COMP 1029 – Programming Paradigms
Name: Lua Chong En
Student ID: 20417309

Java Coursework Report

To start off, I learned a lot whilst working on this Java coursework. I gained very meaningful knowledge on how to apply Object Oriented Programming into my code. Learning how to use inheritance, polymorphism, and method overloading. Applying theoretical knowledge learnt from the classroom to the coursework. After learning the C programming language, I felt like there was a big difference in coding in Java compared to C. But after completing the coursework, I really enjoyed learning a new programming language. I would like to thank Ms. Doreen as well for providing very critical and effective feedback which I used to improve my work.

The challenges I faced when completing this coursework was implementing Object Oriented Programming. I had issues understanding how to implement inheritance into my code. After being able to fully understand how OOP works in Java, I was able to successfully implement inheritance, polymorphism and method overloading in my code.

Overall, this was a very insightful coursework. I was able to experience using OOP which is important in software development, and I found this coursework very fulfilling as I was able to finish it ahead of schedule.

My assumptions doing this coursework is that the user can move freely around the hospital, given that the user is 1 meter or more away from all 4 directions, the user has the freedom to move and go to as many areas in the hospital as they wish. There is no restriction set on the user if they are deemed as normal contact. However, if the user is deemed casual contact or high risk, there are restrictions and rules imposed. My rules for contact status is that if the users entered distance is < 0.5 then it is High Risk, between 0.5 and 1.0 then it is Casual Contact and if > 1.0 then is Normal.

My code has 3 classes. These classes are Restricted_Spots, Static_Distancing, and Dynamic_Distancing.

Class 1: Restricted Spots

Within the Restricted_Spots class, here is where all the 6 different areas in my program are initialized. I created another class called ‘Area’ which is a nested class. In the Area class include the instance variables for each of the Area’s attributes. Nested inside the ‘Area’ class is a constructor which takes in 5 arguments, and it initializes each of the instance variables for the ‘Area’ class. As for the random number generator, I used math.random and used it to generate a random number of people inside the area, the ranges are from 0 to SpotMaximumCapacity. As for SpotMaximumCapacity, I divided SpotArea by 4 as there are 4 directions of 1 meter. Following is a method called ‘permission’ which determines whether the user will be able to enter the Area or not, it takes 2 parameters and uses an if else block and the value returned will be used in the Static_Distancing class to overall determine entrance or not. The ‘display’ method prints out each of the Area’s attributes as well as my Student ID and Name. In addition, it prints out the current date and time. To print out the

Date and Time, I imported 3 packages and formatted the date and time to be more readable. ofPattern is a method of DateTimeFormatter that allows me to create my own pattern for the date and time. Then I have 6 different subclasses which inherits from parent class 'Area', each of the 6 classes uses the 'extends' keyword to indicate inheritance and on top of that, I utilized the 'super' keyword which allowed me to set the specified values of the subclass by overriding the constructor, here I utilized another OOP method which is Polymorphism.

As for the access modifier is all public because the methods are accessed from the Static Distancing class. For the permission method, it is public static int because it returns an integer value depending on the if else block. For the display method, it is public static void because there is no return value.

Class 2: Dynamic Distancing

Moving on to the Dynamic_Distancing class, this is the class where I determine the dynamic distancing in the 4 directions, user's contact status, user's mask rules and clinic appointment. My 3 extraordinary features are: user's mask rules, clinic appointment, and FindShortestDistance method. User's mask rules and clinic appointment utilize the FindShortestDistance method which finds the shortest distance from 4 directions. The Direction nested class takes in 4 parameters from 4 directions and uses a series of if blocks to determine the distance between the user and other individuals in the area. Given that the user enters a distance less than 1 meter, the series of if blocks calculate how much the user will need to move away to practice dynamic distancing. Moving on to the Contact Status method, it takes is 4 parameters as well from 4 different directions. There is an array of 3 strings which are used when printing out the contact status within a series of if else blocks to determine contact status. The array is accessed by indexing. ANSI color escape sequence is also used here to add some color to the strings printed to signify importance. Following, there is another method called contact status which takes in 1 parameter, this method is used to display mask rules for the user. This method utilizes method overloading as there are 2 methods with the same name but different number of parameters. The parameter passed into this method is the return value from FindShortestDistance. Then finally the Clinic Appointment method uses a random date generator to generate a random date for the user to attend a hospital appointment based on their distance and a series of if else blocks. It utilizes the LocalDate package to get the current year, month, and day.

As for the Contact Rules, if any of the entered distance is less than 0.5 meters, then the contact status will be determined as High Risk. If the entered distance is between 0.5 and 1 meter, then the contact status will be determined as Casual Contact. And if the entered distance is over or equal to 1 meter, then the contact status will be determined as Normal. These rules apply for dynamic distancing in the 4 directions, user's contact status, user's mask rules and clinic appointment.

For the methods Direction, Contact Status (4 parameters), Contact Status (1 parameter), and clinic appointment. They are all public static void as they have no return value. But for FindShortestDistance, it has a floating-point return value hence it is public static float. The access modifier is all public because the methods are accessed from the Static Distancing class.

Class 3: Static Distancing

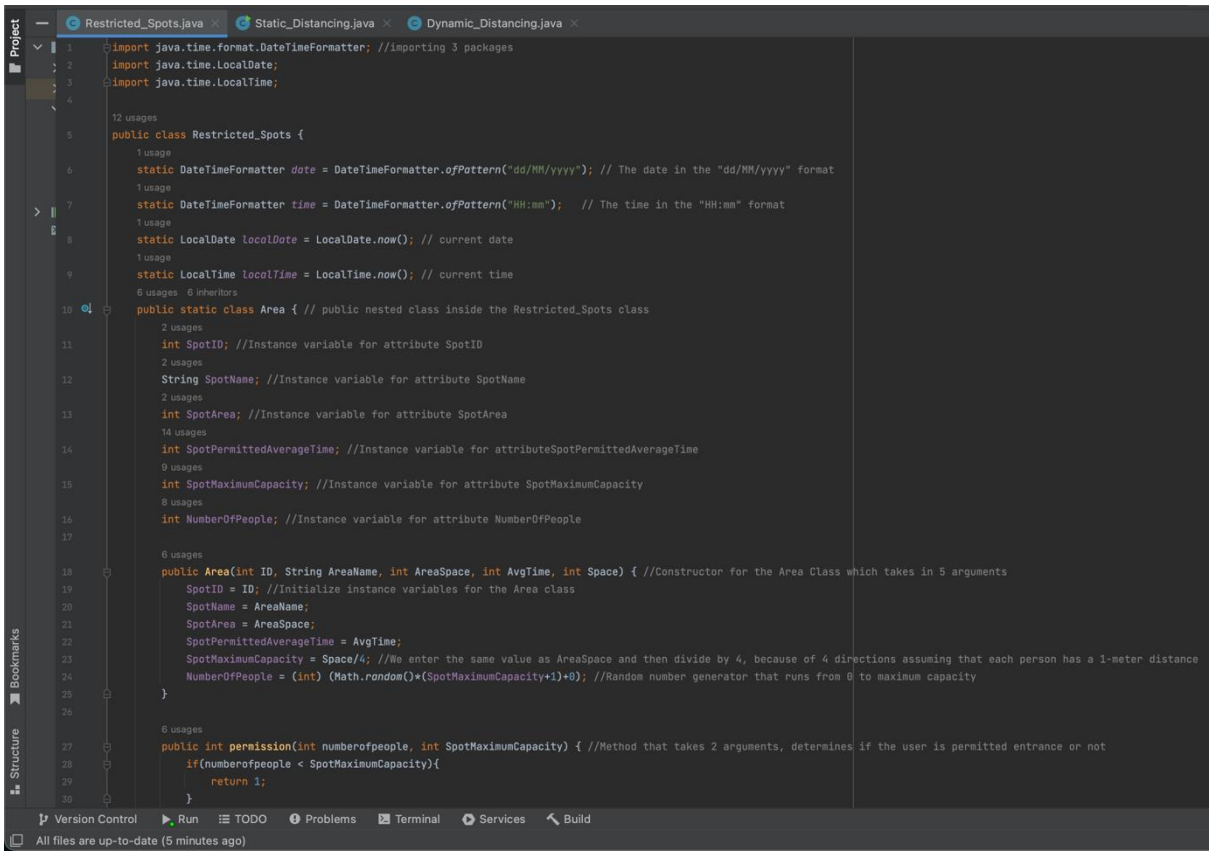
Moving onto the Static Distancing class, here is where main is located. I utilized inheritance by using the 'extend' keyword, where the SD subclass inherits the Dynamic Distancing class, so within main, methods will be called from SD. Within main, there is a switch that is used to allow the user to choose their desired area to enter. Before the switch, I used a do-while loop to create a loop with a condition that will only continue given that the while condition is satisfied. The switch has 7 choices, the first 6 are the areas of the hospital and the last choice is exit, there is also a default case which is added. I used the """ syntax to create a multi-line string for easier reading when printing the 7 choices. I imported the scanner object and used scanner to take in user input for the switch and the do-while loop. For each of the 6 cases, I created an object of the Area class from Restricted_Spots. By creating this object, it allowed me to access the methods in the class and utilize it in main. There is then an 'if' statement that checks if permission is permitted to enter the area or not. If the user is permitted to enter the area, it means that the area is not full and then the system will proceed to ask the user to enter the distance between them and the person to their front, back, left, and right. This and the other methods are from the SD subclass which are inherited from Dynamic Distancing. This method will then return the distances that the user will need to move in the relevant directions given that any of the distances is less than 1 meter. Continuing, Contact Status method is called here using the 4 parameters, this method returns the 'Contact Status' of the user. The next method is also Contact Status, but this is using method overloading, it uses the FindShortestDistance return value as a parameter and determines the 'Mask Rules' for the user. The final method called is Clinic Appointment which also uses the FindShortestDistance return value as a parameter and uses a random date generator to book a hospital checkup appointment for the user depending on their contact status.

However, given that the area is full, the system will ask the user to wait and ask for their response. If the user does choose to wait, then the system will assume that the SpotPermittedAverageTime has passed, and the user can enter. Then the functions inherited by SD from Dynamic Distancing class will then all be called.

