ADVANCED PROGRAMMING

LABORATORY PRACTICE CAL3

MAY 2021

Course 2020/2021 – Ordinary call

Computer Science Degree – 09123306B – García Azagra, Óscar

Computer Science Degree – x8059334L– Tamas, Darius Dumitras

INDEX

A HIGH LEVEL ANALYSIS………………………………………………………………………………………………3

GENERAL DESIGN OF THE SYSTEM AND SYNCHRONIZATION TOOLS USED…………………..4

MAIN CLASSES…………………………………………………………………………………………………………….5

CLASS DIAGRAM………………………………………………………………………………………………………..10

PROBLEMS FACED, SOLVED AND UNSOLVED…………………………………………….……………….11

ANNEX……………………………………………………………………………………………,…………………………13

HIGH-LEVEL ANALYSIS

This program simulates the behavior of a hospital whose only function is to vaccinate people.

This hospital has different rooms to accomplish that:

* Reception: In this room, there is an auxiliary worker working as receptionist. The patients will enter, and, after checking with the receptionist whether they have an appointment or not:
  + If they have an appointment, they will wait in the restroom for the auxiliary to tell them an available desk so they can get the vaccine
  + If they don’t have an appointment, they will leave the hospital
* Vaccination Room: This room has 10 desks for people to be vaccinated. There are 10 workers that are in charge of vaccinating the patients, and they will take from 3 to 5 seconds to be vaccinated, as long as there are available vaccines. After that they will go to the first observation desk that is available
* Observation Room: This room has 20 desks for people. The vaccinated patients will wait for 10 seconds to confirm that the vaccine doesn’t have any side effect on them
  + If they don’t have any reaction, they will leave the hospital
  + If they have a reaction, a worker will come to their desk to check that side effect, and after helping with it (taking from 2 to 5 seconds) the patient will leave

As mentioned, there are 3 groups of people with different functionalities here:

* Patients: They have 2 possible special features
  + Having an appointment: 99% of them have an appointment
  + Having a reaction to the vaccine: 5% of them have reactions to the vaccine
* Healthcare workers: They are the ones in charge of vaccinating and checking up the patients. After vaccinating 15 people they have a break of 5-8 seconds until they have to come back. There are some special cases:
  + Helping patients with reaction to the vaccine is their priority, so, if they are working but there’s no one on their desk in the vaccination room, or they are resting, they will move to the observation room to help
* Auxiliary workers: There are 2 really different types:
  + Receptionist: They are in charge of checking if the patients have an appointment or not, and telling them which desk in the vaccination room they should go in. They take breaks every 10 patients
  + Vaccines preparer: They prepare the vaccines doses for the healthcare workers to apply. They take breaks every 20 vaccines

GENERAL DESIGN OF THE SYSTEM AND SYNCHRONIZATION TOOLS USED

After this overview of the program, we can focus on a more programming-oriented view of the problem:

All of the points mentioned in the high-level analysis will be represented by classes:

* Hospital: This class will represent the combination of all the different rooms that are part of the hospital. Its main purpose is to allow both communication and synchronization between these rooms and the users (workers and patients)
* Reception: This class will represent the first room in the hospital. In this room, an auxiliary worker will be in charge of coordination of the patients, indicating them the desk they need to go in order to be vaccinated.
  + In order to do that, first they have to check if the patient has an appointment. In negative case, the patient’s thread life cycle will end
  + In positive case, the auxiliary worker will move this patient from an ArrayList to another. When they first come, they are stored in the WaitingQueue, and after being confirmed to have an appointment they will be moved to the EnteringQueue, where they wait to be assigned a desk
  + Once they are assigned a desk, the desk’s ID is returned
* Vaccination Room: This class will represent a room that consists of 10 desks, where patients are vaccinated. In order to be active, they need a Healthcare worker in charge of them. After waiting a random time between 3 and 5 seconds, the patient is vaccinated, and they are lead to the observation room
* Observation Room: This class will represent a room that consists of 20 desks where patients wait 10 seconds. During this time, they are checked if they have a reaction to the vaccine. If that’s the case, a healthcare worker will be summoned there (either one already working or one resting). After that, the patient’s life cycle ends
* Patient: This class will be the one that uses all the hospital. It will call methods from the hospital that are in charge of the rooms to move between them in order to be vaccinated.
* Healthcare workers: This class represents a thread that will start asleep. After that rest, they will move to the first available desk in the vaccination room, or to the observation room if there are patients with complications. After 15 vaccinations (they have to be synchronized with the patients), they will rest.
* Auxiliar Worker: This class has an if, differentiating the 2 different types
  + Receptionist (id = 1): They are in charge of checking if the patients have an appointment or not and telling them which desk in the vaccination room they should go in, synchronizing them with the healthcare workers. They take breaks every 10 patients
* Vaccines preparer (id = 2): They prepare the vaccines doses for the healthcare workers to apply. They take breaks every 20 vaccines

Synchronization Tools Used:

Semaphores: The main use for semaphores in this program are for allowing mutual exclusion on shared variables so we can avoid corruption. There are 2 main types of semaphores

* Binary: These semaphores have only 2 states, blocked and not blocked. They can be used as substitutes of locks
* Non-binary: These semaphores are used in this program to control the number of times that the vaccination room and observation rooms are entered and checked

Locks: They are used to allow mutual exclusion and avoid race conditions.

Monitors: They are used to check conditions and making threads sleep or wake using that conditions

MAIN CLASSES

Hospital:

ATTRIBUTES

* Reception reception
* VaccRoom vaccRoom
* ObservationRoom obsRoom
* Atomic Integer capacity
* HashMap<Integer, Patient> patients
* HashMap<Integer, HcareWorker> hcareWorkers
* ArrayList<HcareWorker> restroom
* Semaphore semEnterVacc
* Semaphore semEnterObs
* Semaphore semPatients
* Semaphore semException
* MainWindow window;
* CustomLogger clogger

METHODS

* enterHospital(): This method increases the number of patients in the hospital, add a patient to the patients HashMap and inserts them into the waiting queue
* enterReception(): This method checks the first available desk and returns it
* enterVaccRoom(): This method places the patient into their designed desk, vaccinates them and checks the available rooms that are on the observation room, and returns the ID of the first one
* enterObservationRoom(): This method places the patient into their designed desk, checks if they have any problem with the vaccine (in case they have, it will be treated). After that, the patient will leave the hospital (leaving as well the patients HashMap, and decreasing the number of patients)
* addPatient(): Adds a patient into the HashMap
* removePatient()Removes a patient from the HashMap
* addWorker(): Adds a worker into the HashMap
* restRoomToString(): Method that returns the information of the restroom
* startWindow(): Method that runs the thread that updates the window

RECEPTION

ATTRIBUTES

* ArrayList<Patient> waitingQ:
* AuxWorker auxWorker1
* ArrayList<Patient> enteringQ
* Semaphore waitingSemaphore
* Semaphore enteringSemaphore
* Hospital hospital
* Lock wLock
* Lock eLock

METHODS

* enterWaitingQueue(): Adds the patient to the waiting queue and makes them to wait to be awake
* exitWaitingQueue(): Removes the patient from the waiting queue
* enterEnteringQueue(): Enters the patient to the entering queue
* exitEnteringQueue(): Removes the patient from the entering queue and awakes them

VACCINATION ROOM

ATTRIBUTES

* AtomicInteger vaccines
* ArrayList<Desk> desks
* AuxWorker aWorker
* Lock desksLock
* Condition availableDesk
* Lock vaccLock
* Condition vaccinating
* Semaphore semDesks

METHODS

* sitPatient(): Sits the patient in their designed desk
* exitPatient() Removes the patient from the desk
* getAvailableDesk(): Gets the ID of the first available desk
* vaccinate(): Method used to synchronize the wait of both worker and patient

OBSERVATION ROOM

ATTRIBUTES

* ArrayList<Desk> desks
* Lock desksLock
* Condition availableDesk

METHODS

* sitPatient(): Sits the patient in their designed desk
* exitPatient() Removes the patient from the desk
* getAvailableDesk(): Gets the ID of the first available desk
* checkComplications(): This method checks if there are any infected patients on the observation room. If there are, their desk’s ID will be stored in an ArrayList and returned

PATIENT

ATTRIBUTES

* int pid
* int randomChance
* Hospital hospital
* int timeToGetDesk
* int timeToVaccine
* int timeWithComplications
* boolean waitToGetVaccinated

METHODS

* run()

HEALTHCARE WORKER

ATTRIBUTES

* int hid
* int pVaccinated
* int iDDeskVacc
* int iDDeskObs
* int timeToVaccine
* int timeToRest
* int counter
* int maximum
* boolean beenAwaken
* boolean working
* Lock lock
* Condition noWorkToDo
* Hospital hospital
* boolean isVaccinating

METHODS

* run()

AUXILIARY WORKER

ATTRIBUTES

* int aid
* int counter
* int totalCount
* Hospital hospital
* boolean isResting
* Semaphore semCounter

METHODS

* run()
* availableDesk(): Checks that the patient has an appointment and returns their designed desk

STRING MANAGER

ATTRIBUTES

* Hospital hospital
* MainWindow window

METHODS

* run()
* textSetter(): Using the data gathered from the other methods, it updates all the text fields of the window

CUSTOM LOGGER

ATTRIBUTES

String path

BufferedWriter bw

METHODS

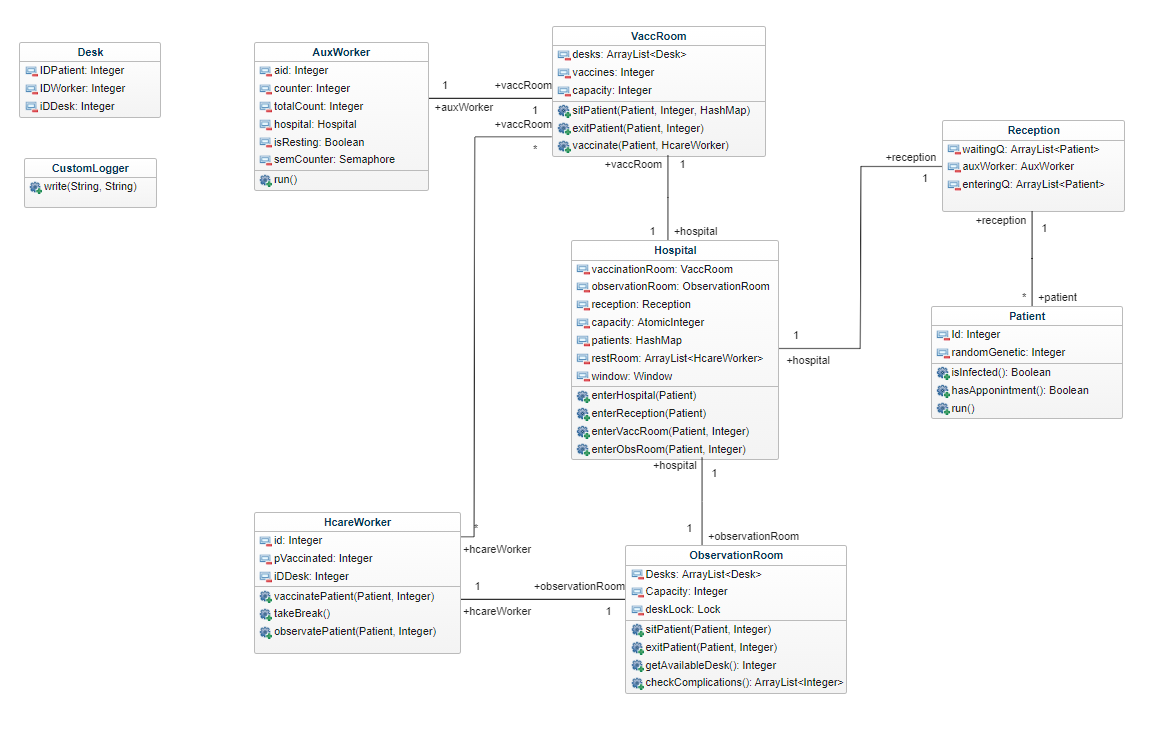
* write()

DESK

ATTRIBUTES

* int iDDesk
* int iDPatient: There are three possible numbers
  + -1: The desk is free
  + 0: The desk is reserved, that is, there is a patient assigned to it, but they haven’t arrived yet
  + n>0: The ID of the patient in the desk
* int iDWorker
  + -1: The desk is free
  + 0: The desk is reserved, that is, there is a worker assigned to it, but they haven’t arrived yet
  + n>0: The ID of the worker in the desk

CLASS DIAGRAM



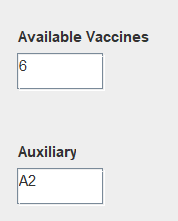
PROBLEMS FACED, SOLVED AND UNSOLVED.

