() in StaticDistancing is called. The function call will pass spotsid as an argument. This method will display the waiting time for user according to worst case (maximum waiting time). Then, the estimated time until user is allowed to enter is displayed. The program will prompt user to determine the willingness to wait until the time. User input is stored in variable wait. There is nested if-else statement with condition on whether user to wait. If user is willing to wait ('y' or 'Y' is entered), the program will execute the codes in if statement. The program will assume time is up and user is granted permission to enter. It will proceed in executing the block of codes to print user's current details. The only difference is that the allocated time is now based on the worst case where the program assume that user has waiting for the maximum waiting. After printing current details, program will prompt whether user wants to apply to other restricted spots. 'y' or 'Y' key will print the menu and continue the while loop and other keys will break out from the while loop.
If user is not willing to wait (enter other keys), the program will execute the code in else statement. It will prompt whether user wants to apply to another restricted spot and act accordingly based on user input.

```
System.out.printf("\n\033[4m\nDo you want to try to re-apply to
the same location again?\n\033[24m\n");
```

System.out.printf formats the output is executed and print in underline. The program will prompt whether user wants to apply the same venue and the input will be stored in variable choice1. The nested if-else statement in which code in if-statement will be executed if user enters 'y' or 'Y' while else statement will be executed if user enter other keys. When code in if-statement is executed, spotsid will store 0 as its new value. The program then executes the function call mainmenu() and pass the spotsid as argument. The program encounters the continue keyword and the while loop rerun again. If user decided not to reapply to the same location, the program will execute the System.out.println that prompt

user on whether user wants to see only the restricted spot ID that has not been applied yet. If user enters 'y' or 'Y', program will execute the if-statement and print the System.out.println where the program display "Available spots: " in bold. mainmenu() method is called and the row in which the restricted spot ID is not the same as the value stored in spotsid is printed and the program runs while loop after encountering the continue keyword. Otherwise, program will call mainmenu() method and print the complete menu which contain restricted spots information from 1 to 4 and then encounter the continue keyword. If user does not want to apply to another location, the program will display "Thank you!" and then break out of while loop after encountering break keyword.

```
if (input!=null){
     input.close();
}
```

This if statement is outside the while loop and checks whether the Scanner object, input is not null and, if it is not null, it will call close() method to release any system resources associated with it.

b) StaticDistancing.java

```
import java.util.*;
import java.text.SimpleDateFormat;
```

This program consists of two Java libraries which are java.util.* and java.text.SimpleDateFormat that is imported into the program using import declaration. Java.util.* is used to import all the classes within java.util package. This package contains various utility classes such as the Scanner class for reading input from the user, Date class for representing dates, and Calendar class for working with dates and times. By using the wildcard *, all the classes within the java.util package are imported, so it does not have to specify each class individually. On the other hand, java.text.SimpleDateFormat is used to import the SimpleDateFormat class from the java.text package. This class provides methods for formatting and parsing dates according to a specified pattern. By importing this class, the program can create instances of the SimpleDateFormat class and use its methods to format and parse dates in the program. The program starts with declaring the name of the class called StaticDistancing which is of public class.

i. enquiry

```
public static boolean enquiry(Scanner input, String question)
```

The method has a public access modifier, static, accepts user input through Scanner class and accepts variable called question with String datatype. It returns value of Boolean datatype to the line of code where the function is called in socialdistancing() method.

```
do {
    System.out.print(question + " (y/n): ");
    char response = input.next().charAt(0);
    input.nextLine();
```

```
    if (response == 'y' || response == 'Y') {
        return true;
     } else if (response == 'n' || response == 'N') \
        return false;
     } else {
        System.out.println("\033[91;1mInvalid response. Please
enter y or n.\n\033[0m");
      }
}while (true);
```

Inside the method is a do-while loop where program will print out the String which is stored in variable question and then, prompt user to enter either 'y', 'Y', 'n' or 'N'. The program executes codes in if statement and returns true when user enters 'y' or 'Y' while else if statement when user enters 'n' or 'N' which will return false. The restriction in the program is that the loop will continue to run if user enter another key other than 'y' or 'n' key. Program will print out the error message in red and bold every time user enters an invalid input.

ii. distance

```
private static double distance(Scanner input, String direction,
String question, boolean personExists)
```

The method has a public access modifier, static, accepts user input through Scanner class, variable called direction and question with String datatype, and variable personExists of boolean datatype. The method returns a value of double datatype to the line of code where the function is called.

```
 if (!personExists) {
   System.out.println("No person detected to the " + direction + ".");
   return 1.0;
}
```