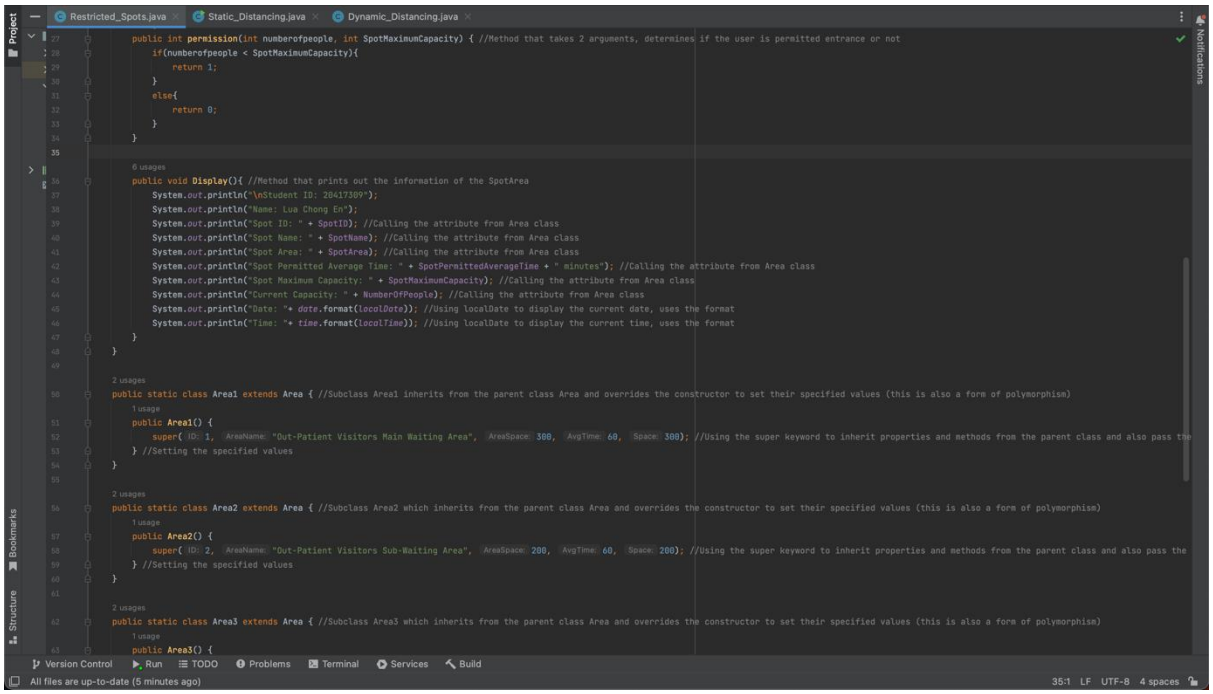
At the end of the switch case, the system will ask if the user wants to enter another area of the hospital and take in user input. The while condition will take in the input and determine to either continue or end the loop which will end the program. I utilized the object.equals to help determine if the input by the user is valid or not.

Code Screenshots with Comments

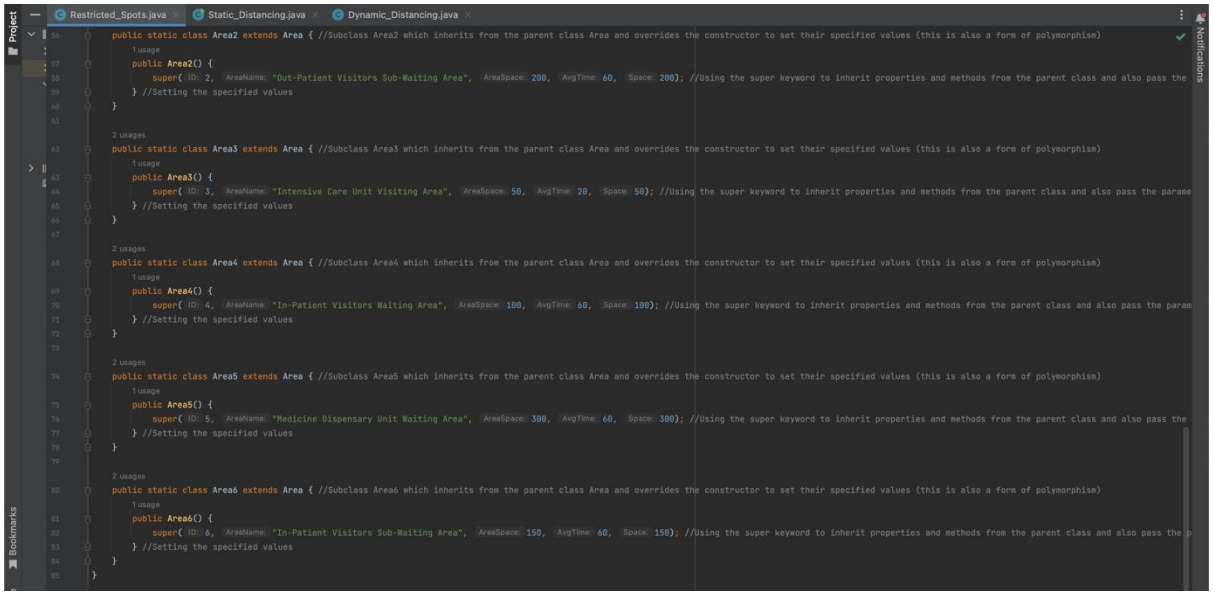
Class 1: Restricted Spots



```java
import java.time.format.DateTimeFormatter; //importing 3 packages
import java.time.LocalDate;
import java.time.LocalTime;

public class Restricted_Spots {
    static DateTimeFormatter date = DateTimeFormatter.ofPattern("dd/MM/yyyy"); // The date in the "dd/MM/yyyy" format
    static DateTimeFormatter time = DateTimeFormatter.ofPattern("HH:mm");   // The time in the "HH:mm" format
    static LocalDate localDate = LocalDate.now(); // current date
    static LocalTime localTime = LocalTime.now(); // current time
    public static class Area { // public nested class inside the Restricted_Spots class
        int SpotID; //Instance variable for attribute SpotID
        String SpotName; //Instance variable for attribute SpotName
        int SpotArea; //Instance variable for attribute SpotArea
        int SpotPermittedAverageTime; //Instance variable for attributeSpotPermittedAverageTime
        int SpotMaximumCapacity; //Instance variable for attribute SpotMaximumCapacity
        int NumberOfPeople; //Instance variable for attribute NumberOfPeople

        public Area(int ID, String AreaName, int AreaSpace, int AvgTime, int Space) { //Constructor for the Area Class which takes in 5 arguments
            SpotID = ID; //Initialize instance variables for the Area class
            SpotName = AreaName;
            SpotArea = AreaSpace;
            SpotPermittedAverageTime = AvgTime;
            SpotMaximumCapacity = Space/4; //We enter the same value as AreaSpace and then divide by 4, because of 4 directions assuming that each person has a 1-meter distance
            NumberOfPeople = (int) (Math.random()*(SpotMaximumCapacity+1)+0); //Random number generator that runs from 0 to maximum capacity
        }

        public int permission(int numberofpeople, int SpotMaximumCapacity) { //Method that takes 2 arguments, determines if the user is permitted entrance or not
            if(numberofpeople < SpotMaximumCapacity){
                return 1;
            }
```

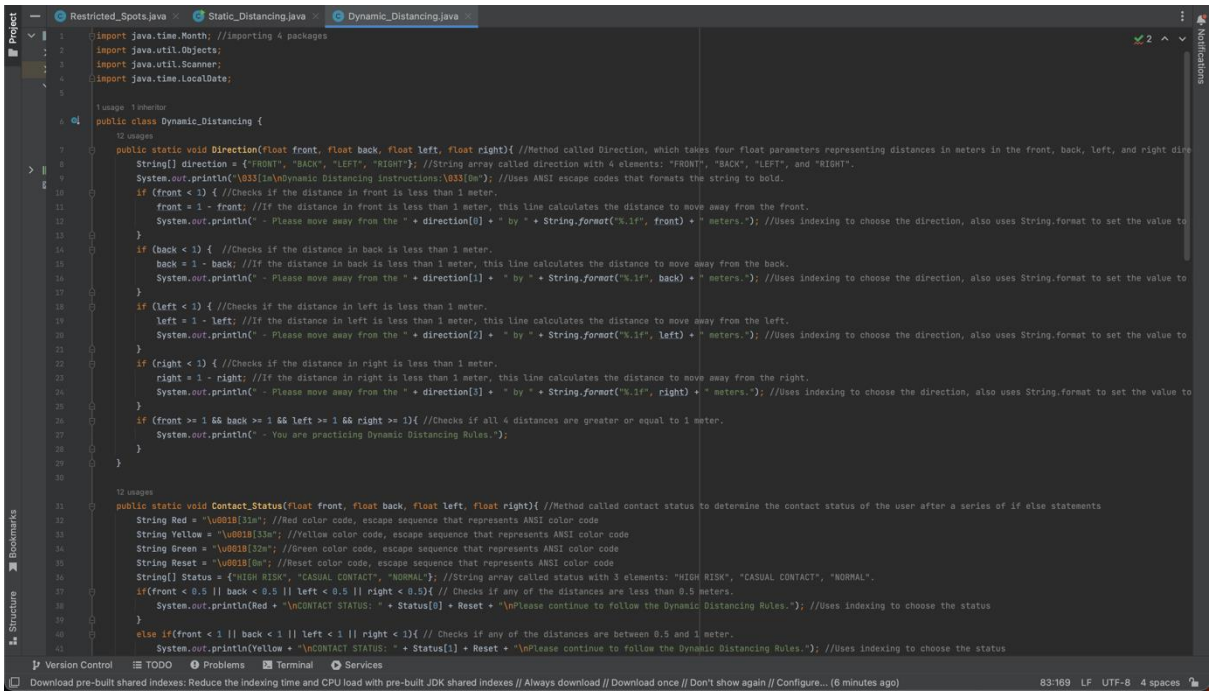


```java
        public int permission(int numberofpeople, int SpotMaximumCapacity) { //Method that takes 2 arguments, determines if the user is permitted entrance or not
            if(numberofpeople < SpotMaximumCapacity){
                return 1;
            }
            else{
                return 0;
            }
        }

        public void Display(){ //Method that prints out the information of the SpotArea
            System.out.println("\nStudent ID: 2041730P");
            System.out.println("Name: Lua Chong En");
            System.out.println("Spot ID: " + SpotID); //Calling the attribute from Area class
            System.out.println("Spot Name: " + SpotName); //Calling the attribute from Area class
            System.out.println("Spot Area: " + SpotArea); //Calling the attribute from Area class
            System.out.println("Spot Permitted Average Time: " + SpotPermittedAverageTime + " minutes"); //Calling the attribute from Area class
            System.out.println("Spot Maximum Capacity: " + SpotMaximumCapacity); //Calling the attribute from Area class
            System.out.println("Current Capacity: " + NumberOfPeople); //Calling the attribute from Area class
            System.out.println("Date: "+ date.format(localDate)); //Using localDate to display the current date, uses the format
            System.out.println("Time: "+ time.format(localTime)); //Using localDate to display the current time, uses the format
        }
    }

    public static class Area1 extends Area { //Subclass Area1 inherits from the parent class Area and overrides the constructor to set their specified values (this is also a form of polymorphism)
        public Area1() {
            super( ID: 1,  AreaName: "Out-Patient Visitors Main Waiting Area",  AreaSpace: 300,  AvgTime: 60,  Space: 300); //Using the super keyword to inherit properties and methods from the parent class and also pass the
        } //Setting the specified values
    }

    public static class Area2 extends Area { //Subclass Area2 which inherits from the parent class Area and overrides the constructor to set their specified values (this is also a form of polymorphism)
        public Area2() {
            super( ID: 2,  AreaName: "Out-Patient Visitors Sub-Waiting Area",  AreaSpace: 200,  AvgTime: 60,  Space: 200); //Using the super keyword to inherit properties and methods from the parent class and also pass the
        } //Setting the specified values
    }

    public static class Area3 extends Area { //Subclass Area3 which inherits from the parent class Area and overrides the constructor to set their specified values (this is also a form of polymorphism)
        public Area3() {
```