A list of the development of the project can be found on the following [GitHub](https://github.com/Yaxeb/ProgramacionAvanzada):

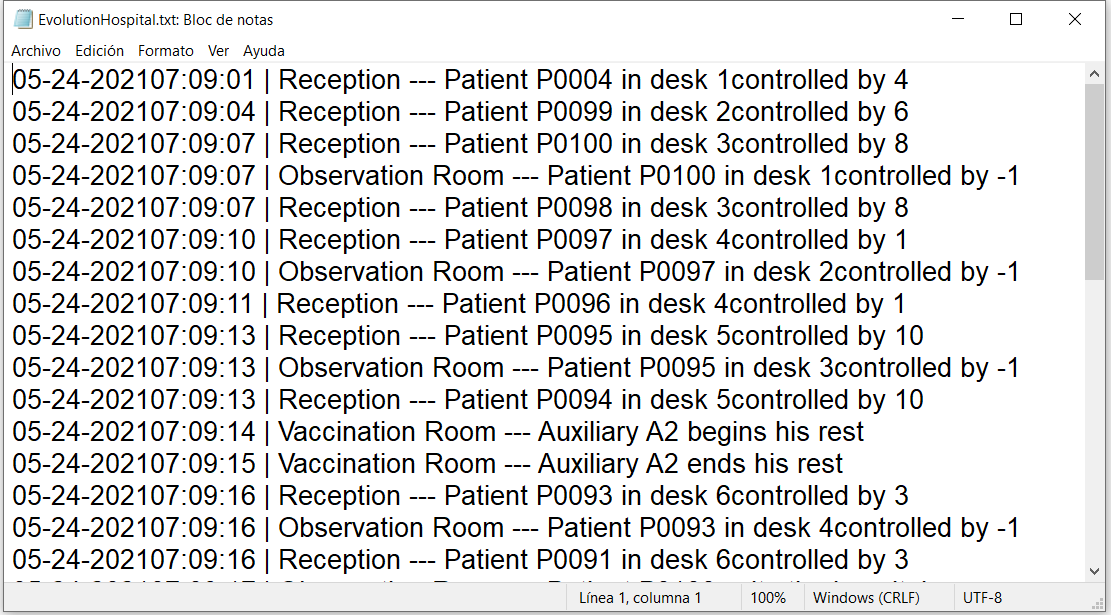
Since the code is not fully finished this section is written to be useful for a clearer understanding of the problems solved, since the unsolved ones (which will also be mentioned) are easily spotted by executing the code.

SOLVED PROBLEMS:

* Registration of the patients in the hospital, discarding the ones without an appointment.
* Creation of the vaccines.

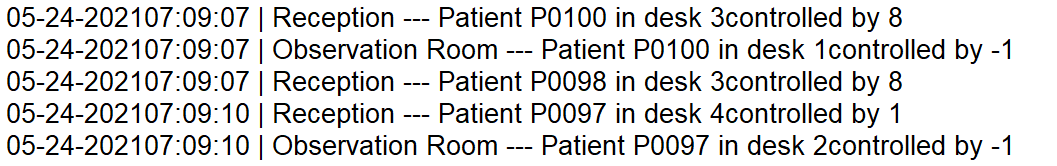


* Thread interruptions from its normal behavior (sleeps)
* Log implementation: Logs are stored inside the folder “Logs”



* Movement of the patient through the rooms.

Checking the logs, timestamps can be seen on the left part, patients move through rooms



UNSOLVED PROBLEMS:

On the following [commit](https://github.com/Yaxeb/ProgramacionAvanzada/commit/576a17fac2facbb7e65d5691dd065c6d646a4fc4): We had managed to fix every single problem besides some threads not properly showing while being in the vaccination room.

The next commit, we tested the behavior with forced users which could have reactions while being vaccinated, after that was fixed, we changed the code to create 100 threads. And let them run freely, shortly after we noticed the synchronization was not done properly, we changed the synchronization when vaccinating to be the method “vaccinate(Patient, HcareWorker)”.

That one was our last progress and when we got stuck, without active waiting (checking on if the worker is going to vaccinate the patient constantly) or changing non-viable lines of code we couldn’t figure it out in time, and it is our last progress.

ANNEX

package pecl;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.concurrent.Semaphore;

import java.util.concurrent.atomic.AtomicInteger;

import java.util.concurrent.locks.Condition;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

public class VaccRoom {

private final AtomicInteger vaccines;

private ArrayList<Desk> desks;

private AuxWorker aWorker;

private Lock desksLock;

private Condition availableDesk;

private Lock vaccLock;

private Condition vaccinating;

private Semaphore semDesks = new Semaphore(10);

/\*\*

\* This method initializes the object

\* @param hospital The hospital this room belongs to

\*/

public VaccRoom() {

this.vaccines = new AtomicInteger();

this.desks = new ArrayList<>(10);

for(int i = 0; i<10; i++)

{

this.desks.add(new Desk(i+1));

}

this.desksLock = new ReentrantLock();

this.availableDesk = desksLock.newCondition();

this.vaccLock = new ReentrantLock();

this.vaccinating = vaccLock.newCondition();

}

/\*\*

\* This method locates a patient into the desk with the ID given

\* It uses a lock to avoid data corruption and race conditions when

\* accessing the desks attribute

\* @param patient The patient who's going to sit in the desk

\* @param iDDesk The ID of the desk

\*/

public synchronized void sitPatient(Patient patient, int iDDesk, HashMap<Integer ,HcareWorker> workers){

Desk d = desks.get(iDDesk-1);

d.setPatient(patient.getPid());

desks.set(iDDesk-1, d);

workers.get(d.getWorker()).signalNoWorkToDo();

}

/\*\*

\* This method prepares a patient to leave the observation room

\* It uses a lock to avoid data corruption and race conditions when

\* accessing the desks attribute

\* It sets the Patient's id in the desk as -1, indicating that this desk

\* has no patients.

\* It also signals that a desk could be available to the method

\* getAvailableDesk()

\* @param patient The patient who's going to sit in the desk

\* @param iDDesk The ID of the desk

\*/

public synchronized void exitPatient(Patient patient, int iDDesk){

Desk d = desks.get(iDDesk-1);

d.setPatient(-1);

desks.set(iDDesk-1, d);

semDesks.release();

}

/\*\*

\* This method looks at all the desks sequentially, and, in case there is an

\* available one, it returns its ID.

\* It uses a lock to avoid data corruption and race conditions when

\* accessing the desks attribute.

\* It checks whether the room doesn't have a patient, but it has a worker

\*(by checking that Patient's ID is -1 and Worker's ID is not -1)

\* and, if it is, it will return that desk's ID

\* Otherwise, the method will be stopped using await() until it is signaled

\* by the method exitPatient() that there is a possible new free desk

\* @return The ID of the first available desk

\*/

public int getAvailableDesk(){

int i = 0;

try{

semDesks.acquire();

boolean found = false;

while(!found)

{

if (i >= desks.size() -1 )

{

i = 0;

}

Desk d = desks.get(i);

if(d.getPatient() == -1 && d.getWorker() != -1)

{//the desk has a worker and no patients

found = true;

d.setPatient(0); //We reserve the desk

}

i++; //we do it regardless, because the desk's ID is id+1, so even if

// we have found the first available desk, we still need to add 1

}

}catch(Exception e){}

return i;

}

public void vaccinate(Patient patient, HcareWorker worker)

{

try

{

vaccLock.lock();

while(!worker.isVaccinating())

{

try

{

vaccinating.await();

} catch (InterruptedException ex) {}

}

}

catch(Exception e){}

finally

{

vaccLock.unlock();

}

}

public void notifyVaccine(Patient patient)

{

try

{

vaccLock.lock();

vaccinating.signal();

}catch(Exception e){}

finally

{

vaccLock.unlock();

}

}

/\*\*

\* This method returns the number of available vaccines

\* @return The number of available vaccines

\*/

public int getVaccines(){

return vaccines.get();

}

/\*\*

\* This method increments the number of available vaccines by 1

\*/

public void createVaccine(){

vaccines.set(vaccines.addAndGet(1));

}

/\*\*

\* This method decrements the number of available vaccines by 1

\*/

public void takeVaccine(){

vaccines.set(vaccines.decrementAndGet());

}

/\*\*

\* This method returns all the desks from the Vaccination Room

\* @return An ArrayList containing all desks from the Vaccination Room

\*/

public synchronized ArrayList<Desk> getDesks(){

return this.desks;

}

public void setDesks(ArrayList<Desk> desks) {

ArrayList<Desk> d = new ArrayList<>();

try{

desksLock.lock();

this.desks = desks;

}catch(Exception e){}

finally{

desksLock.unlock();

}

}

public AuxWorker getAuxWorker() {

return aWorker;

}

public void setAuxWorker(AuxWorker aWorker) {

this.aWorker = aWorker;

}

}

package pecl;

import hospitalInterface.\*;

import java.util.ArrayList;

public class StringManager extends Thread{

private Hospital hospital;

private MainWindow window;

/\*\*

\* Constructor method for StringManager

\* @param hospital The hospital which information is displayed

\* @param window The window to display the information

\*/

public StringManager(Hospital hospital, MainWindow window) {

this.hospital = hospital;

this.window = window;

}

@Override

public void run(){

while(true){

textSetter();

}

}

/\*\*

\* Method that updates the interface. It sequentially updates all parameters

\*/

public void textSetter(){

//Reception

window.getReception().setText(hospital.getReception().allPatientsToString()); // Reception

//window.getReception().setText(""+hospital.getReception().getEnteringQueue().size());

if (hospital.getReception().getWaitingQueue().size() > 0){

window.getReceptionPatient().setText("P" + String.format("%04d", hospital.getReception().getWaitingQueue().get(0).getPid())); //Patient

}

if (hospital.getReception().getAuxWorker().isResting()) //Auxiliary

{

window.getReceptionAux().setText("");

}

else

{

window.getReceptionAux().setText("A"+hospital.getReception().getAuxWorker().getAid());