This if-statement will only be executed if variable personExists is false. It will return the distance as 1.0 which complies to the safe social distance so the contact status will not be affected. If personExists stores a true value, then it will execute the while loop after this if-statement to prompt for distance of user and the person in the direction, then store value in variable direction of String datatype.

```
try {
     distance = Double.parseDouble(checkdistance);
     break;
} catch (NumberFormatException e) {
     System.out.println("\033[91;1mPlease enter number only\n\033[0m");

}
```

The try-catch block will check whether user input is of type double or not. If not, then the NumberFormatException is thrown and caught by the catch block, and it will print an error message in red and bold font. The program will continue to prompt for user input until the input is a valid input. If the input is valid, then the try block sets the check distance variable and exits the loop using the break keyword.

```
return distance >= 0.0 ? distance : 1.0;
```

This conditional operator is used to determine the distance value that should be returned by the method. If distance is more than or equal to 0, the condition is true, and the value of distance is returned. If condition is false, then value 1.0 is returned instead as negative value will not be considered.

iii. socialdistancing

```
public DynamicDistancing.ContactStatus socialdistancing()
```

The method has a public access modifier, static, does not receive any argument but returns an object of the method of the ContactStatus enumeration that is defined within the DynamicDistancing class. The method starts with creating a new instance of Scanner class from java.util library, and then assigned to variable input which is used to prompt user input. It then prints the System.out.println to print the string which consist of newline character and string which is printed in bold.

```
boolean hasFront = enquiry(input, "Is there a person in front of you?");
double front = distance(input, "front", "What is the distance between
you and the person in front", hasFront);
```

The above two line of codes shows function call for enquiry method which has two arguments which are prompting for user input, and String containing the question on whether there is a person in the direction of the user. The return Boolean value of enquiry method is stored in variable hasFront of Boolean datatype. The variable hasFront becomes one of the arguments for function call distance which includes three other arguments which are prompting for user input, String which indicate the direction and the question on the distance between the user and the person which is in the direction. The return value is stored in variable front of double datatype. These two lines of code will be repeated for another three times for enquiry and distance in the direction of back, left, and right. The difference are just the variable names the input is stored into and the details of direction of the String in the parameter of function call.

```
DynamicDistancing distance = new DynamicDistancing(front, back, left,
right);
distance.checkDistance();
DynamicDistancing.ContactStatus contactStatus = distance.getContactStatus();
return contactStatus;
```

The program continues with a constructor which creates a new instance of DynamicDistancing class, which is assigned to variable distance of type DynamicDistancing using new keyword that pass four arguments which are the distances obtained from user input in double datatype. The function call for checkDistance() checks whether user is in safe distance from the person and will determine the contact status of the person based on the number of person who are close (less than 1.0m) to the user. The contact status is obtained through function call for getter method in DynamicDistancing which is getContactStatus(). The return type is stored in variable contactStatus of ContactStatus datatype from DynamicDistancing class. The variable contactStatus is returned to its function call in the main method of display.java.

iv. time

```
public void time(int id)
```

The method is of public access modifier and has a parameter, variable id of int datatype. This method returns nothing, and hence, it is a void.

```
Calendar calendar = Calendar.getInstance();
SimpleDateFormat time = new SimpleDateFormat("HH:mm:ss");
Date current_time = new Date();
SimpleDateFormat date = new SimpleDateFormat("dd/MM/yyyy");
Date current_date = new Date();
int place = id;
```

The program creates a Calendar() instance using the Calendar.getInstance() method. Calendar() class is used to perform operations such as getting current date and time, adding, or subtracting dates and times, and formatting dates and times. It is followed by creating two SimpleDateFormat() objects, time, and date, using the specified date and time formats. SimpleDateFormat() class used to format and parse dates and times according to a specified pattern. The code continues with two variables, current_time and current_date, using Date() constructor. Date() class represents a specific instant in time, with millisecond precision.
The code is followed by a switch statement with the expression id that stores spot ID.

```
case 1: {
    RestrictedSpots place1 = new icu();
    calendar.add(Calendar.MINUTE, place1.gettime);
    Date due_time = calendar.getTime();
    System.out.println("Allowed Time: " + time.format(current_time) + " to " +
time.format(due_time));
    System.out.println("Date: " + date.format(current_date)); break;
}
```