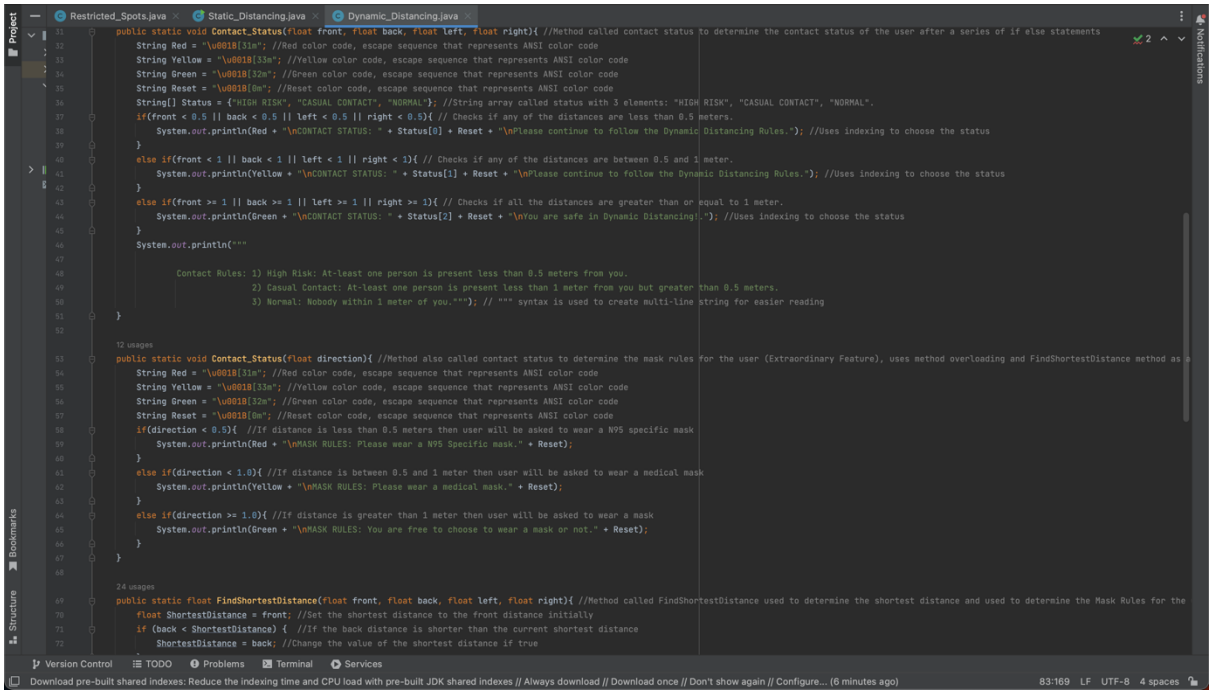


```
Restricted_Spots.java | Static_Distancing.java | Dynamic_Distancing.java

    public static class Area2 extends Area { //Subclass Area2 which inherits from the parent class Area and overrides the constructor to set their specified values (this is also a form of polymorphism)
        1 usage
        public Area2() {
            super( ID: 2,  AreaName: "Out-Patient Visitors Sub-Waiting Area",  AreaSpace: 200,  AvgTime: 60,  Space: 200); //Using the super keyword to inherit properties and methods from the parent class and also pass the
        } //Setting the specified values
    }

    2 usages
    public static class Area3 extends Area { //Subclass Area3 which inherits from the parent class Area and overrides the constructor to set their specified values (this is also a form of polymorphism)
        1 usage
        public Area3() {
            super( ID: 3,  AreaName: "Intensive Care Unit Visiting Area",  AreaSpace: 50,  AvgTime: 20,  Space: 50); //Using the super keyword to inherit properties and methods from the parent class and also pass the parame
        } //Setting the specified values
    }

    2 usages
    public static class Area4 extends Area { //Subclass Area4 which inherits from the parent class Area and overrides the constructor to set their specified values (this is also a form of polymorphism)
        1 usage
        public Area4() {
            super( ID: 4,  AreaName: "In-Patient Visitors Waiting Area",  AreaSpace: 100,  AvgTime: 60,  Space: 100); //Using the super keyword to inherit properties and methods from the parent class and also pass the param
        } //Setting the specified values
    }

    2 usages
    public static class Area5 extends Area { //Subclass Area5 which inherits from the parent class Area and overrides the constructor to set their specified values (this is also a form of polymorphism)
        1 usage
        public Area5() {
            super( ID: 5,  AreaName: "Medicine Dispensary Unit Waiting Area",  AreaSpace: 300,  AvgTime: 60,  Space: 300); //Using the super keyword to inherit properties and methods from the parent class and also pass the
        } //Setting the specified values
    }

    2 usages
    public static class Area6 extends Area { //Subclass Area6 which inherits from the parent class Area and overrides the constructor to set their specified values (this is also a form of polymorphism)
        1 usage
        public Area6() {
            super( ID: 6,  AreaName: "In-Patient Visitors Sub-Waiting Area",  AreaSpace: 150,  AvgTime: 60,  Space: 150); //Using the super keyword to inherit properties and methods from the parent class and also pass the p
        } //Setting the specified values
    }
}
```

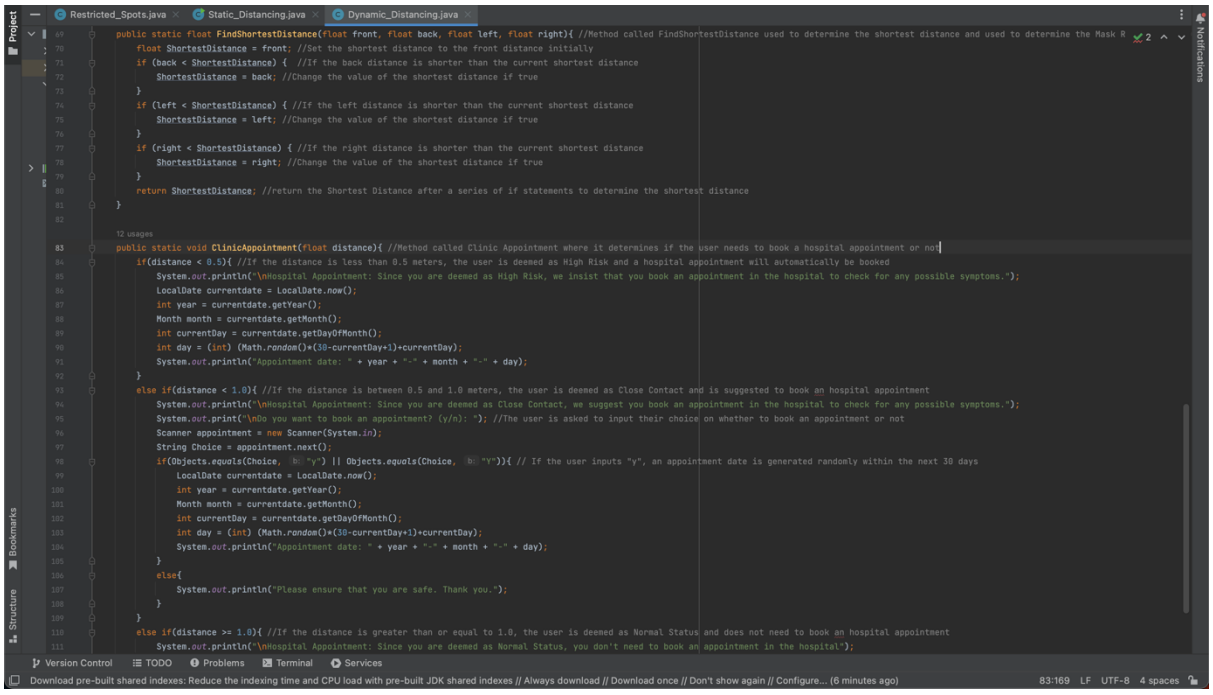
Class 2: Dynamic Distancing



```
Restricted_Spots.java | Static_Distancing.java | Dynamic_Distancing.java

import java.time.Month; //importing 4 packages
import java.util.Objects;
import java.util.Scanner;
import java.time.LocalDate;

1 usage  1 inheritor
public class Dynamic_Distancing {
    12 usages
    public static void Direction(float front, float back, float left, float right){ //Method called Direction, which takes four float parameters representing distances in meters in the front, back, left, and right dire
        String[] direction = {"FRONT", "BACK", "LEFT", "RIGHT"}; //String array called direction with 4 elements: "FRONT", "BACK", "LEFT", and "RIGHT".
        System.out.println("\033[1m\nDynamic Distancing instructions:\033[0m"); //Uses ANSI escape codes that formats the string to bold.
        if (front < 1) { //Checks if the distance in front is less than 1 meter.
            front = 1 - front; //If the distance in front is less than 1 meter, this line calculates the distance to move away from the front.
            System.out.println(" - Please move away from the " + direction[0] + " by " + String.format("%.1f", front) + " meters."); //Uses indexing to choose the direction, also uses String.format to set the value to
        }
        if (back < 1) {  //Checks if the distance in back is less than 1 meter.
            back = 1 - back; //If the distance in back is less than 1 meter, this line calculates the distance to move away from the back.
            System.out.println(" - Please move away from the " + direction[1] +  " by " + String.format("%.1f", back) + " meters."); //Uses indexing to choose the direction, also uses String.format to set the value to
        }
        if (left < 1) { //Checks if the distance in left is less than 1 meter.
            left = 1 - left; //If the distance in left is less than 1 meter, this line calculates the distance to move away from the left.
            System.out.println(" - Please move away from the " + direction[2] +  " by " + String.format("%.1f", left) + " meters."); //Uses indexing to choose the direction, also uses String.format to set the value to
        }
        if (right < 1) { //Checks if the distance in right is less than 1 meter.
            right = 1 - right; //If the distance in right is less than 1 meter, this line calculates the distance to move away from the right.
            System.out.println(" - Please move away from the " + direction[3] +  " by " + String.format("%.1f", right) + " meters."); //Uses indexing to choose the direction, also uses String.format to set the value to
        }
        if (front >= 1 && back >= 1 && left >= 1 && right >= 1){ //Checks if all 4 distances are greater or equal to 1 meter.
            System.out.println(" - You are practicing Dynamic Distancing Rules.");
        }
    }

    12 usages
    public static void Contact_Status(float front, float back, float left, float right){ //Method called contact status to determine the contact status of the user after a series of if else statements
        String Red = "\u001B[31m"; //Red color code, escape sequence that represents ANSI color code
        String Yellow = "\u001B[33m"; //Yellow color code, escape sequence that represents ANSI color code
        String Green = "\u001B[32m"; //Green color code, escape sequence that represents ANSI color code
        String Reset = "\u001B[0m"; //Reset color code, escape sequence that represents ANSI color code
        String[] Status = {"HIGH RISK", "CASUAL CONTACT", "NORMAL"}; //String array called status with 3 elements: "HIGH RISK", "CASUAL CONTACT", "NORMAL".
        if(front < 0.5 || back < 0.5 || left < 0.5 || right < 0.5){ // Checks if any of the distances are less than 0.5 meters.
            System.out.println(Red + "\nCONTACT STATUS: " + Status[0] + Reset + "\nPlease continue to follow the Dynamic Distancing Rules."); //Uses indexing to choose the status
        }
        else if(front < 1 || back < 1 || left < 1 || right < 1){ // Checks if any of the distances are between 0.5 and 1 meter.
            System.out.println(Yellow + "\nCONTACT STATUS: " + Status[1] + Reset + "\nPlease continue to follow the Dynamic Distancing Rules."); //Uses indexing to choose the status

Version Control  TODO  Problems  Terminal  Services
Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built JDK shared indexes // Always download // Download once // Don't show again // Configure... (6 minutes ago)
83:169  LF  UTF-8  4 spaces
```