}

//Restroom

window.getRestRoom().setText(hospital.restRoomToString());

//Vaccination Room

ArrayList<javax.swing.JTextArea> vDesks = window.getVDesks();

for(int i = 0;i<vDesks.size();i++)

{

vDesks.get(i).setText(hospital.getVaccRoom().getDesks().get(i).toString());

}

window.getVaccines().setText(""+hospital.getVaccRoom().getVaccines()); //Vaccines

if (hospital.getVaccRoom().getAuxWorker().isResting()) //Auxiliary

{

window.getVaccRoomAux().setText("");

}

else

{

window.getVaccRoomAux().setText("A"+hospital.getVaccRoom().getAuxWorker().getAid());

}

//Observation Room

ArrayList<javax.swing.JTextArea> oDesks = window.getODesks();

for(int i = 0;i<oDesks.size();i++)

{

oDesks.get(i).setText(hospital.getObsRoom().getDesks().get(i).toString());

}

}

}

package pecl;

import java.util.ArrayList;

import java.util.concurrent.Semaphore;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

public class Reception {

private final ArrayList<Patient> waitingQ;

private AuxWorker auxWorker1;

private final ArrayList<Patient> enteringQ;

private Semaphore waitingSemaphore;

private Semaphore enteringSemaphore;

private Hospital hospital;

private Lock wLock = new ReentrantLock();

private Lock eLock = new ReentrantLock();

public Reception(){

this.waitingQ = new ArrayList<>();

this.enteringQ = new ArrayList<>();

this.waitingSemaphore = new Semaphore(1);

this.enteringSemaphore = new Semaphore(1);

}

/\*\*

\* Method that inserts a patient in a waiting queue. This queue represents

\* the patients waiting to be attended by the auxiliar to check if they have

\* an appointment.

\* @param patient The patient entering the hospital

\*/

public synchronized void enterWaitingQueue(Patient patient){

//waitingSemaphore.acquire();

waitingQ.add(patient);

try {

wait();

// System.out.println("EntraWaiting");

} catch (InterruptedException ex) { }

}

/\*\*

\* Method that removes a patient from the waiting queue. This queue represents

\* the patients waiting to be attended by the auxiliar to check if they have

\* an appointment.

\* @param patient The patient exiting the waiting queue

\*/

public synchronized void exitWaitingQueue(Patient patient){

waitingQ.remove(patient);

}

/\*\*

\* Method that inserts a patient in an entering queue. This queue represents

\* the patients waiting to go to the vaccination Room

\* @param patient The patient waiting to enter the vaccination room

\*/

public synchronized void enterEnteringQueue(Patient patient){

enteringQ.add(patient);

try {

wait();

} catch (InterruptedException ex) {}

}

/\*\*

\* Method that removes a patient from the entering queue. This queue represents

\* the patients waiting to go to the vaccination Room

\* @param patient The patient waiting to enter the vaccination room

\*/

public synchronized void exitEnteringQueue(Patient patient){

enteringQ.remove(patient);

}

public synchronized void getNextPatient(){

notify();

}

/\*\*

\* Returns the Entering queue. This queue represents

\* the patients waiting to go to the vaccination Room

\* @return The Entering Queue

\*/

public ArrayList<Patient> getEnteringQueue(){

return this.enteringQ;

}

/\*\*

\* Returns the Waiting queue. This queue represents

\* the patients waiting to be attended by the auxiliar to check if they have

\* an appointment.

\* @return The Waiting queue

\*/

public ArrayList<Patient> getWaitingQueue(){

return this.waitingQ;

}

public String allPatientsToString(){

ArrayList<Patient> all = new ArrayList<>();

String text = "";

all.addAll(getEnteringQueue());

all.addAll(getWaitingQueue());

for (Patient patient : all)

{

text+= "P"+String.format("%04d", patient.getPid()) + ", ";

}

return text;

}

/\*\*

\* Returns the auxiliary worker who is in charge of checking the appointments

\* @return The auxiliary worker who is in charge of checking the appointments

\*/

public AuxWorker getAuxWorker(){

return this.auxWorker1;

}

/\*\*

\* Sets the auxiliary worker who will be in charge of checking the appointments

\* @param auxWorker1 The auxiliary worker who will be in charge of checking the appointments

\*/

public void setAuxWorker(AuxWorker auxWorker1){

this.auxWorker1 = auxWorker1;

}

}

package pecl;

import java.util.logging.Level;

import java.util.logging.Logger;

public class Patient extends Thread{

private final int pid;

private final int randomChance;

private Hospital hospital;

private int timeToGetDesk;

private int timeToVaccine;

private int timeWithComplications;

private boolean waitToGetVaccinated;

// private CustomLogger clogger;

/\*\*

\* Method constructor for Patient

\* @param pid The patient's ID

\* @param hospital The hospital the patient is attending

\*/

public Patient(int pid, Hospital hospital) {

this.pid = pid;

this.randomChance = (int) (Math.random() \* 101);

this.timeWithComplications = 0;

this.hospital = hospital;

// this.clogger = hospital.getLogger();

}

@Override

public void run(){

//hospital.getLogger().log("test log");

hospital.enterHospital(this);

//clogger.log("Patient " + pid + " Entered the hospital. ");

int iDDesk = hospital.enterReception(this, hospital.getReception().getAuxWorker());

//clogger.log("Patient " + pid + " Passed through reception. ");

if (iDDesk != 0) {

int obsDesk = hospital.enterVaccRoom(this, iDDesk);

hospital.enterObservationRoom(this, obsDesk);

try

{

sleep(timeWithComplications);

}

catch (InterruptedException ex)

{

Logger.getLogger(Patient.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

public void setTimeToGetDesk(int time){

this.timeToGetDesk = time;

}

/\*\*

\* Sets the time it takes to vaccine

\* @param time The time it takes to vaccine

\*/

public void setTimeToVaccine(int time){

this.timeToVaccine = time;

}

/\*\*

\* Sets the time it takes to solve the complications

\* @param time Time it takes to solve the complications

\*/

public void setTimeWithComplications(int time){

this.timeWithComplications = time;

}

/\*\*

\* Returns the Patient's ID

\* @return Patient's ID

\*/

public int getPid() {

return pid;

}

/\*\*

\* Returns whether the Patient is Infected or not.

\* For that, when the Patient is created, a random number is generated

\* If the number is less or equal than 5 (5% chance) the patient will react to

\* the vaccine and would need to be treated

\* @return True if the Patient will have a reaction, False otherwise

\*/

public boolean isInfected() {

return randomChance <= 5;

}

/\*\*

\* Returns whether the Patient has an appointment or not.

\* For that, when the Patient is created, a random number is generated

\* If the number is different from 0 (99% chance) the patient will have

\* an appointment and would enter the entering queue. Otherwise it will

\* leave the hospital

\* @return True if the patient has an appointment, False otherwise

\*/

public boolean hasAppointment(){

return randomChance != 0;

}

}

package pecl;

import hospitalInterface.\*;

import java.util.ArrayList;

public class PECL {

public static void main(String[] args) {

// TODO code application logic here

MainWindow window = new MainWindow();

window.setVisible(true);

Reception reception = new Reception();

VaccRoom vaccRoom = new VaccRoom();

ObservationRoom obsRoom = new ObservationRoom();

Hospital hospital = new Hospital(reception, vaccRoom, obsRoom, window);

AuxWorker a1 = new AuxWorker(1,10,hospital);

hospital.getReception().setAuxWorker(a1);

hospital.startWindow();

a1.start();

AuxWorker a2 = new AuxWorker(2,20,hospital);

hospital.getVaccRoom().setAuxWorker(a2);

a2.start();

ArrayList<HcareWorker> l = new ArrayList<>();

for (int i = 1; i<= 10; i++)

{

HcareWorker worker = new HcareWorker(i, hospital);

hospital.addWorker(worker);

worker.start();

}

for(int i = 1; i <= 100; i++)

{

Patient patient = new Patient(i, hospital);

patient.start();

}

}

}

package pecl;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.concurrent.locks.Condition;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

public class ObservationRoom {

private ArrayList<Desk> desks;

private Lock desksLock;

private Condition availableDesk;

/\*\*

\* This method initializes the object

\* @param hospital The hospital this room belongs to

\*/

public ObservationRoom(){

desks = new ArrayList<>(20);

for(int i = 0; i<20; i++)

{

this.desks.add(new Desk(i+1));

}

this.desksLock = new ReentrantLock();

this.availableDesk = desksLock.newCondition();

}

/\*\*

\* This method locates a patient into the desk with the ID given

\* It uses a lock to avoid data corruption and race conditions when

\* accessing the desks attribute

\* @param patient The patient who's going to sit in the desk

\* @param iDDesk The ID of the desk

\*/

public synchronized void sitPatient(Patient patient, int iDDesk){

Desk d = desks.get(iDDesk-1);

d.setPatient(patient.getPid());

desks.set(iDDesk-1, d);

}

/\*\*

\* This method prepares a patient to leave the observation room

\* It uses a lock to avoid data corruption and race conditions when

\* accessing the desks attribute

\* It sets the Patient's id in the desk as -1, indicating that this desk

\* has no patients.

\* It also signals that a desk could be available to the method

\* getAvailableDesk()

\* @param patient The patient who's going to sit in the desk

\* @param iDDesk The ID of the desk

\*/

public void exitPatient(Patient patient, int iDDesk){

try{

desksLock.lock();

Desk d = desks.get(iDDesk-1);

d.setPatient(-1);

desks.set(iDDesk-1, d);

availableDesk.signal();

}catch(Exception e){}

finally{

desksLock.unlock();

}

}

/\*\*

\* This method looks at all the desks sequentially, and, in case there is an

\* available one, it returns its ID.

\* It uses a lock to avoid data corruption and race conditions when

\* accessing the desks attribute.