This is an example for case 1 that will be executed if the value pass through function call is 1. The program starts with creating a new instance of the spot using new keyword, which is assigned to variable first of type RestrictedSpots. Using the function call for add method in the calendar that passes two argument, current time in minutes using Calendar.MINUTE field and getting the allocated time of the spot through getter method, gettime() in RestrictedSpot to access the time in the subclass according to the instance. The new value is obtained through function call of getter method, getTime() method where the return value is stored in due_time of Date() datatype. The program display the String output, current_time and due_time, through function call of format() datatype, where one receive current_time and another receive due_time. This line shows the time span user is allowed in the restricted spot. Then, the next line prints the current date stored in current_date through function call of format() that will pass the parameter current_date. After printing out the line of code, program will encounter a break keyword and escape from the switch statement. The block of codes for case 2, 3 and 4 are the same but with different instances. There is no default case in the switch statement because the validity of location ID has been checked by the while loop in main method in display.java.

v. waitingTime

The method is of public access modifier and has a parameter, variable id of int datatype. This method returns nothing, and hence, it is a void.
The program starts off like the time method but without initialising the date.

```
calendar.add(Calendar.MINUTE, 30);
Date due_time = calendar.getTime();
```

The difference from time method is the addition of 30 minutes in the current time in minutes field is done before the while loop. The value is assigned to due_time of Date() datatype. The program continues with a switch statement with expression, id, that contains the value pass by function call in the main method of display.java. Inside the switch statement, contains 4 cases, which are case 1, 2, 3 and 4. In all the case, the program will fetch information from RestrictedSpots the same way as discussed in time method. In this method, the program will only display the waiting time in worst case (maximum waiting time) through getter method which is the function call to gettime in RestrictedSpots where the gettime can access the waiting time for the subclass that was instantiated in the constructor. It will then print the estimated time until user's turn which is also based on estimated waiting time that is stored in due_time and pass as argument in function call format(). The program will break out of switch statement after encountering the break keyword. The same block of code is used for case 2, 3 and 4 where the only difference is the subclass used to instantiate the constructor.

vi. estimatedTime

The method is of public access modifier and has a parameter, variable id of int datatype. This method returns nothing, and hence, it is a void.
The program starts the same way as the waitingTime method but without the two lines of code which adds 30 minutes to current time.
The program has switch statement with expression, id, that contains the value pass by function call in main method of display.java. Inside the switch statement, contains 4 cases, which are case 1, 2, 3 and 4. Inside the case block contains the line of code to add 30 minutes in the current time and the value is stored in start_time using the getter method in Calendar class. Next line continues to add another 30 minutes (in total, 60 minutes has been added) and store the value in due_time of Date datatype through function call of getTime() method in Calendar class. The program then display the output of the String, start_time and due_time, through function call of format() datatype, where one receive start_time and another receive due_time. This line shows the time span user is allowed in the restricted spot. The same block of code is used for case 2, 3 and 4 where the only difference is the subclass used to instantiate the constructor.

vii. capacity

The method is of public access modifier and has a parameter, variable id of int datatype. This method returns a value of Boolean datatype, and hence, it is a Boolean.

```
Random rand = new Random();
Boolean permission = true;
```

The method starts with creating a new instance of the Random class and assigns to variable rand using new keyword. The variable permission of datatype Boolean is assigned to value, true. The program continues with switch statement with expression, id, that contains the value pass by function call in the main method of display.java.

```
case 1:{
   RestrictedSpots place = new icu();
   int currentcapacity = rand.nextInt(place.getcapacity());

   System.out.println("Maximum number of visitor allowed: " + place.getcapacity());

   System.out.println("Current number of visitor: " + currentcapacity);

   if (currentcapacity >= place.getcapacity()){
      System.out.println("\n\033[4;41mMaximum visitor capacity has reached\033[0m");
      permission = false;
   }else{
      int spaceleft = currentcapacity + 1;
      System.out.println("\n\033[44;4mYou may enter\033[0m");
      System.out.println("Current number of visitor (including you): " + spaceleft);
    } break;
}
```

The code in case 1 starts with creating new instance of the spot using new keyword, which is assigned to variable named first of type RestrictedSpots. The variable currentcapacity of int datatype is assigned with a random int between 0 and the max capacity set, in this case is 4, using the function call of nextInt() method with where this call receives a parameter. The parameter is a function call to get the capacity of the spots through getcapacity() in RestrictedSpots that will access the subclass that is instantiated in the constructor. The program continues with printing the capacity of the restricted spot through System.out.println by calling the getcapacity() method in the RestrictedSpots and print the current number of visitors through variable currentcapacity.