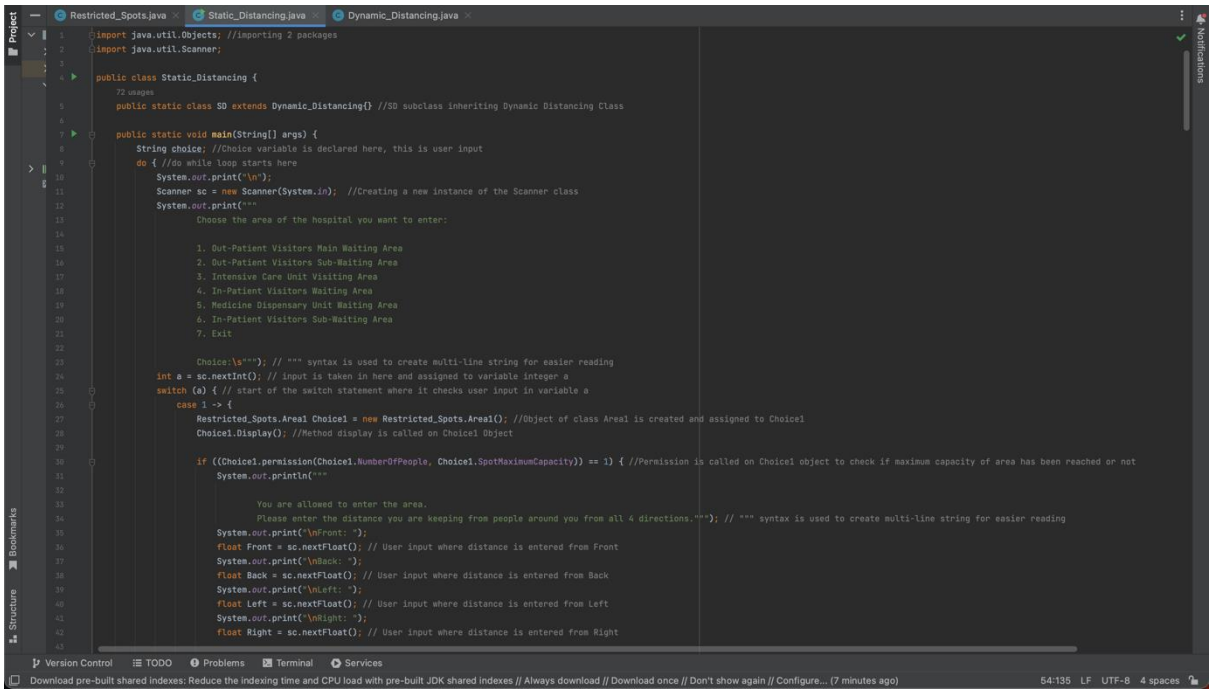


```java
public static void Contact_Status(float front, float back, float left, float right){ //Method called contact status to determine the contact status of the user after a series of if else statements
    String Red = "\u001B[31m"; //Red color code, escape sequence that represents ANSI color code
    String Yellow = "\u001B[33m"; //Yellow color code, escape sequence that represents ANSI color code
    String Green = "\u001B[32m"; //Green color code, escape sequence that represents ANSI color code
    String Reset = "\u001B[0m"; //Reset color code, escape sequence that represents ANSI color code
    String[] Status = {"HIGH RISK", "CASUAL CONTACT", "NORMAL"}; //String array called status with 3 elements: "HIGH RISK", "CASUAL CONTACT", "NORMAL".
    if(front < 0.5 || back < 0.5 || left < 0.5 || right < 0.5){ // Checks if any of the distances are less than 0.5 meters.
        System.out.println(Red + "\nCONTACT STATUS: " + Status[0] + Reset + "\nPlease continue to follow the Dynamic Distancing Rules."); //Uses indexing to choose the status
    }
    else if(front < 1 || back < 1 || left < 1 || right < 1){ // Checks if any of the distances are between 0.5 and 1 meter.
        System.out.println(Yellow + "\nCONTACT STATUS: " + Status[1] + Reset + "\nPlease continue to follow the Dynamic Distancing Rules."); //Uses indexing to choose the status
    }
    else if(front >= 1 || back >= 1 || left >= 1 || right >= 1){ // Checks if all the distances are greater than or equal to 1 meter.
        System.out.println(Green + "\nCONTACT STATUS: " + Status[2] + Reset + "\nYou are safe in Dynamic Distancing!."); //Uses indexing to choose the status
    }
    System.out.println("""

            Contact Rules: 1) High Risk: At-least one person is present less than 0.5 meters from you.
                           2) Casual Contact: At-least one person is present less than 1 meter from you but greater than 0.5 meters.
                           3) Normal: Nobody within 1 meter of you."""); // """ syntax is used to create multi-line string for easier reading
}

12 usages
public static void Contact_Status(float direction){ //Method also called contact status to determine the mask rules for the user (Extraordinary Feature), uses method overloading and FindShortestDistance method as a
    String Red = "\u001B[31m"; //Red color code, escape sequence that represents ANSI color code
    String Yellow = "\u001B[33m"; //Yellow color code, escape sequence that represents ANSI color code
    String Green = "\u001B[32m"; //Green color code, escape sequence that represents ANSI color code
    String Reset = "\u001B[0m"; //Reset color code, escape sequence that represents ANSI color code
    if(direction < 0.5){  //If distance is less than 0.5 meters then user will be asked to wear a N95 specific mask
        System.out.println(Red + "\nMASK RULES: Please wear a N95 Specific mask." + Reset);
    }
    else if(direction < 1.0){ //If distance is between 0.5 and 1 meter then user will be asked to wear a medical mask
        System.out.println(Yellow + "\nMASK RULES: Please wear a medical mask." + Reset);
    }
    else if(direction >= 1.0){ //If distance is greater than 1 meter then user will be asked to wear a mask
        System.out.println(Green + "\nMASK RULES: You are free to choose to wear a mask or not." + Reset);
    }
}

24 usages
public static float FindShortestDistance(float front, float back, float left, float right){ //Method called FindShortestDistance used to determine the shortest distance and used to determine the Mask Rules for the
    float ShortestDistance = front; //Set the shortest distance to the front distance initially
    if (back < ShortestDistance) {  //If the back distance is shorter than the current shortest distance
        ShortestDistance = back; //Change the value of the shortest distance if true
```

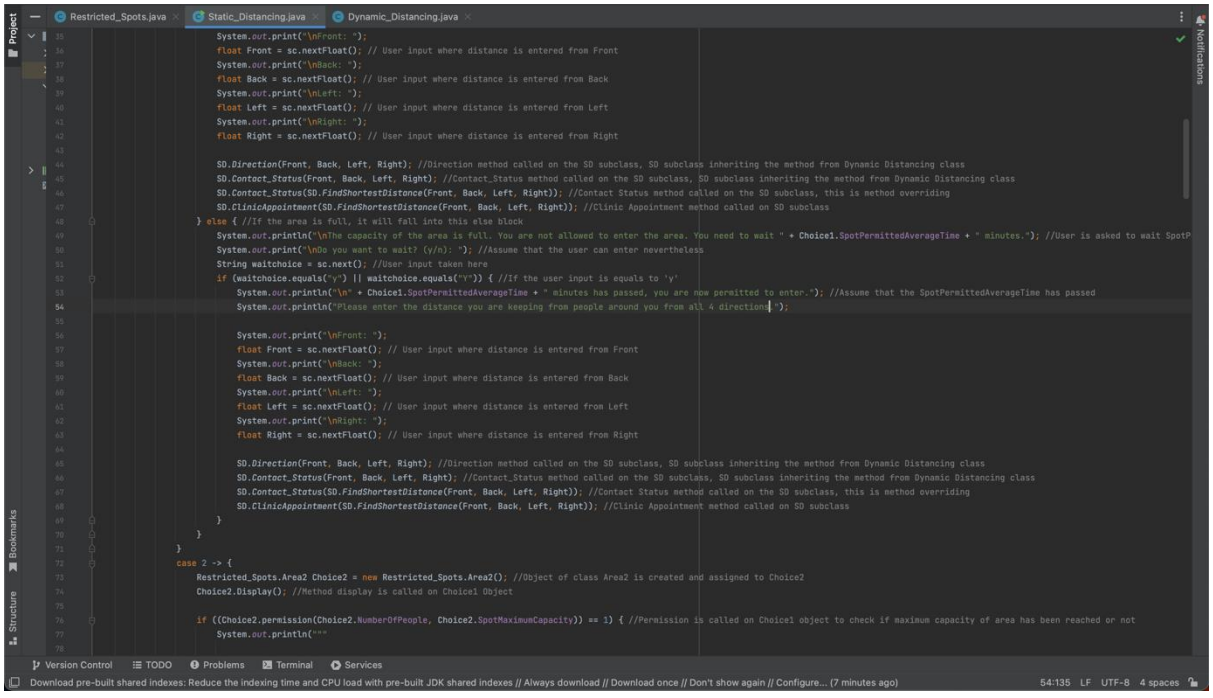


```java
public static float FindShortestDistance(float front, float back, float left, float right){ //Method called FindShortestDistance used to determine the shortest distance and used to determine the Mask R
    float ShortestDistance = front; //Set the shortest distance to the front distance initially
    if (back < ShortestDistance) {  //If the back distance is shorter than the current shortest distance
        ShortestDistance = back; //Change the value of the shortest distance if true
    }
    if (left < ShortestDistance) { //If the left distance is shorter than the current shortest distance
        ShortestDistance = left; //Change the value of the shortest distance if true
    }
    if (right < ShortestDistance) { //If the right distance is shorter than the current shortest distance
        ShortestDistance = right; //Change the value of the shortest distance if true
    }
    return ShortestDistance; //return the Shortest Distance after a series of if statements to determine the shortest distance
}

12 usages
public static void ClinicAppointment(float distance){ //Method called Clinic Appointment where it determines if the user needs to book a hospital appointment or not
    if(distance < 0.5){ //If the distance is less than 0.5 meters, the user is deemed as High Risk and a hospital appointment will automatically be booked
        System.out.println("\nHospital Appointment: Since you are deemed as High Risk, we insist that you book an appointment in the hospital to check for any possible symptoms.");
        LocalDate currentdate = LocalDate.now();
        int year = currentdate.getYear();
        Month month = currentdate.getMonth();
        int currentDay = currentdate.getDayOfMonth();
        int day = (int) (Math.random()*(30-currentDay+1)+currentDay);
        System.out.println("Appointment date: " + year + "-" + month + "-" + day);
    }
    else if(distance < 1.0){ //If the distance is between 0.5 and 1.0 meters, the user is deemed as Close Contact and is suggested to book an hospital appointment
        System.out.println("\nHospital Appointment: Since you are deemed as Close Contact, we suggest you book an appointment in the hospital to check for any possible symptoms.");
        System.out.print("\nDo you want to book an appointment? (y/n): "); //The user is asked to input their choice on whether to book an appointment or not
        Scanner appointment = new Scanner(System.in);
        String Choice = appointment.next();
        if(Objects.equals(Choice, b: "y") || Objects.equals(Choice, b: "Y")){ // If the user inputs "y", an appointment date is generated randomly within the next 30 days
            LocalDate currentdate = LocalDate.now();
            int year = currentdate.getYear();
            Month month = currentdate.getMonth();
            int currentDay = currentdate.getDayOfMonth();
            int day = (int) (Math.random()*(30-currentDay+1)+currentDay);
            System.out.println("Appointment date: " + year + "-" + month + "-" + day);
        }
        else{
            System.out.println("Please ensure that you are safe. Thank you.");
        }
    }
    else if(distance >= 1.0){ //If the distance is greater than or equal to 1.0, the user is deemed as Normal Status and does not need to book an hospital appointment
        System.out.println("\nHospital Appointment: Since you are deemed as Normal Status, you don't need to book an appointment in the hospital");
```