\* It checks whether the room is empty (checking both Patient's ID and

\* Worker's ID are -1) and, if it is, it will return that desk's ID

\* Otherwise, the method will be stopped using await() until it is signaled

\* by the method exitPatient() that there is a possible new free desk

\* @return The ID of the first available desk

\*/

public int getAvailableDesk(){

int i = 0;

try{

desksLock.lock();

boolean found = false;

while(!found)

{

i = 0;

while (i < desks.size() && !found)

{

Desk d = desks.get(i);

if(d.getPatient() == -1 && d.getWorker() == -1)

{//the desk has a worker and no patients

found = true;

}

i++;

}

if (!found){

try{

availableDesk.await();

}catch(Exception e){}

}

}

}catch(Exception e){}

finally

{

desksLock.unlock();

}

return i;

}

/\*\*

\* This method checks if there is any desk with a patient with complications

\* due to the vaccine

\* It uses the HashMap of patients to retrieve the Patient using the ID stored

\* in the desk, and while checking it, if it's infected with a mutation due to the

\* vaccine, it adds the ID of their desk to an ArrayList.After traversing all desks, the counter is returned

\* @param patients

\* @return An ArrayList with all the Desks' ID that have a patient with complications

\*/

public ArrayList<Integer> checkComplications(HashMap<Integer, Patient> patients){

ArrayList<Integer> desksWithComplications = new ArrayList<>();

try{

desksLock.lock();

for (int i = 0; i<desks.size(); i++)

{

if (patients.get(desks.get(i).getPatient()).isInfected()){

desksWithComplications.add(i+1);

}

}

}catch(Exception e){}

finally{

desksLock.unlock();

}

return desksWithComplications;

}

/\*\*

\* Returns the ArrayList of desks which are located in the Observation Room

\* @return ArrayList<Desk>

\*/

public ArrayList<Desk> getDesks(){

return this.desks;

}

}

package pecl;

import hospitalInterface.MainWindow;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.concurrent.Semaphore;

import java.util.concurrent.atomic.AtomicInteger;

import java.util.logging.Level;

import java.util.logging.Logger;

public class Hospital {

private Reception reception;

private VaccRoom vaccRoom;

private ObservationRoom obsRoom;

private AtomicInteger capacity;

private HashMap<Integer, Patient> patients;

private HashMap<Integer, HcareWorker> hcareWorkers;

private ArrayList<HcareWorker> restRoom;

private Semaphore semEnterVacc = new Semaphore(10);

private Semaphore semEnterObs = new Semaphore(20);

private Semaphore semPatients = new Semaphore(1);

private Semaphore semException = new Semaphore(1);

private MainWindow window;

private CustomLogger clogger;

public Hospital(Reception reception, VaccRoom vaccRoom, ObservationRoom obsRoom, MainWindow window) {

this.reception = reception;

this.vaccRoom = vaccRoom;

this.obsRoom = obsRoom;

this.capacity = new AtomicInteger();

this.patients = new HashMap<>();

this.hcareWorkers = new HashMap<>();

this.restRoom = new ArrayList<>();

this.window = window;

this.clogger = new CustomLogger();

}

public void enterHospital(Patient patient){

capacity.set(capacity.addAndGet(1));

addPatient(patient);

reception.enterWaitingQueue(patient);

}

public synchronized int enterReception(Patient patient, AuxWorker aWorker){

int desk = aWorker.availableDesk(patient);

if(desk == 0)

{

System.out.println("Patient P"+String.format("%04d", patient.getPid()) + " has come without an appointment");

clogger.write("Patient P"+String.format("%04d", patient.getPid()) + " has come without an appointment", "Reception");

}

else

{

System.out.println("Patient P"+String.format("%04d", patient.getPid())+" in desk "+desk + "controlled by " + getVaccRoom().getDesks().get(desk-1).getWorker());

clogger.write("Patient P"+String.format("%04d", patient.getPid())+" in desk "+desk + "controlled by " + getVaccRoom().getDesks().get(desk-1).getWorker(), "Reception");

}

return desk;

}

public int enterVaccRoom(Patient patient, int iDDesk){

try

{

semEnterVacc.acquire();

}

catch (InterruptedException ex)

{

Logger.getLogger(Hospital.class.getName()).log(Level.SEVERE, null, ex);

}

vaccRoom.sitPatient(patient, iDDesk, hcareWorkers);

vaccRoom.vaccinate(patient, hcareWorkers.get(getVaccRoom().getDesks().get(iDDesk-1).getWorker()));

try //we try to enter the observation room

{

semEnterObs.acquire();

}

catch(Exception e){}

int obsDesk = obsRoom.getAvailableDesk();

vaccRoom.exitPatient(patient,iDDesk); // it leaves the desk

semEnterVacc.release();

return obsDesk;

}

public void enterObservationRoom(Patient patient, int iDDesk){

boolean allWorkersBusy = true;

System.out.println("Patient P"+String.format("%04d", patient.getPid())+" in desk "+iDDesk + "controlled by " + getObsRoom().getDesks().get(iDDesk-1).getWorker());

clogger.write("Patient P"+String.format("%04d", patient.getPid())+" in desk "+iDDesk + "controlled by " + getObsRoom().getDesks().get(iDDesk-1).getWorker(), "Observation Room");

obsRoom.sitPatient(patient, iDDesk);

try

{

patient.sleep(10000);

} catch(Exception e) {

try {

semException.acquire();

ArrayList<Desk> desksVaccRoom = vaccRoom.getDesks();

for (Desk desk : desksVaccRoom)

{

// In case that there is a worker and not a patient

if (desk.getWorker() != -1 && desk.getPatient() == -1)

{

allWorkersBusy = false;

// first case, the worker is sleeping because it has no work to do

if (hcareWorkers.get(desk.getWorker()).isWorking())

{

hcareWorkers.get(desk.getWorker()).signalNoWorkToDo();

}

}

}

// second case, all workers were busy and are some workers sleeping.

if (allWorkersBusy && !restRoom.isEmpty()){

restRoom.get(0).interrupt();

}

} catch (InterruptedException ex) {

Logger.getLogger(Hospital.class.getName()).log(Level.SEVERE, null, ex);

}

}

obsRoom.exitPatient(patient, iDDesk);

System.out.println("Patient P"+String.format("%04d", patient.getPid())+" exits the hospital");

clogger.write("Patient P"+String.format("%04d", patient.getPid())+ " exits the hospital", "Observation Room");

semEnterObs.release();

capacity.addAndGet(-1);

removePatient(patient);

}

public VaccRoom getVaccRoom(){

return this.vaccRoom;

}

public void setVaccRoom(VaccRoom vaccRoom){

this.vaccRoom = vaccRoom;

}

public Reception getReception(){

return this.reception;

}

public void setReception(Reception reception){

this.reception = reception;

}

public ObservationRoom getObsRoom(){

return this.obsRoom;

}

public void setObsRoom(ObservationRoom obsRoom){

this.obsRoom = obsRoom;

}

public CustomLogger getClogger() {

return clogger;

}

public void setClogger(CustomLogger clogger) {

this.clogger = clogger;

}

public ArrayList<HcareWorker> getRestRoom(){

return this.restRoom;

}

public void addPatient(Patient patient){

try

{

semPatients.acquire();

}catch(Exception e){}

patients.put(patient.getPid(), patient);

semPatients.release();

}

public void removePatient(Patient patient){

try

{

semPatients.acquire();

}catch(Exception e){}

patients.remove(patient.getPid());

semPatients.release();

}

public Patient getPatient(int patientID){

try

{

semPatients.acquire();

}catch(Exception e){}

Patient patient = patients.get(patientID);

semPatients.release();

return patient;

}

public HashMap<Integer, HcareWorker> getHcareWorkers()

{

return hcareWorkers;

}

public synchronized void addWorker(HcareWorker worker)

{

hcareWorkers.put(worker.getHId(), worker);

}

public HashMap<Integer, Patient> getPatients(){

try

{

semPatients.acquire();

}catch(Exception e){}

HashMap<Integer, Patient> p = patients;

semPatients.release();

return p;

}

public String restRoomToString(){

String text = "";

for (HcareWorker worker : restRoom) {

text += "H" + worker.getHId() + ", ";

}

return text;

}

public CustomLogger getLogger(){

return this.clogger;

}

public void startWindow()

{

StringManager updater = new StringManager(this,this.window);

updater.start();

}

}

package pecl;

import java.util.ArrayList;

import java.util.concurrent.locks.Condition;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

import java.util.logging.Level;

import java.util.logging.Logger;

public class HcareWorker extends Thread{

private int hid;

private int iDDeskVacc;

private int iDDeskObs;

private int timeToVaccine;

private int timeToRest;

private int counter;

private final int maximum;

private boolean beenAwaken;

private boolean working;

private Lock lock;

private Condition noWorkToDo;

private Hospital hospital;

private boolean isVaccinating;

// private CustomLogger clogger;

/\*\*

\* Constructor of the HcareWorker

\* @param id

\* @param pVaccinated

\* @param hospital

\*/

public HcareWorker(int id, Hospital hospital) {

this.hid = id;

this.hospital = hospital;

this.beenAwaken = false;

this.maximum = 2;

this.lock = new ReentrantLock();

this.noWorkToDo = lock.newCondition();

this.working = false;

this.iDDeskVacc = -1;

this.isVaccinating = false;

// this.clogger = hospital.getLogger();

}

@Override

public void run(){

//System.out.println("haha xd");

ArrayList<Desk> desksVaccRoom;

ArrayList<Desk> desksObsRoom;

int timeWithComplications;

if (beenAwaken)

{

lock.lock();

desksObsRoom = hospital.getObsRoom().getDesks();

iDDeskObs = hospital.getObsRoom().checkComplications(hospital.getPatients()).get(0);

desksObsRoom.get(iDDeskObs).setWorker(hid);

int idPatient = desksObsRoom.get(iDDeskObs).getPatient();

timeWithComplications = 2000 + (int) Math.random() \* 3001;

hospital.getPatient(idPatient).setTimeWithComplications(timeWithComplications);

try {

sleep(timeWithComplications);

}

catch (InterruptedException ex)

{

System.out.println("Interrupted while helping the patient with complications #1");

}

finally

{

lock.unlock();

}

}

else

{

try

{

//starting the schedule.

//hospital.getLogger().log("Healthcare Worker " + hid + " started his schedule. ");

sleep(1000 + (int) (Math.random() \* 2001));

}

// impossible to happen since HcareWorkers are created first

catch (InterruptedException ex) {}

}

try {

desksVaccRoom = hospital.getVaccRoom().getDesks();

int i = 1;

while (iDDeskVacc == -1)

{

if (i >= desksVaccRoom.size() + 1)

{

i = 1;

}

Desk desk = desksVaccRoom.get(i-1);

if (desk.getWorker() == -1)

{

desk.setWorker(hid);

desksVaccRoom.set(i-1, desk);

iDDeskVacc = i;

hospital.getVaccRoom().setDesks(desksVaccRoom);

}

i++;

}

}

catch(Exception e){}

finally

{

//lock.unlock();

}

try

{

lock.lock();

desksVaccRoom = hospital.getVaccRoom().getDesks();

while (desksVaccRoom.get(iDDeskVacc - 1).getPatient() == -1)

{

working = false;

noWorkToDo.await();

}

// Worker has work to do (vaccinate patient).

working = true;

isVaccinating = true;

timeToVaccine = 3000 + (int) Math.random() \* 2000;

sleep(timeToVaccine); // vaccinating

int pid = desksVaccRoom.get(iDDeskVacc-1).getPatient();

isVaccinating = false;

hospital.getVaccRoom().notifyVaccine(hospital.getPatient(pid));

counter++;

if (counter % maximum == 0)

{

try

{

hospital.getRestRoom().add(this);

timeToRest = 5000 + (int) Math.random() \* 3000;

sleep(timeToRest);

}

catch (InterruptedException ex)

{

// awaken while taking a break.

beenAwaken = true;

this.start();

System.out.println("Awaken while having a break...");

}

}

}

catch (InterruptedException ex)

{

System.out.println("Interrupted while on condition of not working. ");

}

finally

{

lock.unlock();

}

// checking if any patient is requesting help due to complications

while (!hospital.getObsRoom().checkComplications(hospital.getPatients()).isEmpty()) {

System.out.println("COMPLICATIONS APPEARED! ");

lock.lock();

desksObsRoom = hospital.getObsRoom().getDesks();

desksVaccRoom = hospital.getVaccRoom().getDesks();

iDDeskObs = hospital.getObsRoom().checkComplications(hospital.getPatients()).get(0);

desksVaccRoom.get(iDDeskVacc-1).setWorker(-1);

desksObsRoom.get(iDDeskObs-1).setWorker(hid);

hospital.getObsRoom().checkComplications(hospital.getPatients()).remove(0);

iDDeskVacc = -1;

int idPatient = desksObsRoom.get(iDDeskObs-1).getPatient();

timeWithComplications = 2000 + (int) Math.random() \* 3001;

hospital.getPatient(idPatient).setTimeWithComplications(timeWithComplications);

lock.unlock();

try

{

// treating the patient.

sleep(timeWithComplications);

}

catch (InterruptedException ex)

{

System.out.println("Interrupted while helping the patient with complications #2");

Logger.getLogger(HcareWorker.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

/\*\*

\* Method which signalls if the HcareWorker has work to do.