The program continues with if-else statement where code in if statement will be executed when the currentcapacity is more than or equal to the maximum number of visitors allowed in the spot (4, in the case of ICU). When the condition in if-statement is met, the program will print the statement in red highlight and underlined and set permission to false. If the condition is not met, else statement will be executed and a new variable spaceleft of int datatype will be assigned to the value from addition between currentcapacity with 1. The program will print out the statement that shows visitor is allowed to enter in blue highlight and underlined. Then, it will print variable spaceleft using System.out.println. The program will break out of switch statement once it encounters break keyword. The block of codes is the same for all the case 2, 3 and 4 where the differences are the subclass which is instantiated in the constructor and the maximum number of visitors allowed. After switch statement, the method returns permission back to the function call in main method of display.java.

c) DynamicDistancing

The class is a public class which means it is visible to all the other classes and methods.

```
public enum ContactStatus{
        CLOSE,
        DISTANCING,
        CASUAL
}
```

This is a public enumeration named ContactStatus which is declared using the enum keyword, followed by the enumeration name in curly braces. Inside the curly braces, three named constants are declared: CLOSE, DISTANCING, and CASUAL, each separated by a comma. These constants represent different levels of contact that is assigned to the users based on the number of person(s) close to the user at the four sides.

```
private ContactStatus contactStatus;
private double front;
private double back;
private double left;
private double right;
```

The above initialise contactStatus as ContactStatus which is an enumeration and front, back, left, and right as double datatype. All the variables are private class modifier which can only be accessed within the class where it is declared. It cannot be accessed by other classes.

```
public DynamicDistancing(){
        back = 0.0;
        left = 0.0;
        right = 0.0;
        contactStatus = ContactStatus.DISTANCING;
}
public DynamicDistancing(double front, double back, double left, double
right){
        this.front = front;
        this.back = back;
        this.left = left;
        this.right = right;
        contactStatus = ContactStatus.DISTANCING;
}
```

DynamicDistancing has two constructors: a default constructor that initialises all the instance variables to 0 and sets contactStatus to DISTANCING, and a parameterized constructor using this keyword that has four double arguments representing the distances in the directions respectively. It sets the distance variables to the provided values and sets contactStatus to DISTANCING.

```
public void setDistance (double front, double back, double left, double
right{
        this.front = front;
```

```
        this.back = back;
        this.left = left;
        this.right = right;
        checkDistance();
    }
```

The program has setter method called setDistance() which takes four double arguments representing the distances and has this keyword to differentiate with the default constructor and to set the distance variables to the provided values and then call the checkDistance() method to update the contactStatus.

```
public double getFront(){
        return front;
    }
```

It has getter method for all the directions and contact status, which are getFront(), getBack(), getLeft(), getRight() and getContactStatus() to allow external code to retrieve these double and enum values. Hence, the values of these variables can be accessed by other code without allowing direct modification of the values, which helps to ensure the integrity of the data in the object.

i. checkDistance

The method is a public access modifier with no datatype and no return type, which is null. It will first initialise j with int datatype with 0.

```
if (front >= 0 && front < 1.0){
            j += 1;
}
```

There are 3 other if statement which are similar where the difference is the variable name to calculate for each direction. The if statement will only be executed when the condition, value of front is larger than or equal to 0 and lesser than 1, is met. Then, the value stored in variable j will be incremented by 1. Variable j is used to calculate the number of people who are close to the user.

```
if (front >= 0 && front < 1.0){
            double move = 1.0 - front;
            System.out.printf("\n\033[43mPlease move %.2fm from the
person(s) at the front of you.\033[0m\n", move);
        }
```

There are 3 other if statement which are like this if statement where the difference is the variable name for each direction and the name in the string for the warning message. When the condition of the if statement is met, the program will initialise variable move with double datatype of value 1.0 subtracted by the distance. Then, there is a newline character, and the string will be printed out with yellow highlight and with formatted output System.out.printf which will print the value in variable move.

```
if (j>1){
  System.out.println("\nThere are people from more than one side
who are close to you.");
  System.out.println("\nYou are in a restricted area with close
contact risk.");
```

```
    contactStatus = ContactStatus.CLOSE;
}
else if (j == 1){
    System.out.println("One side");
    contactStatus = ContactStatus.CASUAL;
}
else{
    contactStatus = ContactStatus.DISTANCING;
}
```