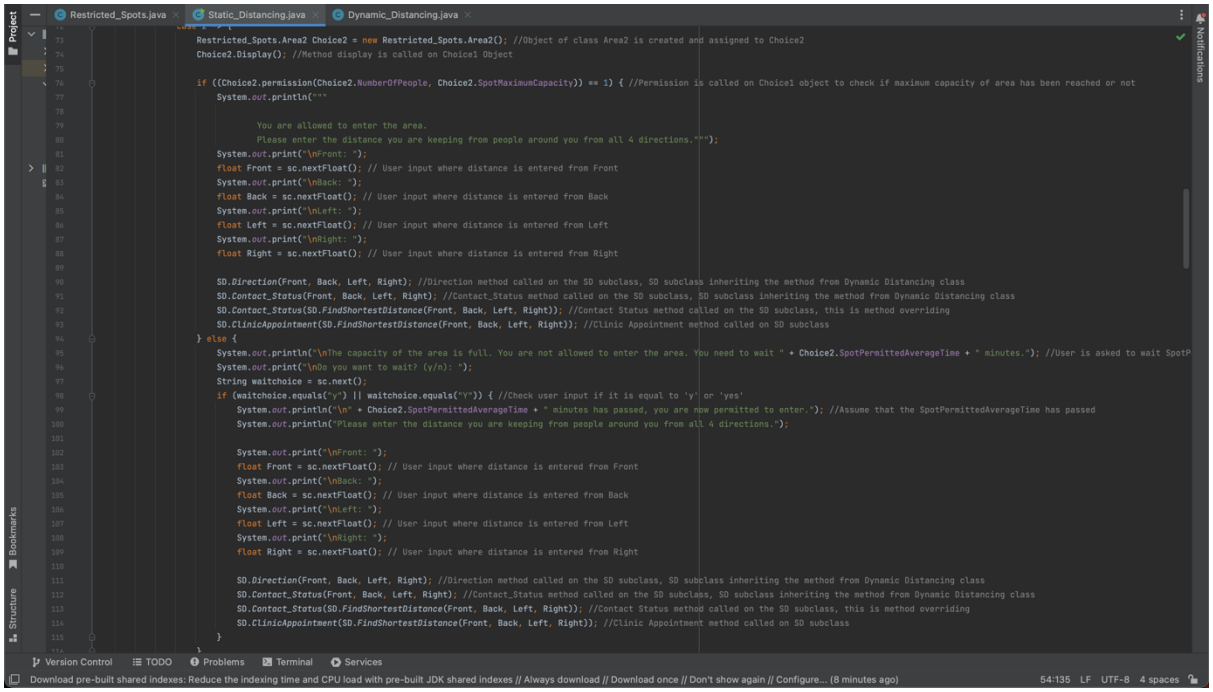
Class 3: Static Distancing



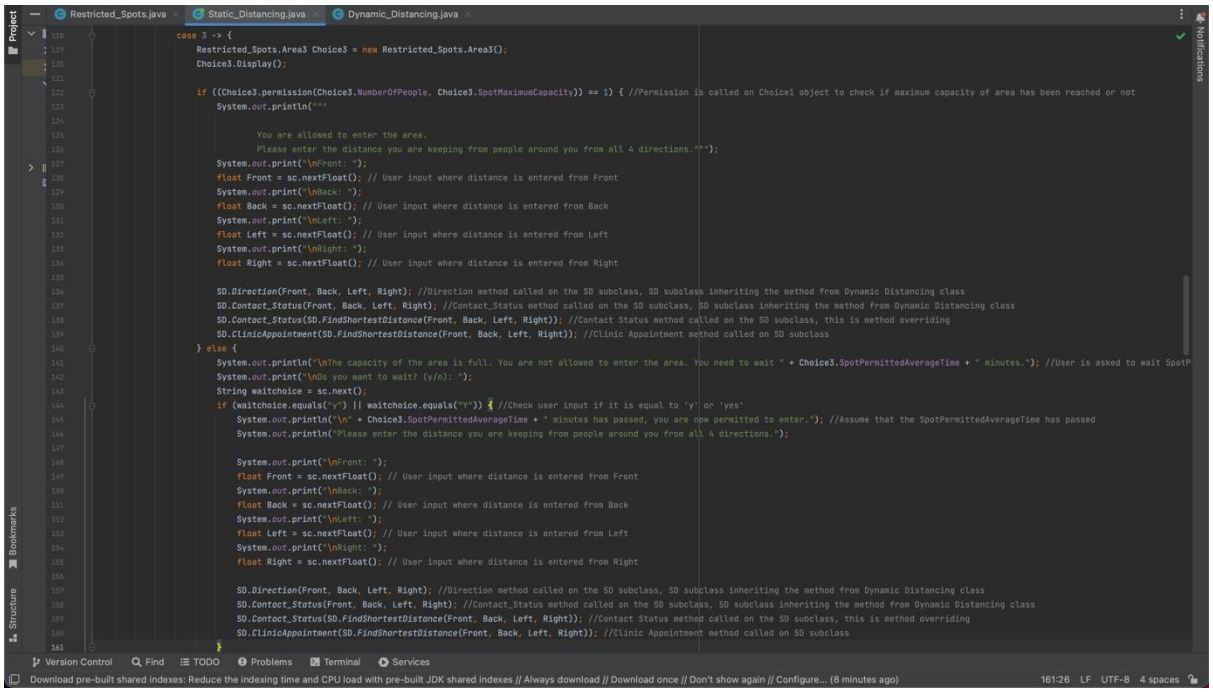
```java
import java.util.Objects; //importing 2 packages
import java.util.Scanner;

public class Static_Distancing {
    72 usages
    public static class SD extends Dynamic_Distancing{} //SD subclass inheriting Dynamic Distancing Class

    public static void main(String[] args) {
        String choice; //Choice variable is declared here, this is user input
        do { //do while loop starts here
            System.out.print("\n");
            Scanner sc = new Scanner(System.in);  //Creating a new instance of the Scanner class
            System.out.print("""
                    Choose the area of the hospital you want to enter:

                    1. Out-Patient Visitors Main Waiting Area
                    2. Out-Patient Visitors Sub-Waiting Area
                    3. Intensive Care Unit Visiting Area
                    4. In-Patient Visitors Waiting Area
                    5. Medicine Dispensary Unit Waiting Area
                    6. In-Patient Visitors Sub-Waiting Area
                    7. Exit

                    Choice:\s"""); // """ syntax is used to create multi-line string for easier reading
            int a = sc.nextInt(); // input is taken in here and assigned to variable integer a
            switch (a) { // start of the switch statement where it checks user input in variable a
                case 1 -> {
                    Restricted_Spots.Area1 Choice1 = new Restricted_Spots.Area1(); //Object of class Area1 is created and assigned to Choice1
                    Choice1.Display(); //Method display is called on Choice1 Object

                    if ((Choice1.permission(Choice1.NumberOfPeople, Choice1.SpotMaximumCapacity)) == 1) { //Permission is called on Choice1 object to check if maximum capacity of area has been reached or not
                        System.out.println("""

                                You are allowed to enter the area.
                                Please enter the distance you are keeping from people around you from all 4 directions."""); // """ syntax is used to create multi-line string for easier reading
                        System.out.print("\nFront: ");
                        float Front = sc.nextFloat(); // User input where distance is entered from Front
                        System.out.print("\nBack: ");
                        float Back = sc.nextFloat(); // User input where distance is entered from Back
                        System.out.print("\nLeft: ");
                        float Left = sc.nextFloat(); // User input where distance is entered from Left
                        System.out.print("\nRight: ");
                        float Right = sc.nextFloat(); // User input where distance is entered from Right
```