\*

\*/

public void signalNoWorkToDo(){

lock.lock();

try {

noWorkToDo.signal();

} finally {

lock.unlock();

}

}

public synchronized boolean isVaccinating() {

return isVaccinating;

}

public boolean isWorking(){

return this.working;

}

public void setTimeToVaccine(int time){

this.timeToVaccine = time;

}

public int getHId() {

return hid;

}

}

package pecl;

public class Desk {

private int iDDesk;

private int iDPatient;

private int iDWorker;

public Desk(int iDDesk){

this.iDDesk = iDDesk;

this.iDPatient = -1;

this.iDWorker = -1;

}

/\*\*

\* When the ID is -1, it means that no thread is occupying it

\*/

public void leavePatient(){

this.iDPatient = -1;

}

/\*\*

\* When the ID is -1, it means that no thread is occupying it

\*/

public void leaveWorker(){

this.iDWorker = -1;

}

/\*\*

\* Sets the Patient ID on the desk

\* @param idPatient The ID of the patient who is sitting on the desk

\*/

public void setPatient(int idPatient){

this.iDPatient = idPatient;

}

/\*\*

\* Returns the ID of the patient that is on the desk.

\* If the number returned is -1, it means there is no patient on the desk

\* @return The ID of the patient or -1 if there is no patient

\*/

public int getPatient(){

return this.iDPatient;

}

/\*\*

\* Sets the Worker ID on the desk

\* @param idWorker The ID of the worker who is sitting on the desk

\*/

public void setWorker(int idWorker){

this.iDWorker = idWorker;

}

/\*\*

\* Returns the ID of the worker that is on the desk

\* If the number returned is -1, it means there is no worker on the desk

\* @return The ID of the worker or -1 if there is no worker

\*/

public int getWorker(){

return this.iDWorker;

}

@Override

public String toString() {

String pid = String.format("%04d", iDPatient);

String wid = String.format("%02d", iDWorker);

if (iDPatient < 1 && iDWorker == -1) //Empty desk

{

return "";

}

else if ((iDPatient < 1) && iDWorker != -1)//Desk without Patient

{

return "H" + wid;

}

else if ((iDPatient > 1) && iDWorker == -1)//Desk without worker

{

return "P" + pid;

}

else //Full desk

{

return "H" + wid + ", " + "P" + pid;

}

}

}

package pecl;

import java.io.BufferedWriter;

import java.io.FileWriter;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

public class CustomLogger {

private String path = "./Logs/EvolutionHospital.txt";

private BufferedWriter bw;

public CustomLogger() {

try {

this.bw = new BufferedWriter(new FileWriter(path));

}catch(Exception e){

System.out.println("Error");

}

}

public synchronized void write(String line, String room)

{

try

{

String time = LocalDateTime.now().format(DateTimeFormatter.ofPattern("[MM-dd-yyyy][HH:mm:ss]"));

bw.write(time + " | " + room + " --- " + line);

bw.newLine();

bw.flush();

}catch(Exception e){

System.out.println("Error");

}

}

}

package pecl;

import java.util.concurrent.Semaphore;

import java.util.logging.Level;

import java.util.logging.Logger;

public class AuxWorker extends Thread {

private int aid;

private int counter;

private int totalCount;

private Hospital hospital;

private boolean isResting;

private Semaphore semCounter = new Semaphore(1);

public AuxWorker(int aid, int maximum, Hospital hospital) {

this.aid = aid;

this.hospital = hospital;

this.counter = 0;

this.totalCount = 0;

this.isResting = false;

}

@Override

public void run() {

// reception assistant

if (aid == 1)

{

while (totalCount != 2000)

{

if (counter == 15)

{

System.out.println("entering queue size: " + hospital.getReception().getEnteringQueue().size());

System.out.println("Auxiliary A1 begins his rest");

hospital.getClogger().write("Auxiliary A1 begins his rest", "Reception");

isResting = true;

try

{

sleep(3000 + (int) (Math.random() \* 2001));

}

catch (InterruptedException ex)

{

Logger.getLogger(AuxWorker.class.getName()).log(Level.SEVERE, null, ex);

}

finally

{

System.out.println("Auxiliary A1 ends his rest");

hospital.getClogger().write("Auxiliary A1 ends his rest", "Reception");

isResting = false;

resetCounter();

}

}

hospital.getReception().getNextPatient();

}

}

else

{

while (totalCount != 2000)

{

try

{

sleep(500 + (int) (Math.random() \* 501));

hospital.getVaccRoom().createVaccine();

addToCounter();

}

catch (InterruptedException ex)

{

Logger.getLogger(AuxWorker.class.getName()).log(Level.SEVERE, null, ex);

}

if (counter == 20)

{

System.out.println("Auxiliary A2 begins his rest");

hospital.getClogger().write("Auxiliary A2 begins his rest", "Vaccination Room");

isResting = true;

try

{

sleep(1000 + (int) (Math.random() \* 4001));

}

catch (InterruptedException ex)

{

Logger.getLogger(AuxWorker.class.getName()).log(Level.SEVERE, null, ex);

}

finally

{

System.out.println("Auxiliary A2 ends his rest");

hospital.getClogger().write("Auxiliary A2 ends his rest", "Vaccination Room");

isResting = false;

resetCounter();

}

}

}

}

}

public synchronized int availableDesk(Patient patient){

if (patient.hasAppointment()){

hospital.getReception().exitWaitingQueue(patient);

hospital.getReception().getAuxWorker().addToCounter();

int timeToSleep = 500 + (int) (Math.random() \* 500);

patient.setTimeToGetDesk(timeToSleep);

try

{ // checking the desk

AuxWorker.sleep(timeToSleep);

}

catch (InterruptedException ex)

{

Logger.getLogger(AuxWorker.class.getName()).log(Level.SEVERE, null, ex);

}

hospital.getReception().enterEnteringQueue(patient);

int vacDesk = hospital.getVaccRoom().getAvailableDesk();

hospital.getReception().exitEnteringQueue(patient);

return vacDesk;

}

else

{

hospital.getReception().exitWaitingQueue(patient); //the patient didn't have an appointment

hospital.removePatient(patient); // so it leaves the hospital

return 0;

}

}

public int getAid() {

return aid;

}

public boolean isResting(){

return this.isResting;

}

public void addToCounter(){

try

{

semCounter.acquire();

counter++;

totalCount++;

}catch(Exception e){}

finally{

semCounter.release();

}

}

public void resetCounter(){

try

{

semCounter.acquire();

counter = 0;

}catch(Exception e){}

finally{

semCounter.release();

}

}

}

package hospitalInterface;

import java.util.ArrayList;

import pecl.\*;

public class MainWindow extends javax.swing.JFrame {

/\*\*

\* Creates new form MainWindow

\*/

public MainWindow() {

initComponents();

}

/\*\*

\* This method is called from within the constructor to initialize the form.

\* WARNING: Do NOT modify this code. The content of this method is always

\* regenerated by the Form Editor.

\*/

@SuppressWarnings("unchecked")

// <editor-fold defaultstate="collapsed" desc="Generated Code">

private void initComponents() {

ReceptionLabel = new javax.swing.JLabel();

ReceptionTextPane = new javax.swing.JScrollPane();

ReceptionText = new javax.swing.JTextArea();

RPatientLabel = new javax.swing.JLabel();

RPatientTextPane = new javax.swing.JScrollPane();

RPatientText = new javax.swing.JTextArea();

RAuxiliaryLabel = new javax.swing.JLabel();

RAuxiliaryTextPane = new javax.swing.JScrollPane();

RAuxiliaryText = new javax.swing.JTextArea();

RestroomLabel = new javax.swing.JLabel();

RestroomTextPane = new javax.swing.JScrollPane();

RestroomText = new javax.swing.JTextArea();

VaccinationRoomLabel = new javax.swing.JLabel();

VDeskLabel1 = new javax.swing.JLabel();

VDeskPane1 = new javax.swing.JScrollPane();

VDesk1 = new javax.swing.JTextArea();

VDeskLabel2 = new javax.swing.JLabel();

VDeskPane2 = new javax.swing.JScrollPane();

VDesk2 = new javax.swing.JTextArea();

VDeskLabel3 = new javax.swing.JLabel();

VDeskPane3 = new javax.swing.JScrollPane();

VDesk3 = new javax.swing.JTextArea();

VDeskLabel4 = new javax.swing.JLabel();

VDeskPane4 = new javax.swing.JScrollPane();

VDesk4 = new javax.swing.JTextArea();

VDeskLabel5 = new javax.swing.JLabel();

VDeskPane5 = new javax.swing.JScrollPane();

VDesk5 = new javax.swing.JTextArea();

VDeskLabel6 = new javax.swing.JLabel();

VDeskPane6 = new javax.swing.JScrollPane();

VDesk6 = new javax.swing.JTextArea();

VDeskLabel7 = new javax.swing.JLabel();

VDeskPane7 = new javax.swing.JScrollPane();

VDesk7 = new javax.swing.JTextArea();

VDeskLabel8 = new javax.swing.JLabel();

VDeskPane8 = new javax.swing.JScrollPane();

VDesk8 = new javax.swing.JTextArea();

VDeskLabel9 = new javax.swing.JLabel();

VDeskPane9 = new javax.swing.JScrollPane();

VDesk9 = new javax.swing.JTextArea();

VDeskLabel10 = new javax.swing.JLabel();

VDeskPane10 = new javax.swing.JScrollPane();

VDesk10 = new javax.swing.JTextArea();

VVaccinesLabel = new javax.swing.JLabel();

VVaccinesPane = new javax.swing.JScrollPane();

VVaccinesText = new javax.swing.JTextArea();

VAuxiliaryLabel = new javax.swing.JLabel();

VAuxiliaryPane = new javax.swing.JScrollPane();

VAuxiliaryText = new javax.swing.JTextArea();

ObservationRoomLabel = new javax.swing.JLabel();

ODeskLabel1 = new javax.swing.JLabel();

ODeskPane1 = new javax.swing.JScrollPane();

ODesk1 = new javax.swing.JTextArea();

ODeskLabel2 = new javax.swing.JLabel();

ODeskPane2 = new javax.swing.JScrollPane();

ODesk2 = new javax.swing.JTextArea();

ODeskLabel3 = new javax.swing.JLabel();

ODeskPane3 = new javax.swing.JScrollPane();

ODesk3 = new javax.swing.JTextArea();

ODeskLabel4 = new javax.swing.JLabel();

ODeskPane4 = new javax.swing.JScrollPane();

ODesk4 = new javax.swing.JTextArea();

ODeskLabel5 = new javax.swing.JLabel();

ODeskPane5 = new javax.swing.JScrollPane();

ODesk5 = new javax.swing.JTextArea();

ODeskLabel6 = new javax.swing.JLabel();

ODeskPane6 = new javax.swing.JScrollPane();

ODesk6 = new javax.swing.JTextArea();

ODeskLabel7 = new javax.swing.JLabel();

ODeskPane7 = new javax.swing.JScrollPane();

ODesk7 = new javax.swing.JTextArea();