The if-else if-else statement determine the user's contact status and store it in variable contactStatus. If more than one people are close to user, then the program will run the if statement and print out all the output in System.out.println which gives warning to user and set the contact status to CLOSE. If only one person is close to user, then the program runs the else if statement and print out one line which tells user one side is close to user. Then, set contact status to CASUAL. If no person is close to user, else statement is executed, and contact status is DISTANCING.

d) RestrictedSpots

```
public abstract class RestrictedSpots
```

The class is of public access modifier, and it is of abstract class which cannot be instantiated and cannot create an object directly from it. The class serves as a blueprint for the four subclasses, which are icu, out_mainwaiting, out_subwaiting and in_mainwaiting classes to extend it. Hence, RestrictedSpots serves as a superclass for the other classes, which is a form of external inheritance.

```
public RestrictedSpots(){
        ;
}
public abstract int getid();
public abstract String getname();
public abstract int gettime();
public abstract int getcapacity();
```

The default constructor RestrictedSpots has an empty body and does not take in any arguments. It is used to create objects of the class that extends the RestrictedSpots class. In this class, there are four abstract methods, which are getid(), gettime() and getcapacity() of int datatype, and getname() of String datatype. These methods do not have any implementation in the abstract class and are marked as abstract with abstract keyword. This means that any subclasses of RestrictedSpots must implement these methods and provide return of the information of the subclass in the methods. Hence, it allows for greater flexibility, as the four subclasses can provide different implementations of the abstract methods to give information of the restricted spots.

e) icu, out_mainwaiting, out_subwaiting and in_mainwaiting (all extends RestrictedSpots class) which is an example of overriding.

```
public class icu extends RestrictedSpots
```

All the four subclasses are similar where the difference is the information that each subclasses hold, which are the spot ID, name, and capacity that the restricted spot can hold. Hence, icu subclass will be taken as an example to explain the program. This subclass is of public access modifier and extends RestrictedSpots class, which is a form of external inheritance. It starts by initialising four private class variables which are id, time, and capacity of int datatype, and name of String datatype. The default constructor icu has the default value of the private variables that have been stated. Each of the variable initialised has a getter method which returns the default value of each variable that has been initialised in the default constructor and can be accessed by RestrictedSpots whenever the getter method in RestrictedSpots is called. Thus, the icu class is overriding the abstract methods declared in its parent class RestrictedSpots. It provides its own implementation of the getid(), getname(), gettime(), and getcapacity() methods that were declared abstract in RestrictedSpots. This is an example of method overriding in Java, where a subclass provides its own implementation of a method that is already defined in its superclass.

5. Additional Features

The program fulfils all the aspects that is needed in the coursework as well as having several extraordinary features where some are already explain above (in highlights). The features include implementing external inheritance such as RestrictedSpots class as superclass and the four spots as the subclass that extends the superclass. The method overriding is done in the inheritance by providing the spot ID, name, time, and capacity in the methods declared in the RestrictedSpots class.

Next, the program adds colours for error and warning messages, allowed, or not allowed entrance to the restricted spots, the contact status of the user, bolding the current details and at the beginning of the program before prompting for user input and using underline. All of these are done to make it nicer and easier to read. It also emphasises the seriousness when red is used in error messages, when user is not allowed entrance to the restricted spot and when the contact status is CLOSE.

This program also differentiates the contact status of DISTANCING, CASUAL and CLOSE through the number of people who are close to the user because the size of the restricted spot is big, and the maximum number of visitors are small so it is more feasible to determine contact status this way because it assumes that all the details of the visitor is recorded in the database and the hospital system can track back all the CLOSE contact visitors. The visitor will be asked to move in the direction that is opposite to the person that is close to the visitor. User who is in CASUAL or CLOSE contact will be asked to re-enter the presence and distance of the person in all the directions and then, finalise the contact status of the user.

In addition to that, the program includes implementing current time and printing out the time user is allowed in the restricted spot, the estimated time user will be allowed to enter the restricted spot if the user was not allowed entrance initially, and the time user is allowed to enter after allowed entrance again based on the assumptions that the time that user enter is the maximum time user has to wait before user is allowed entrance to the restricted spot.

Other additional features include restrictions like user ID, spot ID, and distance between user and another visitor can only consist of only numbers. Another restriction would be that user can only enter ‘y’, ‘Y’, ‘n’ or ‘N’ when being prompted on whether there is

someone in the direction of the user. Next, if user enter the same spot ID as the user did previously, the program will display an error message in red and bold and user will be prompted to enter the input.