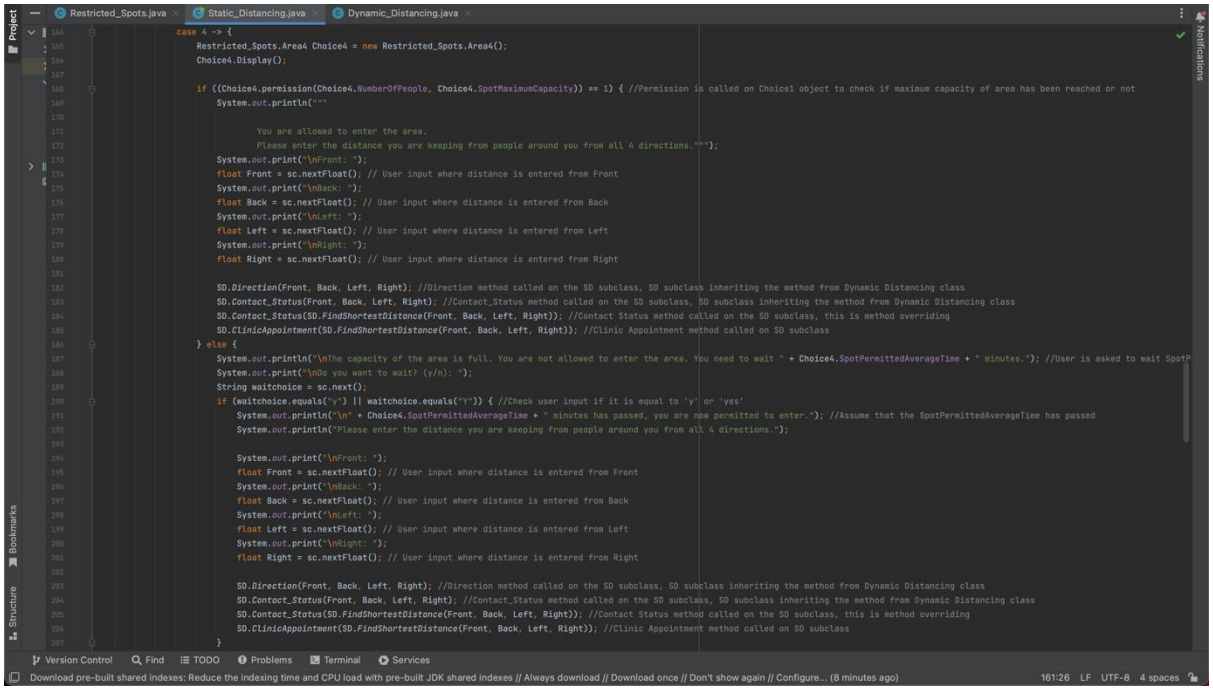
```java
                        System.out.print("\nFront: ");
                        float Front = sc.nextFloat(); // User input where distance is entered from Front
                        System.out.print("\nBack: ");
                        float Back = sc.nextFloat(); // User input where distance is entered from Back
                        System.out.print("\nLeft: ");
                        float Left = sc.nextFloat(); // User input where distance is entered from Left
                        System.out.print("\nRight: ");
                        float Right = sc.nextFloat(); // User input where distance is entered from Right

                        SD.Direction(Front, Back, Left, Right); //Direction method called on the SD subclass, SD subclass inheriting the method from Dynamic Distancing class
                        SD.Contact_Status(Front, Back, Left, Right); //Contact_Status method called on the SD subclass, SD subclass inheriting the method from Dynamic Distancing class
                        SD.Contact_Status(SD.FindShortestDistance(Front, Back, Left, Right)); //Contact Status method called on the SD subclass, this is method overriding
                        SD.ClinicAppointment(SD.FindShortestDistance(Front, Back, Left, Right)); //Clinic Appointment method called on SD subclass
                    } else { //If the area is full, it will fall into this else block
                        System.out.println("\nThe capacity of the area is full. You are not allowed to enter the area. You need to wait " + Choice1.SpotPermittedAverageTime + " minutes."); //User is asked to wait SpotP
                        System.out.print("\nDo you want to wait? (y/n): "); //Assume that the user can enter nevertheless
                        String waitchoice = sc.next(); //User input taken here
                        if (waitchoice.equals("y") || waitchoice.equals("Y")) { //If the user input is equals to 'y'
                            System.out.println("\n" + Choice1.SpotPermittedAverageTime + " minutes has passed, you are now permitted to enter."); //Assume that the SpotPermittedAverageTime has passed
                            System.out.println("Please enter the distance you are keeping from people around you from all 4 directions");

                            System.out.print("\nFront: ");
                            float Front = sc.nextFloat(); // User input where distance is entered from Front
                            System.out.print("\nBack: ");
                            float Back = sc.nextFloat(); // User input where distance is entered from Back
                            System.out.print("\nLeft: ");
                            float Left = sc.nextFloat(); // User input where distance is entered from Left
                            System.out.print("\nRight: ");
                            float Right = sc.nextFloat(); // User input where distance is entered from Right

                            SD.Direction(Front, Back, Left, Right); //Direction method called on the SD subclass, SD subclass inheriting the method from Dynamic Distancing class
                            SD.Contact_Status(Front, Back, Left, Right); //Contact_Status method called on the SD subclass, SD subclass inheriting the method from Dynamic Distancing class
                            SD.Contact_Status(SD.FindShortestDistance(Front, Back, Left, Right)); //Contact Status method called on the SD subclass, this is method overriding
                            SD.ClinicAppointment(SD.FindShortestDistance(Front, Back, Left, Right)); //Clinic Appointment method called on SD subclass
                        }
                    }
                }
                case 2 -> {
                    Restricted_Spots.Area2 Choice2 = new Restricted_Spots.Area2(); //Object of class Area2 is created and assigned to Choice2
                    Choice2.Display(); //Method display is called on Choice1 Object

                    if ((Choice2.permission(Choice2.NumberOfPeople, Choice2.SpotMaximumCapacity)) == 1) { //Permission is called on Choice1 object to check if maximum capacity of area has been reached or not
                        System.out.println("""
```



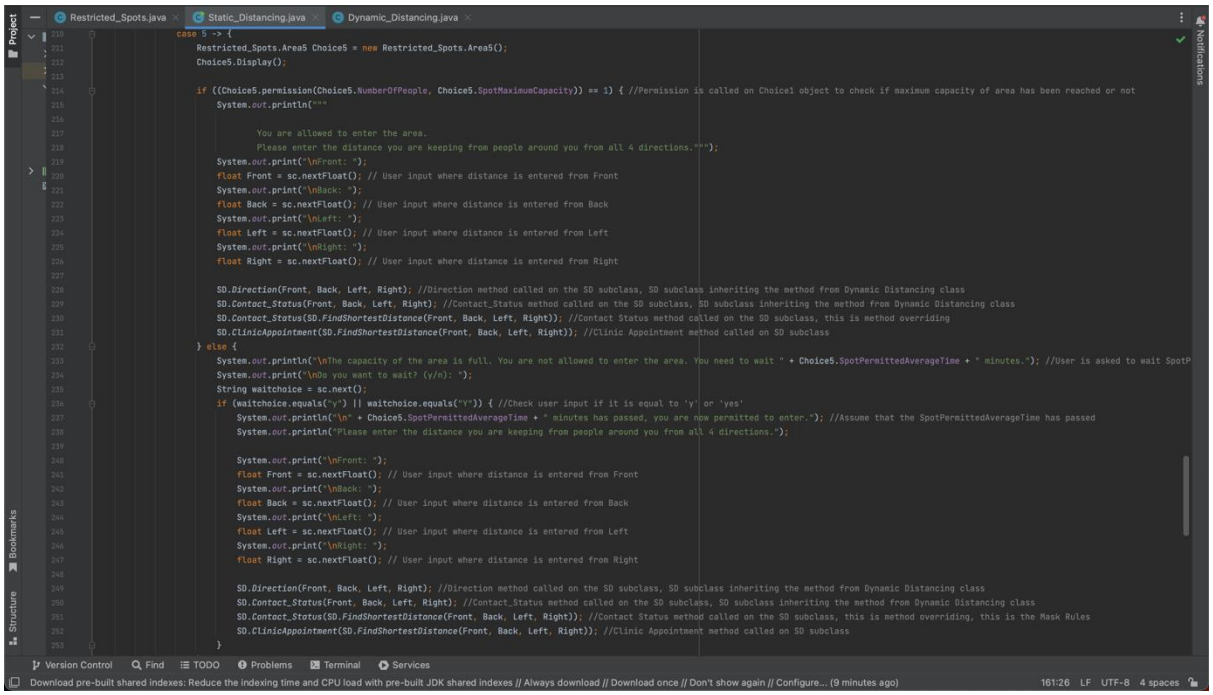
```
Restricted_Spots.Area2 Choice2 = new Restricted_Spots.Area2(); //Object of class Area2 is created and assigned to Choice2
Choice2.Display(); //Method display is called on Choice1 Object

if ((Choice2.permission(Choice2.NumberOfPeople, Choice2.SpotMaximumCapacity)) == 1) { //Permission is called on Choice1 object to check if maximum capacity of area has been reached or not
    System.out.println("""

            You are allowed to enter the area.
            Please enter the distance you are keeping from people around you from all 4 directions.""");
    System.out.print("\nFront: ");
    float Front = sc.nextFloat(); // User input where distance is entered from Front
    System.out.print("\nBack: ");
    float Back = sc.nextFloat(); // User input where distance is entered from Back
    System.out.print("\nLeft: ");
    float Left = sc.nextFloat(); // User input where distance is entered from Left
    System.out.print("\nRight: ");
    float Right = sc.nextFloat(); // User input where distance is entered from Right

    SD.Direction(Front, Back, Left, Right); //Direction method called on the SD subclass, SD subclass inheriting the method from Dynamic Distancing class
    SD.Contact_Status(Front, Back, Left, Right); //Contact_Status method called on the SD subclass, SD subclass inheriting the method from Dynamic Distancing class
    SD.Contact_Status(SD.FindShortestDistance(Front, Back, Left, Right)); //Contact Status method called on the SD subclass, this is method overriding
    SD.ClinicAppointment(SD.FindShortestDistance(Front, Back, Left, Right)); //Clinic Appointment method called on SD subclass
} else {
    System.out.println("\nThe capacity of the area is full. You are not allowed to enter the area. You need to wait " + Choice2.SpotPermittedAverageTime + " minutes."); //User is asked to wait SpotP
    System.out.print("\nDo you want to wait? (y/n): ");
    String waitchoice = sc.next();
    if (waitchoice.equals("y") || waitchoice.equals("Y")) { //Check user input if it is equal to 'y' or 'yes'
        System.out.println("\n" + Choice2.SpotPermittedAverageTime + " minutes has passed, you are now permitted to enter."); //Assume that the SpotPermittedAverageTime has passed
        System.out.println("Please enter the distance you are keeping from people around you from all 4 directions.");

        System.out.print("\nFront: ");
        float Front = sc.nextFloat(); // User input where distance is entered from Front
        System.out.print("\nBack: ");
        float Back = sc.nextFloat(); // User input where distance is entered from Back
        System.out.print("\nLeft: ");
        float Left = sc.nextFloat(); // User input where distance is entered from Left
        System.out.print("\nRight: ");
        float Right = sc.nextFloat(); // User input where distance is entered from Right

        SD.Direction(Front, Back, Left, Right); //Direction method called on the SD subclass, SD subclass inheriting the method from Dynamic Distancing class
        SD.Contact_Status(Front, Back, Left, Right); //Contact_Status method called on the SD subclass, SD subclass inheriting the method from Dynamic Distancing class
        SD.Contact_Status(SD.FindShortestDistance(Front, Back, Left, Right)); //Contact Status method called on the SD subclass, this is method overriding
        SD.ClinicAppointment(SD.FindShortestDistance(Front, Back, Left, Right)); //Clinic Appointment method called on SD subclass
    }
}
```