ODeskLabel8 = new javax.swing.JLabel();

ODeskPane8 = new javax.swing.JScrollPane();

ODesk8 = new javax.swing.JTextArea();

ODeskLabel9 = new javax.swing.JLabel();

ODeskPane9 = new javax.swing.JScrollPane();

ODesk9 = new javax.swing.JTextArea();

ODeskLabel10 = new javax.swing.JLabel();

ODeskPane10 = new javax.swing.JScrollPane();

ODesk10 = new javax.swing.JTextArea();

ODeskLabel11 = new javax.swing.JLabel();

ODeskPane11 = new javax.swing.JScrollPane();

ODesk11 = new javax.swing.JTextArea();

ODeskLabel12 = new javax.swing.JLabel();

ODeskPane12 = new javax.swing.JScrollPane();

ODesk12 = new javax.swing.JTextArea();

ODeskLabel13 = new javax.swing.JLabel();

ODeskPane13 = new javax.swing.JScrollPane();

ODesk13 = new javax.swing.JTextArea();

ODeskLabel14 = new javax.swing.JLabel();

ODeskPane14 = new javax.swing.JScrollPane();

ODesk14 = new javax.swing.JTextArea();

ODeskLabel15 = new javax.swing.JLabel();

ODeskPane15 = new javax.swing.JScrollPane();

ODesk15 = new javax.swing.JTextArea();

ODeskLabel16 = new javax.swing.JLabel();

ODeskPane16 = new javax.swing.JScrollPane();

ODesk16 = new javax.swing.JTextArea();

ODeskLabel17 = new javax.swing.JLabel();

ODeskPane17 = new javax.swing.JScrollPane();

ODesk17 = new javax.swing.JTextArea();

ODeskLabel18 = new javax.swing.JLabel();

ODeskPane18 = new javax.swing.JScrollPane();

ODesk18 = new javax.swing.JTextArea();

ODeskLabel19 = new javax.swing.JLabel();

ODeskPane19 = new javax.swing.JScrollPane();

ODesk19 = new javax.swing.JTextArea();

ODeskLabel20 = new javax.swing.JLabel();

ODeskPane20 = new javax.swing.JScrollPane();

ODesk20 = new javax.swing.JTextArea();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);

setPreferredSize(new java.awt.Dimension(1280, 720));

ReceptionLabel.setText("RECEPTION");

ReceptionTextPane.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

ReceptionTextPane.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

ReceptionText.setEditable(false);

ReceptionText.setColumns(20);

ReceptionText.setRows(5);

ReceptionTextPane.setViewportView(ReceptionText);

RPatientLabel.setText("Patient");

RPatientTextPane.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

RPatientTextPane.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

RPatientText.setEditable(false);

RPatientText.setColumns(20);

RPatientText.setRows(5);

RPatientTextPane.setViewportView(RPatientText);

RAuxiliaryLabel.setText("Auxiliary");

RAuxiliaryTextPane.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

RAuxiliaryTextPane.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

RAuxiliaryText.setEditable(false);

RAuxiliaryText.setColumns(20);

RAuxiliaryText.setRows(5);

RAuxiliaryTextPane.setViewportView(RAuxiliaryText);

RestroomLabel.setText("RESTROOM");

RestroomTextPane.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

RestroomTextPane.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

RestroomText.setEditable(false);

RestroomText.setColumns(20);

RestroomText.setRows(5);

RestroomText.setPreferredSize(new java.awt.Dimension(400, 300));

RestroomTextPane.setViewportView(RestroomText);

VaccinationRoomLabel.setText("VACCINATION ROOM");

VDeskLabel1.setText("Desk 1");

VDeskPane1.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

VDeskPane1.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

VDeskPane1.setPreferredSize(new java.awt.Dimension(100, 50));

VDesk1.setEditable(false);

VDesk1.setColumns(20);

VDesk1.setRows(5);

VDeskPane1.setViewportView(VDesk1);

VDeskLabel2.setText("Desk 2");

VDeskPane2.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

VDeskPane2.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

VDeskPane2.setPreferredSize(new java.awt.Dimension(100, 50));

VDesk2.setEditable(false);

VDesk2.setColumns(20);

VDesk2.setRows(5);

VDeskPane2.setViewportView(VDesk2);

VDeskLabel3.setText("Desk 3");

VDeskPane3.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

VDeskPane3.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

VDeskPane3.setPreferredSize(new java.awt.Dimension(100, 50));

VDesk3.setEditable(false);

VDesk3.setColumns(20);

VDesk3.setRows(5);

VDeskPane3.setViewportView(VDesk3);

VDeskLabel4.setText("Desk 4");

VDeskPane4.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

VDeskPane4.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

VDeskPane4.setPreferredSize(new java.awt.Dimension(100, 50));

VDesk4.setEditable(false);

VDesk4.setColumns(20);

VDesk4.setRows(5);

VDeskPane4.setViewportView(VDesk4);

VDeskLabel5.setText("Desk 5");

VDeskPane5.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

VDeskPane5.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

VDeskPane5.setPreferredSize(new java.awt.Dimension(100, 50));

VDesk5.setEditable(false);

VDesk5.setColumns(20);

VDesk5.setRows(5);

VDeskPane5.setViewportView(VDesk5);

VDeskLabel6.setText("Desk 6");

VDeskPane6.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

VDeskPane6.setToolTipText("");

VDeskPane6.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

VDeskPane6.setPreferredSize(new java.awt.Dimension(100, 50));

VDesk6.setEditable(false);

VDesk6.setColumns(20);

VDesk6.setRows(5);

VDeskPane6.setViewportView(VDesk6);

VDeskLabel7.setText("Desk 7");

VDeskPane7.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

VDeskPane7.setToolTipText("");

VDeskPane7.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

VDeskPane7.setPreferredSize(new java.awt.Dimension(100, 50));

VDesk7.setEditable(false);

VDesk7.setColumns(20);

VDesk7.setRows(5);

VDeskPane7.setViewportView(VDesk7);

VDeskLabel8.setText("Desk 8");

VDeskPane8.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

VDeskPane8.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

VDeskPane8.setPreferredSize(new java.awt.Dimension(100, 50));

VDesk8.setEditable(false);

VDesk8.setColumns(20);

VDesk8.setRows(5);

VDeskPane8.setViewportView(VDesk8);

VDeskLabel9.setText("Desk 9");

VDeskPane9.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

VDeskPane9.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

VDeskPane9.setPreferredSize(new java.awt.Dimension(100, 50));

VDesk9.setEditable(false);

VDesk9.setColumns(20);

VDesk9.setRows(5);

VDeskPane9.setViewportView(VDesk9);

VDeskLabel10.setText("Desk 10");

VDeskPane10.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

VDeskPane10.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

VDeskPane10.setPreferredSize(new java.awt.Dimension(100, 50));

VDesk10.setEditable(false);

VDesk10.setColumns(20);

VDesk10.setRows(5);

VDeskPane10.setViewportView(VDesk10);

VVaccinesLabel.setText("Available Vaccines");

VVaccinesPane.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

VVaccinesPane.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

VVaccinesPane.setPreferredSize(new java.awt.Dimension(70, 30));

VVaccinesText.setEditable(false);

VVaccinesText.setColumns(20);

VVaccinesText.setRows(5);

VVaccinesPane.setViewportView(VVaccinesText);

VAuxiliaryLabel.setText("Auxiliary");

VAuxiliaryPane.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

VAuxiliaryPane.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

VAuxiliaryText.setEditable(false);

VAuxiliaryText.setColumns(20);

VAuxiliaryText.setRows(5);

VAuxiliaryPane.setViewportView(VAuxiliaryText);

ObservationRoomLabel.setText("OBSERVATION ROOM");

ODeskLabel1.setText("Desk 1");

ODeskPane1.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

ODeskPane1.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

ODeskPane1.setPreferredSize(new java.awt.Dimension(100, 50));

ODesk1.setEditable(false);

ODesk1.setColumns(20);

ODesk1.setRows(5);

ODeskPane1.setViewportView(ODesk1);

ODeskLabel2.setText("Desk 2");

ODeskPane2.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

ODeskPane2.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

ODeskPane2.setPreferredSize(new java.awt.Dimension(100, 50));

ODesk2.setEditable(false);

ODesk2.setColumns(20);

ODesk2.setRows(5);

ODeskPane2.setViewportView(ODesk2);

ODeskLabel3.setText("Desk 3");

ODeskPane3.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

ODeskPane3.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

ODeskPane3.setPreferredSize(new java.awt.Dimension(100, 50));

ODesk3.setEditable(false);

ODesk3.setColumns(20);

ODesk3.setRows(5);

ODeskPane3.setViewportView(ODesk3);

ODeskLabel4.setText("Desk 4");

ODeskPane4.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

ODeskPane4.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

ODeskPane4.setPreferredSize(new java.awt.Dimension(100, 50));

ODesk4.setEditable(false);

ODesk4.setColumns(20);

ODesk4.setRows(5);

ODeskPane4.setViewportView(ODesk4);

ODeskLabel5.setText("Desk 5");

ODeskPane5.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

ODeskPane5.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

ODeskPane5.setPreferredSize(new java.awt.Dimension(100, 50));

ODesk5.setEditable(false);

ODesk5.setColumns(20);

ODesk5.setRows(5);

ODeskPane5.setViewportView(ODesk5);

ODeskLabel6.setText("Desk 6");

ODeskPane6.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

ODeskPane6.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

ODeskPane6.setPreferredSize(new java.awt.Dimension(100, 50));

ODesk6.setEditable(false);

ODesk6.setColumns(20);

ODesk6.setRows(5);

ODeskPane6.setViewportView(ODesk6);

ODeskLabel7.setText("Desk 7");

ODeskPane7.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

ODeskPane7.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

ODeskPane7.setPreferredSize(new java.awt.Dimension(100, 50));

ODesk7.setEditable(false);

ODesk7.setColumns(20);

ODesk7.setRows(5);

ODeskPane7.setViewportView(ODesk7);

ODeskLabel8.setText("Desk 8");

ODeskPane8.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

ODeskPane8.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

ODeskPane8.setPreferredSize(new java.awt.Dimension(100, 50));

ODesk8.setEditable(false);

ODesk8.setColumns(20);

ODesk8.setRows(5);

ODeskPane8.setViewportView(ODesk8);

ODeskLabel9.setText("Desk 9");

ODeskPane9.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

ODeskPane9.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

ODeskPane9.setPreferredSize(new java.awt.Dimension(100, 50));

ODesk9.setEditable(false);

ODesk9.setColumns(20);

ODesk9.setRows(5);

ODeskPane9.setViewportView(ODesk9);

ODeskLabel10.setText("Desk 10");

ODeskPane10.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

ODeskPane10.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

ODeskPane10.setPreferredSize(new java.awt.Dimension(100, 50));

ODesk10.setEditable(false);

ODesk10.setColumns(20);

ODesk10.setRows(5);

ODeskPane10.setViewportView(ODesk10);

ODeskLabel11.setText("Desk 11");

ODeskPane11.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

ODeskPane11.setToolTipText("");

ODeskPane11.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

ODeskPane11.setPreferredSize(new java.awt.Dimension(100, 50));

ODesk11.setEditable(false);

ODesk11.setColumns(20);

ODesk11.setRows(5);

ODeskPane11.setViewportView(ODesk11);

ODeskLabel12.setText("Desk 12");

ODeskPane12.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

ODeskPane12.setToolTipText("");