6. Overall experience

The implementation of all the restrictions and additional features are a huge motivation for me because it felt exciting and proud to be able run the program according to how it was described was a proud moment for me. All these restrictions and additional features makes the program more complete because the database will store consistent, valid and useful information rather than inconsistent and wrong information. With this, analysis of visitor information such as the contact status as well as the peak hours for visitation are done with higher accuracy because the information are all valid. Hence, more implementations can be made after analysing the peak hours and tendency of visitor to stand close to other visitor so that the system be able to send out warning messages whenever the visitor is close to another visitor at real time.

The coursework opens an opportunity for me to try to implement internal and external inheritance because it is a challenging and, it is my first time learning the concept of inheritance. It allows me to understand better the relationship between a superclass and subclass. In this coursework, I can understand better the way to write method and the access modifier, which are either public or private class access modifier, and which one to choose when writing a method or when initialising a variable. From the program, private class access modifier is usually used when initialising the variables that will not be used in other classes, and public class access modifier when initialising a method that will be called from other methods from the same or different classes. Inheritance is used when there is a set of related classes that share some common function. Using abstract class in the program means that there the subclass is overriding the abstract method written in the methods of the superclass. This part of the code is challenging because I was not aware initially that all the subclasses must have all the abstract getter methods in the superclass. With all the errors the compiler listed, I tried to debug the errors and found the error which is the inconsistent name of the getter method in the subclass. It was a great experience to learn more about inheritance after the process of debugging.

7. Conclusion

In conclusion, the Congestion Management and Crowd Control System is an important system that is needed to prevent overcrowding in the hospital and keep the chances of spreading infectious disease among the patients and hospital staffs at bay. The program fulfils all the aspect that is needed in the coursework as well as having several extraordinary features that are discussed as above. As discussed, this program will be helpful to the hospital to keep the information of the visitors so that the hospital staff can track back the visitor list when there is a spread of disease among the patients and staff because it is highly possible that the spreader is a visitor. Other than that, it can control the number of visitors going in and out of the spots in the hospital to make sure there is enough space for each visitor to get into safety or safe themselves if there is any hazard happening in the hospital such as fire outbreak. Less number of visitors and controlled number of people in the hospital means low risk of stampede happening in times of emergency in the hospital. Thus, the program is very helpful in managing congestion and controlling the size of the crowd. To sum it all up, the program can display the information of the restricted spots, prompt for user input, gives permission for user to enter the restricted spot, determine the contact status of the user, print user details and then prompt whether user wants to apply to other venue.

8. Discussion

To improve the program, we can include the use of Graphical User Interface so that the program can be more interactive to the user. If I have more time to improve the program, I will include the restricted time for user to enter the hospital, such as no more visitors allowed to apply for the restricted spots after a certain time. It will be interesting to add the rough display of the user's and the available visitors' location of the restricted spot to determine the contact status in the form of Graphical User Interface. In addition to that, I would like to implement a file system to record, select, update, and delete the visitor information and retain the information even after the program has ended so that the administration office can check the visitor information anytime they want to. However, in this program, it assumes that the program already has an available database to store user's current detail and the details that have entered by user in the database is inserted into the database. Next, if I was given more time, the program should also record down every location that user has entered and does not allow user to reapply to the same location after they are allowed to enter or when they choose not to apply to the same location after the program has reached the maximum capacity. However, the restriction would be user is only allowed to apply to other spot after they have visited the spot that they are allowed entrance. With this, the hospital system can guarantee that the chance of a visitor applying to all restricted spot and ended up only going to one restricted spot even though the system has already added them to the current visitor number at the time they are allowed to enter at minimal.

9. References

GeeksForGeeks. (2017, March 23). *Inheritance in Java - GeeksforGeeks*. GeeksforGeeks. https://www.geeksforgeeks.org/inheritance-in-java/

Inheritance (The Java™ Tutorials > Learning the Java Language > Interfaces and Inheritance). (n.d.). Docs.oracle.com. https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html

Java Constructors. (2019). W3schools.com. https://www.w3schools.com/java/java_constructors.asp

Java Inheritance (With Examples). (n.d.). Www.programiz.com. https://www.programiz.com/java-programming/inheritance

Method Overriding in Java - javatpoint. (n.d.). Www.javatpoint.com. https://www.javatpoint.com/method-overriding-in-java