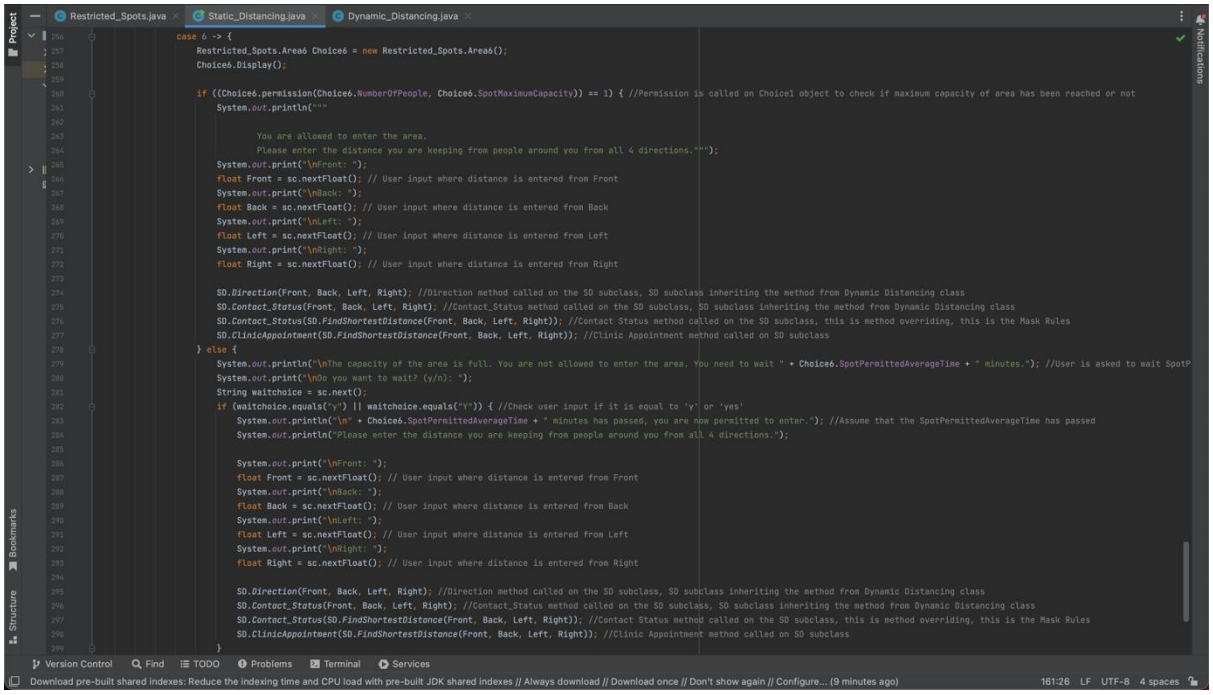
```
case 3 -> {
    Restricted_Spots.Area3 Choice3 = new Restricted_Spots.Area3();
    Choice3.Display();

    if ((Choice3.permission(Choice3.NumberOfPeople, Choice3.SpotMaximumCapacity)) == 1) { //Permission is called on Choice1 object to check if maximum capacity of area has been reached or not
        System.out.println("""

                You are allowed to enter the area.
                Please enter the distance you are keeping from people around you from all 4 directions.""");
        System.out.print("\nFront: ");
        float Front = sc.nextFloat(); // User input where distance is entered from Front
        System.out.print("\nBack: ");
        float Back = sc.nextFloat(); // User input where distance is entered from Back
        System.out.print("\nLeft: ");
        float Left = sc.nextFloat(); // User input where distance is entered from Left
        System.out.print("\nRight: ");
        float Right = sc.nextFloat(); // User input where distance is entered from Right

        SD.Direction(Front, Back, Left, Right); //Direction method called on the SD subclass, SD subclass inheriting the method from Dynamic Distancing class
        SD.Contact_Status(Front, Back, Left, Right); //Contact_Status method called on the SD subclass, SD subclass inheriting the method from Dynamic Distancing class
        SD.Contact_Status(SD.FindShortestDistance(Front, Back, Left, Right)); //Contact Status method called on the SD subclass, this is method overriding
        SD.ClinicAppointment(SD.FindShortestDistance(Front, Back, Left, Right)); //Clinic Appointment method called on SD subclass
    } else {
        System.out.println("\nThe capacity of the area is full. You are not allowed to enter the area. You need to wait " + Choice3.SpotPermittedAverageTime + " minutes."); //User is asked to wait SpotP
        System.out.print("\nDo you want to wait? (y/n): ");
        String waitchoice = sc.next();
        if (waitchoice.equals("y") || waitchoice.equals("Y")) { //Check user input if it is equal to 'y' or 'yes'
            System.out.println("\n" + Choice3.SpotPermittedAverageTime + " minutes has passed, you are now permitted to enter."); //Assume that the SpotPermittedAverageTime has passed
            System.out.println("Please enter the distance you are keeping from people around you from all 4 directions.");

            System.out.print("\nFront: ");
            float Front = sc.nextFloat(); // User input where distance is entered from Front
            System.out.print("\nBack: ");
            float Back = sc.nextFloat(); // User input where distance is entered from Back
            System.out.print("\nLeft: ");
            float Left = sc.nextFloat(); // User input where distance is entered from Left
            System.out.print("\nRight: ");
            float Right = sc.nextFloat(); // User input where distance is entered from Right

            SD.Direction(Front, Back, Left, Right); //Direction method called on the SD subclass, SD subclass inheriting the method from Dynamic Distancing class
            SD.Contact_Status(Front, Back, Left, Right); //Contact_Status method called on the SD subclass, SD subclass inheriting the method from Dynamic Distancing class
            SD.Contact_Status(SD.FindShortestDistance(Front, Back, Left, Right)); //Contact Status method called on the SD subclass, this is method overriding
            SD.ClinicAppointment(SD.FindShortestDistance(Front, Back, Left, Right)); //Clinic Appointment method called on SD subclass
        }
    }
```

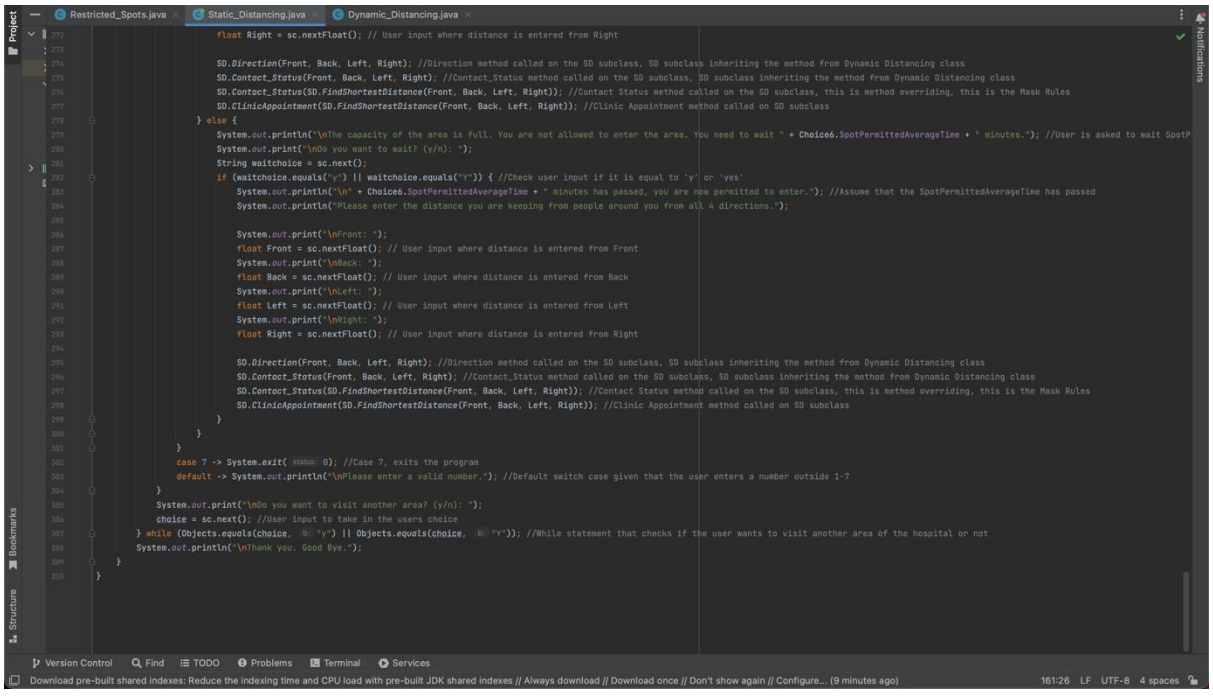


```
Restricted_Spots.java | Static_Distancing.java | Dynamic_Distancing.java

case 4 -> {
    Restricted_Spots.Area4 Choice4 = new Restricted_Spots.Area4();
    Choice4.Display();

    if ((Choice4.permission(Choice4.NumberOfPeople, Choice4.SpotMaximumCapacity)) == 1) { //Permission is called on Choice1 object to check if maximum capacity of area has been reached or not
        System.out.println("""

                You are allowed to enter the area.
                Please enter the distance you are keeping from people around you from all 4 directions.""");
        System.out.print("\nFront: ");
        float Front = sc.nextFloat(); // User input where distance is entered from Front
        System.out.print("\nBack: ");
        float Back = sc.nextFloat(); // User input where distance is entered from Back
        System.out.print("\nLeft: ");
        float Left = sc.nextFloat(); // User input where distance is entered from Left
        System.out.print("\nRight: ");
        float Right = sc.nextFloat(); // User input where distance is entered from Right

        SD.Direction(Front, Back, Left, Right); //Direction method called on the SD subclass, SD subclass inheriting the method from Dynamic Distancing class
        SD.Contact_Status(Front, Back, Left, Right); //Contact_Status method called on the SD subclass, SD subclass inheriting the method from Dynamic Distancing class
        SD.Contact_Status(SD.FindShortestDistance(Front, Back, Left, Right)); //Contact Status method called on the SD subclass, this is method overriding
        SD.ClinicAppointment(SD.FindShortestDistance(Front, Back, Left, Right)); //Clinic Appointment method called on SD subclass
    } else {
        System.out.println("\nThe capacity of the area is full. You are not allowed to enter the area. You need to wait " + Choice4.SpotPermittedAverageTime + " minutes."); //User is asked to wait SpotP
        System.out.print("\nDo you want to wait? (y/n): ");
        String waitchoice = sc.next();
        if (waitchoice.equals("y") || waitchoice.equals("Y")) { //Check user input if it is equal to 'y' or 'yes'
            System.out.println("\n" + Choice4.SpotPermittedAverageTime + " minutes has passed, you are now permitted to enter."); //Assume that the SpotPermittedAverageTime has passed
            System.out.println("Please enter the distance you are keeping from people around you from all 4 directions.");

            System.out.print("\nFront: ");
            float Front = sc.nextFloat(); // User input where distance is entered from Front
            System.out.print("\nBack: ");
            float Back = sc.nextFloat(); // User input where distance is entered from Back
            System.out.print("\nLeft: ");
            float Left = sc.nextFloat(); // User input where distance is entered from Left
            System.out.print("\nRight: ");
            float Right = sc.nextFloat(); // User input where distance is entered from Right

            SD.Direction(Front, Back, Left, Right); //Direction method called on the SD subclass, SD subclass inheriting the method from Dynamic Distancing class
            SD.Contact_Status(Front, Back, Left, Right); //Contact_Status method called on the SD subclass, SD subclass inheriting the method from Dynamic Distancing class
            SD.Contact_Status(SD.FindShortestDistance(Front, Back, Left, Right)); //Contact Status method called on the SD subclass, this is method overriding
            SD.ClinicAppointment(SD.FindShortestDistance(Front, Back, Left, Right)); //Clinic Appointment method called on SD subclass
        }

Version Control | Find | TODO | Problems | Terminal | Services
Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built JDK shared indexes // Always download // Download once // Don't show again // Configure... (8 minutes ago)
161:26  LF  UTF-8  4 spaces
```



```
Restricted_Spots.java | Static_Distancing.java | Dynamic_Distancing.java

case 5 -> {
    Restricted_Spots.Area5 Choice5 = new Restricted_Spots.Area5();
    Choice5.Display();

    if ((Choice5.permission(Choice5.NumberOfPeople, Choice5.SpotMaximumCapacity)) == 1) { //Permission is called on Choice1 object to check if maximum capacity of area has been reached or not
        System.out.println("""

                You are allowed to enter the area.
                Please enter the distance you are keeping from people around you from all 4 directions.""");
        System.out.print("\nFront: ");
        float Front = sc.nextFloat(); // User input where distance is entered from Front
        System.out.print("\nBack: ");
        float Back = sc.nextFloat(); // User input where distance is entered from Back
        System.out.print("\nLeft: ");
        float Left = sc.nextFloat(); // User input where distance is entered from Left
        System.out.print("\nRight: ");
        float Right = sc.nextFloat(); // User input where distance is entered from Right

        SD.Direction(Front, Back, Left, Right); //Direction method called on the SD subclass, SD subclass inheriting the method from Dynamic Distancing class
        SD.Contact_Status(Front, Back, Left, Right); //Contact_Status method called on the SD subclass, SD subclass inheriting the method from Dynamic Distancing class
        SD.Contact_Status(SD.FindShortestDistance(Front, Back, Left, Right)); //Contact Status method called on the SD subclass, this is method overriding
        SD.ClinicAppointment(SD.FindShortestDistance(Front, Back, Left, Right)); //Clinic Appointment method called on SD subclass
    } else {
        System.out.println("\nThe capacity of the area is full. You are not allowed to enter the area. You need to wait " + Choice5.SpotPermittedAverageTime + " minutes."); //User is asked to wait SpotP
        System.out.print("\nDo you want to wait? (y/n): ");
        String waitchoice = sc.next();
        if (waitchoice.equals("y") || waitchoice.equals("Y")) { //Check user input if it is equal to 'y' or 'yes'
            System.out.println("\n" + Choice5.SpotPermittedAverageTime + " minutes has passed, you are now permitted to enter."); //Assume that the SpotPermittedAverageTime has passed
            System.out.println("Please enter the distance you are keeping from people around you from all 4 directions.");

            System.out.print("\nFront: ");
            float Front = sc.nextFloat(); // User input where distance is entered from Front
            System.out.print("\nBack: ");
            float Back = sc.nextFloat(); // User input where distance is entered from Back
            System.out.print("\nLeft: ");
            float Left = sc.nextFloat(); // User input where distance is entered from Left
            System.out.print("\nRight: ");
            float Right = sc.nextFloat(); // User input where distance is entered from Right

            SD.Direction(Front, Back, Left, Right); //Direction method called on the SD subclass, SD subclass inheriting the method from Dynamic Distancing class
            SD.Contact_Status(Front, Back, Left, Right); //Contact_Status method called on the SD subclass, SD subclass inheriting the method from Dynamic Distancing class
            SD.Contact_Status(SD.FindShortestDistance(Front, Back, Left, Right)); //Contact Status method called on the SD subclass, this is method overriding, this is the Mask Rules
            SD.ClinicAppointment(SD.FindShortestDistance(Front, Back, Left, Right)); //Clinic Appointment method called on SD subclass
        }

Version Control | Find | TODO | Problems | Terminal | Services
Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built JDK shared indexes // Always download // Download once // Don't show again // Configure... (9 minutes ago)
161:26  LF  UTF-8  4 spaces
```