ODeskPane12.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

ODeskPane12.setPreferredSize(new java.awt.Dimension(100, 50));

ODesk12.setEditable(false);

ODesk12.setColumns(20);

ODesk12.setRows(5);

ODeskPane12.setViewportView(ODesk12);

ODeskLabel13.setText("Desk 13");

ODeskPane13.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

ODeskPane13.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

ODeskPane13.setPreferredSize(new java.awt.Dimension(100, 50));

ODesk13.setEditable(false);

ODesk13.setColumns(20);

ODesk13.setRows(5);

ODeskPane13.setViewportView(ODesk13);

ODeskLabel14.setText("Desk 14");

ODeskPane14.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

ODeskPane14.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

ODeskPane14.setPreferredSize(new java.awt.Dimension(100, 50));

ODesk14.setEditable(false);

ODesk14.setColumns(20);

ODesk14.setRows(5);

ODeskPane14.setViewportView(ODesk14);

ODeskLabel15.setText("Desk 15");

ODeskPane15.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

ODeskPane15.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

ODeskPane15.setPreferredSize(new java.awt.Dimension(100, 50));

ODesk15.setEditable(false);

ODesk15.setColumns(20);

ODesk15.setRows(5);

ODeskPane15.setViewportView(ODesk15);

ODeskLabel16.setText("Desk 16");

ODeskPane16.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

ODeskPane16.setToolTipText("");

ODeskPane16.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

ODeskPane16.setPreferredSize(new java.awt.Dimension(100, 50));

ODesk16.setEditable(false);

ODesk16.setColumns(20);

ODesk16.setRows(5);

ODeskPane16.setViewportView(ODesk16);

ODeskLabel17.setText("Desk 17");

ODeskPane17.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

ODeskPane17.setToolTipText("");

ODeskPane17.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

ODeskPane17.setPreferredSize(new java.awt.Dimension(100, 50));

ODesk17.setEditable(false);

ODesk17.setColumns(20);

ODesk17.setRows(5);

ODeskPane17.setViewportView(ODesk17);

ODeskLabel18.setText("Desk 18");

ODeskPane18.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

ODeskPane18.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

ODeskPane18.setPreferredSize(new java.awt.Dimension(100, 50));

ODesk18.setEditable(false);

ODesk18.setColumns(20);

ODesk18.setRows(5);

ODeskPane18.setViewportView(ODesk18);

ODeskLabel19.setText("Desk 19");

ODeskPane19.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

ODeskPane19.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

ODeskPane19.setPreferredSize(new java.awt.Dimension(100, 50));

ODesk19.setEditable(false);

ODesk19.setColumns(20);

ODesk19.setRows(5);

ODeskPane19.setViewportView(ODesk19);

ODeskLabel20.setText("Desk 20");

ODeskPane20.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

ODeskPane20.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.VERTICAL\_SCROLLBAR\_NEVER);

ODeskPane20.setPreferredSize(new java.awt.Dimension(100, 50));

ODesk20.setEditable(false);

ODesk20.setColumns(20);

ODesk20.setRows(5);

ODeskPane20.setViewportView(ODesk20);

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());

getContentPane().setLayout(layout);

layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addGap(25, 25, 25)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)

.addComponent(ObservationRoomLabel)

.addGroup(layout.createSequentialGroup()

.addComponent(ReceptionLabel)

.addGap(668, 668, 668)

.addComponent(RestroomLabel))

.addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(ODeskPane11, javax.swing.GroupLayout.PREFERRED\_SIZE, 100, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ODeskPane1, javax.swing.GroupLayout.PREFERRED\_SIZE, 100, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ODeskLabel11))

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(ODeskPane12, javax.swing.GroupLayout.PREFERRED\_SIZE, 100, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ODeskPane2, javax.swing.GroupLayout.PREFERRED\_SIZE, 100, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ODeskLabel12))

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(ODeskPane13, javax.swing.GroupLayout.PREFERRED\_SIZE, 100, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ODeskPane3, javax.swing.GroupLayout.PREFERRED\_SIZE, 100, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ODeskLabel13))

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addComponent(ODeskPane4, javax.swing.GroupLayout.PREFERRED\_SIZE, 100, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGap(18, 18, 18)

.addComponent(ODeskPane5, javax.swing.GroupLayout.PREFERRED\_SIZE, 100, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(ODeskPane14, javax.swing.GroupLayout.PREFERRED\_SIZE, 100, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ODeskLabel14))

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(ODeskLabel15)

.addComponent(ODeskPane15, javax.swing.GroupLayout.PREFERRED\_SIZE, 100, javax.swing.GroupLayout.PREFERRED\_SIZE))))

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(ODeskPane16, javax.swing.GroupLayout.PREFERRED\_SIZE, 100, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ODeskPane6, javax.swing.GroupLayout.PREFERRED\_SIZE, 100, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ODeskLabel6)

.addComponent(ODeskLabel16)))

.addComponent(ODeskLabel1))

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(ODeskPane17, javax.swing.GroupLayout.PREFERRED\_SIZE, 100, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ODeskPane7, javax.swing.GroupLayout.PREFERRED\_SIZE, 100, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ODeskLabel7)

.addComponent(ODeskLabel17))

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(ODeskPane18, javax.swing.GroupLayout.PREFERRED\_SIZE, 100, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ODeskPane8, javax.swing.GroupLayout.PREFERRED\_SIZE, 100, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ODeskLabel8)

.addComponent(ODeskLabel18))

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(ODeskPane9, javax.swing.GroupLayout.PREFERRED\_SIZE, 100, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ODeskLabel9))

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(ODeskLabel10)

.addComponent(ODeskPane10, javax.swing.GroupLayout.PREFERRED\_SIZE, 100, javax.swing.GroupLayout.PREFERRED\_SIZE)))

.addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(ODeskPane19, javax.swing.GroupLayout.PREFERRED\_SIZE, 100, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ODeskLabel19))

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(ODeskLabel20)

.addComponent(ODeskPane20, javax.swing.GroupLayout.PREFERRED\_SIZE, 100, javax.swing.GroupLayout.PREFERRED\_SIZE)))))

.addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(VaccinationRoomLabel)

.addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(VDeskPane6, javax.swing.GroupLayout.PREFERRED\_SIZE, 100, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(VDeskPane1, javax.swing.GroupLayout.PREFERRED\_SIZE, 100, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(VDeskLabel1)

.addComponent(VDeskLabel6))

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(VDeskPane7, javax.swing.GroupLayout.PREFERRED\_SIZE, 100, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(VDeskPane2, javax.swing.GroupLayout.PREFERRED\_SIZE, 100, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(VDeskLabel2)

.addComponent(VDeskLabel7)

.addComponent(ODeskLabel2))

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(VDeskPane8, javax.swing.GroupLayout.PREFERRED\_SIZE, 100, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(VDeskPane3, javax.swing.GroupLayout.PREFERRED\_SIZE, 100, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(VDeskLabel3)

.addComponent(VDeskLabel8)

.addComponent(ODeskLabel3))

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(VDeskPane4, javax.swing.GroupLayout.PREFERRED\_SIZE, 100, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(VDeskLabel4))

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(VDeskLabel5)

.addGroup(layout.createSequentialGroup()

.addComponent(VDeskPane5, javax.swing.GroupLayout.PREFERRED\_SIZE, 100, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(VVaccinesLabel)

.addComponent(VVaccinesPane, javax.swing.GroupLayout.PREFERRED\_SIZE, 70, javax.swing.GroupLayout.PREFERRED\_SIZE)))))

.addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(VDeskPane9, javax.swing.GroupLayout.PREFERRED\_SIZE, 100, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(VDeskLabel9)

.addComponent(ODeskLabel4))

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(ODeskLabel5)

.addComponent(VDeskLabel10)

.addGroup(layout.createSequentialGroup()

.addComponent(VDeskPane10, javax.swing.GroupLayout.PREFERRED\_SIZE, 100, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(VAuxiliaryLabel)

.addComponent(VAuxiliaryPane, javax.swing.GroupLayout.PREFERRED\_SIZE, 70, javax.swing.GroupLayout.PREFERRED\_SIZE)))))))

.addGroup(layout.createSequentialGroup()

.addComponent(ReceptionTextPane, javax.swing.GroupLayout.PREFERRED\_SIZE, 561, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(RPatientLabel)

.addComponent(RPatientTextPane, javax.swing.GroupLayout.PREFERRED\_SIZE, 70, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(RAuxiliaryTextPane, javax.swing.GroupLayout.PREFERRED\_SIZE, 70, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(RAuxiliaryLabel))))

.addGap(73, 73, 73)

.addComponent(RestroomTextPane)))

.addGap(0, 143, Short.MAX\_VALUE))

);

layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(ReceptionLabel)

.addComponent(RestroomLabel))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)

.addComponent(RestroomTextPane, javax.swing.GroupLayout.DEFAULT\_SIZE, 124, Short.MAX\_VALUE)

.addComponent(ReceptionTextPane))

.addGroup(layout.createSequentialGroup()

.addComponent(RPatientLabel)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addComponent(RPatientTextPane, javax.swing.GroupLayout.PREFERRED\_SIZE, 30, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGap(18, 18, 18)

.addComponent(RAuxiliaryLabel)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addComponent(RAuxiliaryTextPane, javax.swing.GroupLayout.PREFERRED\_SIZE, 30, javax.swing.GroupLayout.PREFERRED\_SIZE)))

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addGap(18, 18, 18)

.addComponent(VaccinationRoomLabel)

.addGap(19, 19, 19)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(VDeskLabel1)

.addComponent(VDeskLabel2)

.addComponent(VDeskLabel3)

.addComponent(VDeskLabel4)

.addComponent(VDeskLabel5)))

.addGroup(layout.createSequentialGroup()

.addGap(76, 76, 76)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)

.addComponent(VDeskPane2, javax.swing.GroupLayout.PREFERRED\_SIZE, 50, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(VDeskPane1, javax.swing.GroupLayout.PREFERRED\_SIZE, 50, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(VDeskPane3, javax.swing.GroupLayout.PREFERRED\_SIZE, 50, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(VDeskPane4, javax.swing.GroupLayout.PREFERRED\_SIZE, 50, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(VDeskPane5, javax.swing.GroupLayout.PREFERRED\_SIZE, 50, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGroup(layout.createSequentialGroup()

.addComponent(VVaccinesLabel)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

.addComponent(VVaccinesPane, javax.swing.GroupLayout.PREFERRED\_SIZE, 30, javax.swing.GroupLayout.PREFERRED\_SIZE)))

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(VDeskLabel6)

.addComponent(VDeskLabel8)

.addComponent(VDeskLabel9)

.addComponent(VDeskLabel10))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addGap(2, 2, 2)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(VDeskPane6, javax.swing.GroupLayout.PREFERRED\_SIZE, 50, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(VDeskPane7, javax.swing.GroupLayout.PREFERRED\_SIZE, 50, javax.swing.GroupLayout.PREFERRED\_SIZE)))

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING, false)

.addGroup(layout.createSequentialGroup()

.addComponent(VAuxiliaryLabel)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

.addComponent(VAuxiliaryPane, javax.swing.GroupLayout.PREFERRED\_SIZE, 30, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGroup(javax.swing.GroupLayout.Alignment.LEADING, layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(VDeskPane8, javax.swing.GroupLayout.PREFERRED\_SIZE, 50, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(VDeskPane9, javax.swing.GroupLayout.PREFERRED\_SIZE, 50, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(VDeskPane10, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.PREFERRED\_SIZE, 50, javax.swing.GroupLayout.PREFERRED\_SIZE)))))

.addGap(18, 18, 18)

.addComponent(ObservationRoomLabel, javax.swing.GroupLayout.PREFERRED\_SIZE, 14, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(ODeskLabel1)

.addComponent(ODeskLabel2)

.addComponent(ODeskLabel3)

.addComponent(ODeskLabel4)

.addComponent(ODeskLabel5)

.addComponent(ODeskLabel6)

.addComponent(ODeskLabel7)

.addComponent(ODeskLabel8)

.addComponent(ODeskLabel9)

.addComponent(ODeskLabel10)))

.addComponent(VDeskLabel7))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)

.addGroup(javax.swing.GroupLayout.Alignment.LEADING, layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(ODeskPane2, javax.swing.GroupLayout.PREFERRED\_SIZE, 50, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ODeskPane1, javax.swing.GroupLayout.PREFERRED\_SIZE, 50, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ODeskPane3, javax.swing.GroupLayout.PREFERRED\_SIZE, 50, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ODeskPane4, javax.swing.GroupLayout.PREFERRED\_SIZE, 50, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ODeskPane5, javax.swing.GroupLayout.PREFERRED\_SIZE, 50, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(ODeskLabel11)

.addComponent(ODeskLabel12)

.addComponent(ODeskLabel13)

.addComponent(ODeskLabel14)

.addComponent(ODeskLabel15))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(ODeskPane15, javax.swing.GroupLayout.PREFERRED\_SIZE, 50, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ODeskPane12, javax.swing.GroupLayout.PREFERRED\_SIZE, 50, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ODeskPane11, javax.swing.GroupLayout.PREFERRED\_SIZE, 50, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ODeskPane13, javax.swing.GroupLayout.PREFERRED\_SIZE, 50, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ODeskPane14, javax.swing.GroupLayout.PREFERRED\_SIZE, 50, javax.swing.GroupLayout.PREFERRED\_SIZE)))

.addGroup(javax.swing.GroupLayout.Alignment.LEADING, layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(ODeskPane7, javax.swing.GroupLayout.PREFERRED\_SIZE, 50, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ODeskPane6, javax.swing.GroupLayout.PREFERRED\_SIZE, 50, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ODeskPane8, javax.swing.GroupLayout.PREFERRED\_SIZE, 50, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ODeskPane9, javax.swing.GroupLayout.PREFERRED\_SIZE, 50, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ODeskPane10, javax.swing.GroupLayout.PREFERRED\_SIZE, 50, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(ODeskLabel16)

.addComponent(ODeskLabel17)

.addComponent(ODeskLabel18)

.addComponent(ODeskLabel19)

.addComponent(ODeskLabel20))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(ODeskPane20, javax.swing.GroupLayout.PREFERRED\_SIZE, 50, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ODeskPane17, javax.swing.GroupLayout.PREFERRED\_SIZE, 50, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ODeskPane16, javax.swing.GroupLayout.PREFERRED\_SIZE, 50, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ODeskPane18, javax.swing.GroupLayout.PREFERRED\_SIZE, 50, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(ODeskPane19, javax.swing.GroupLayout.PREFERRED\_SIZE, 50, javax.swing.GroupLayout.PREFERRED\_SIZE))))))

.addContainerGap(960, Short.MAX\_VALUE))

);

pack();

}// </editor-fold>

/\*\*

\* @param args the command line arguments

\*/

public static void main(String args[]) {

/\* Set the Nimbus look and feel \*/

//<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">

/\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.

\* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html

\*/

try {

for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {

if ("Nimbus".equals(info.getName())) {

javax.swing.UIManager.setLookAndFeel(info.getClassName());

break;

}

}

} catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(MainWindow.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(MainWindow.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(MainWindow.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(MainWindow.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

}

//</editor-fold>

/\* Create and display the form \*/

java.awt.EventQueue.invokeLater(new Runnable() {

public void run() {

new MainWindow().setVisible(true);

}

});

}

public javax.swing.JTextArea getReception(){

return ReceptionText;

}

public javax.swing.JTextArea getReceptionPatient(){

return RPatientText;

}

public javax.swing.JTextArea getReceptionAux(){

return RAuxiliaryText;

}

public javax.swing.JTextArea getRestRoom(){

return RestroomText;

}

public ArrayList<javax.swing.JTextArea> getVDesks(){

ArrayList<javax.swing.JTextArea> l = new ArrayList<>();

l.add(VDesk1);

l.add(VDesk2);

l.add(VDesk3);

l.add(VDesk4);

l.add(VDesk5);

l.add(VDesk6);

l.add(VDesk7);

l.add(VDesk8);

l.add(VDesk9);

l.add(VDesk10);

return l;

}

public javax.swing.JTextArea getVaccines(){

return VVaccinesText;

}

public javax.swing.JTextArea getVaccRoomAux(){

return VAuxiliaryText;

}

public ArrayList<javax.swing.JTextArea> getODesks(){

ArrayList<javax.swing.JTextArea> l = new ArrayList<>();

l.add(ODesk1);

l.add(ODesk2);

l.add(ODesk3);

l.add(ODesk4);

l.add(ODesk5);

l.add(ODesk6);

l.add(ODesk7);

l.add(ODesk8);

l.add(ODesk9);

l.add(ODesk10);

l.add(ODesk11);

l.add(ODesk12);

l.add(ODesk13);

l.add(ODesk14);

l.add(ODesk15);

l.add(ODesk16);

l.add(ODesk17);

l.add(ODesk18);

l.add(ODesk19);

l.add(ODesk20);

return l;

}

// Variables declaration - do not modify

private javax.swing.JTextArea ODesk1;

private javax.swing.JTextArea ODesk10;

private javax.swing.JTextArea ODesk11;

private javax.swing.JTextArea ODesk12;

private javax.swing.JTextArea ODesk13;

private javax.swing.JTextArea ODesk14;

private javax.swing.JTextArea ODesk15;

private javax.swing.JTextArea ODesk16;

private javax.swing.JTextArea ODesk17;

private javax.swing.JTextArea ODesk18;

private javax.swing.JTextArea ODesk19;

private javax.swing.JTextArea ODesk2;

private javax.swing.JTextArea ODesk20;

private javax.swing.JTextArea ODesk3;

private javax.swing.JTextArea ODesk4;

private javax.swing.JTextArea ODesk5;

private javax.swing.JTextArea ODesk6;

private javax.swing.JTextArea ODesk7;

private javax.swing.JTextArea ODesk8;

private javax.swing.JTextArea ODesk9;

private javax.swing.JLabel ODeskLabel1;

private javax.swing.JLabel ODeskLabel10;

private javax.swing.JLabel ODeskLabel11;

private javax.swing.JLabel ODeskLabel12;

private javax.swing.JLabel ODeskLabel13;

private javax.swing.JLabel ODeskLabel14;

private javax.swing.JLabel ODeskLabel15;

private javax.swing.JLabel ODeskLabel16;

private javax.swing.JLabel ODeskLabel17;

private javax.swing.JLabel ODeskLabel18;

private javax.swing.JLabel ODeskLabel19;

private javax.swing.JLabel ODeskLabel2;

private javax.swing.JLabel ODeskLabel20;

private javax.swing.JLabel ODeskLabel3;

private javax.swing.JLabel ODeskLabel4;

private javax.swing.JLabel ODeskLabel5;

private javax.swing.JLabel ODeskLabel6;

private javax.swing.JLabel ODeskLabel7;

private javax.swing.JLabel ODeskLabel8;

private javax.swing.JLabel ODeskLabel9;

private javax.swing.JScrollPane ODeskPane1;

private javax.swing.JScrollPane ODeskPane10;

private javax.swing.JScrollPane ODeskPane11;

private javax.swing.JScrollPane ODeskPane12;

private javax.swing.JScrollPane ODeskPane13;

private javax.swing.JScrollPane ODeskPane14;

private javax.swing.JScrollPane ODeskPane15;

private javax.swing.JScrollPane ODeskPane16;

private javax.swing.JScrollPane ODeskPane17;

private javax.swing.JScrollPane ODeskPane18;

private javax.swing.JScrollPane ODeskPane19;

private javax.swing.JScrollPane ODeskPane2;

private javax.swing.JScrollPane ODeskPane20;

private javax.swing.JScrollPane ODeskPane3;

private javax.swing.JScrollPane ODeskPane4;

private javax.swing.JScrollPane ODeskPane5;

private javax.swing.JScrollPane ODeskPane6;

private javax.swing.JScrollPane ODeskPane7;

private javax.swing.JScrollPane ODeskPane8;

private javax.swing.JScrollPane ODeskPane9;

private javax.swing.JLabel ObservationRoomLabel;

private javax.swing.JLabel RAuxiliaryLabel;

private javax.swing.JTextArea RAuxiliaryText;

private javax.swing.JScrollPane RAuxiliaryTextPane;

private javax.swing.JLabel RPatientLabel;

private javax.swing.JTextArea RPatientText;

private javax.swing.JScrollPane RPatientTextPane;

private javax.swing.JLabel ReceptionLabel;

private javax.swing.JTextArea ReceptionText;

private javax.swing.JScrollPane ReceptionTextPane;

private javax.swing.JLabel RestroomLabel;

private javax.swing.JTextArea RestroomText;

private javax.swing.JScrollPane RestroomTextPane;

private javax.swing.JLabel VAuxiliaryLabel;

private javax.swing.JScrollPane VAuxiliaryPane;

private javax.swing.JTextArea VAuxiliaryText;

private javax.swing.JTextArea VDesk1;

private javax.swing.JTextArea VDesk10;

private javax.swing.JTextArea VDesk2;

private javax.swing.JTextArea VDesk3;

private javax.swing.JTextArea VDesk4;

private javax.swing.JTextArea VDesk5;

private javax.swing.JTextArea VDesk6;

private javax.swing.JTextArea VDesk7;

private javax.swing.JTextArea VDesk8;

private javax.swing.JTextArea VDesk9;

private javax.swing.JLabel VDeskLabel1;

private javax.swing.JLabel VDeskLabel10;

private javax.swing.JLabel VDeskLabel2;

private javax.swing.JLabel VDeskLabel3;

private javax.swing.JLabel VDeskLabel4;

private javax.swing.JLabel VDeskLabel5;

private javax.swing.JLabel VDeskLabel6;

private javax.swing.JLabel VDeskLabel7;

private javax.swing.JLabel VDeskLabel8;

private javax.swing.JLabel VDeskLabel9;

private javax.swing.JScrollPane VDeskPane1;

private javax.swing.JScrollPane VDeskPane10;

private javax.swing.JScrollPane VDeskPane2;

private javax.swing.JScrollPane VDeskPane3;

private javax.swing.JScrollPane VDeskPane4;

private javax.swing.JScrollPane VDeskPane5;

private javax.swing.JScrollPane VDeskPane6;

private javax.swing.JScrollPane VDeskPane7;

private javax.swing.JScrollPane VDeskPane8;

private javax.swing.JScrollPane VDeskPane9;

private javax.swing.JLabel VVaccinesLabel;

private javax.swing.JScrollPane VVaccinesPane;

private javax.swing.JTextArea VVaccinesText;

private javax.swing.JLabel VaccinationRoomLabel;

// End of variables declaration

}