Restricted_Spots.java
Static_Distancing.java
Dynamic_Distancing.java
case 6 -> {
Restricted_Spots.Area6 Choice6 = new Restricted_Spots.Area6();
Choice6.Display();
if ((Choice6.permission(Choice6.NumberOfPeople, Choice6.SpotMaximumCapacity)) == 1) { //Permission is called on Choice1 object to check if maximum capacity of area has been reached or not
System.out.println("""
You are allowed to enter the area.
Please enter the distance you are keeping from people around you from all 4 directions.""");
System.out.print("\nFront: ");
float Front = sc.nextFloat(); // User input where distance is entered from Front
System.out.print("\nBack: ");
float Back = sc.nextFloat(); // User input where distance is entered from Back
System.out.print("\nLeft: ");
float Left = sc.nextFloat(); // User input where distance is entered from Left
System.out.print("\nRight: ");
float Right = sc.nextFloat(); // User input where distance is entered from Right
SD.Direction(Front, Back, Left, Right); //Direction method called on the SD subclass, SD subclass inheriting the method from Dynamic Distancing class
SD.Contact_Status(Front, Back, Left, Right); //Contact_Status method called on the SD subclass, SD subclass inheriting the method from Dynamic Distancing class
SD.Contact_Status(SD.FindShortestDistance(Front, Back, Left, Right)); //Contact Status method called on the SD subclass, this is method overriding, this is the Mask Rules
SD.ClinicAppointment(SD.FindShortestDistance(Front, Back, Left, Right)); //Clinic Appointment method called on SD subclass
} else {
System.out.println("\nThe capacity of the area is full. You are not allowed to enter the area. You need to wait " + Choice6.SpotPermittedAverageTime + " minutes."); //User is asked to wait SpotP
System.out.print("\nDo you want to wait? (y/n): ");
String waitchoice = sc.next();
if (waitchoice.equals("y") || waitchoice.equals("Y")) { //Check user input if it is equal to 'y' or 'yes'
System.out.println("\n" + Choice6.SpotPermittedAverageTime + " minutes has passed, you are now permitted to enter."); //Assume that the SpotPermittedAverageTime has passed
System.out.println("Please enter the distance you are keeping from people around you from all 4 directions.");
System.out.print("\nFront: ");
float Front = sc.nextFloat(); // User input where distance is entered from Front
System.out.print("\nBack: ");
float Back = sc.nextFloat(); // User input where distance is entered from Back
System.out.print("\nLeft: ");
float Left = sc.nextFloat(); // User input where distance is entered from Left
System.out.print("\nRight: ");
float Right = sc.nextFloat(); // User input where distance is entered from Right
SD.Direction(Front, Back, Left, Right); //Direction method called on the SD subclass, SD subclass inheriting the method from Dynamic Distancing class
SD.Contact_Status(Front, Back, Left, Right); //Contact_Status method called on the SD subclass, SD subclass inheriting the method from Dynamic Distancing class
SD.Contact_Status(SD.FindShortestDistance(Front, Back, Left, Right)); //Contact Status method called on the SD subclass, this is method overriding, this is the Mask Rules
SD.ClinicAppointment(SD.FindShortestDistance(Front, Back, Left, Right)); //Clinic Appointment method called on SD subclass
}
Version Control
Find
TODO
Problems
Terminal
Services
Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built JDK shared indexes // Always download // Download once // Don't show again // Configure... (9 minutes ago)
161:26
LF
UTF-8
4 spaces


Restricted_Spots.java
Static_Distancing.java
Dynamic_Distancing.java
float Right = sc.nextFloat(); // User input where distance is entered from Right
SD.Direction(Front, Back, Left, Right); //Direction method called on the SD subclass, SD subclass inheriting the method from Dynamic Distancing class
SD.Contact_Status(Front, Back, Left, Right); //Contact_Status method called on the SD subclass, SD subclass inheriting the method from Dynamic Distancing class
SD.Contact_Status(SD.FindShortestDistance(Front, Back, Left, Right)); //Contact Status method called on the SD subclass, this is method overriding, this is the Mask Rules
SD.ClinicAppointment(SD.FindShortestDistance(Front, Back, Left, Right)); //Clinic Appointment method called on SD subclass
} else {
System.out.println("\nThe capacity of the area is full. You are not allowed to enter the area. You need to wait " + Choice6.SpotPermittedAverageTime + " minutes."); //User is asked to wait SpotP
System.out.print("\nDo you want to wait? (y/n): ");
String waitchoice = sc.next();
if (waitchoice.equals("y") || waitchoice.equals("Y")) { //Check user input if it is equal to 'y' or 'yes'
System.out.println("\n" + Choice6.SpotPermittedAverageTime + " minutes has passed, you are now permitted to enter."); //Assume that the SpotPermittedAverageTime has passed
System.out.println("Please enter the distance you are keeping from people around you from all 4 directions.");
System.out.print("\nFront: ");
float Front = sc.nextFloat(); // User input where distance is entered from Front
System.out.print("\nBack: ");
float Back = sc.nextFloat(); // User input where distance is entered from Back
System.out.print("\nLeft: ");
float Left = sc.nextFloat(); // User input where distance is entered from Left
System.out.print("\nRight: ");
float Right = sc.nextFloat(); // User input where distance is entered from Right
SD.Direction(Front, Back, Left, Right); //Direction method called on the SD subclass, SD subclass inheriting the method from Dynamic Distancing class
SD.Contact_Status(Front, Back, Left, Right); //Contact_Status method called on the SD subclass, SD subclass inheriting the method from Dynamic Distancing class
SD.Contact_Status(SD.FindShortestDistance(Front, Back, Left, Right)); //Contact Status method called on the SD subclass, this is method overriding, this is the Mask Rules
SD.ClinicAppointment(SD.FindShortestDistance(Front, Back, Left, Right)); //Clinic Appointment method called on SD subclass
}
}
}
case 7 -> System.exit(status: 0); //Case 7, exits the program
default -> System.out.println("\nPlease enter a valid number."); //Default switch case given that the user enters a number outside 1-7
}
System.out.print("\nDo you want to visit another area? (y/n): ");
choice = sc.next(); //User input to take in the users choice
} while (Objects.equals(choice, b: "y") || Objects.equals(choice, b: "Y")); //While statement that checks if the user wants to visit another area of the hospital or not
System.out.println("\nThank you. Good Bye.");
}
}
Version Control
Find
TODO
Problems
Terminal
Services
Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built JDK shared indexes // Always download // Download once // Don't show again // Configure... (9 minutes ago)
161:26
LF
UTF-8
4 spaces