a. *Grado en Ingeniería Informática*
b. *Curso 2020/2021–Convocatoria Ordinaria*
c. *03228755S – Núñez Rodriguez Juan Carlos*

DEFENSA PRÁCTICA FINAL PROGRAMACIÓN AVANZADA

1. ANÁLISIS DE ALTO NIVEL

1ª Parte:

La función principal de este programa es crear una simulación de un hospital donde vengan pacientes a vacunarse. El hospital se divide en 4 zonas: la recepción y las salas de vacunación, observación y descanso y existen 3 tipos de personas: pacientes, sanitarios y auxiliares.
La función del paciente será entrar al hospital desde la recepción, que le asignen un puesto de la sala de vacunación y le vacunen, pasar por la sala de observación y finalmente irse.
Los sanitarios se encargarán de vacunar a los pacientes que lleguen a su puesto en la sala de vacunación. Además, estas personas también se encargarán de atender a los pacientes que tengan una reacción con la vacuna en la sala de observación.
Los 2 auxiliares que hay dentro del hospital se encargarán uno de crear las vacunas dentro de la sala de vacunación y el otro de atender a los pacientes en recepción y decirles a qué puesto de la sala de vacunación tienen que ir.
Tanto los sanitarios como los auxiliares se tomarán descansos cada cierto tiempo después de cumplir sus labores.

Además de esto, el programa se encargará de imprimir en un archivo llamado evolucionHospital.txt todos los procesos que puedan resultar importantes dentro del sistema indicando su fecha y hora y la acción que se ha producido. Este documento lo creará cada vez que se ejecute el programa y se puede encontrar en la carpeta principal del proyecto.

2ª Parte:

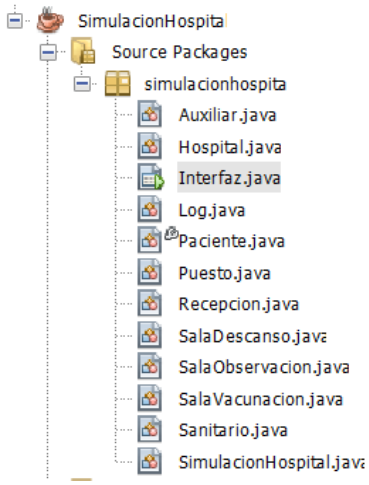
En esta segunda parte la función del programa es la misma que en la primera solo que ahora el programa principal actuará como servidor para posibles clientes que quieran conectarse. Estos clientes podrán hacer dos funciones, visualizar cada segundo lo que va pasando en el programa principal y serán capaces de cerrar los puestos que quieran de la sala de vacunación.

Todo el programa anterior está modificado para que los clientes puedan recibir la información de manera correcta.

2. DISEÑO GENERAL DEL SISTEMA Y HERRAMIENTAS DE SINCRONIZACIÓN

1ª Parte:

Vamos a comenzar explicando un poco por encima el diseño general del sistema y más adelante hablaremos más en detalle de las herramientas de sincronización del programa.
El sistema se compone de varias clases, que vienen a ser las siguientes:



La clase principal se trata de la clase interfaz, la cual se encargará de mostrar la interfaz y de comenzar el proceso de simulación mediante la clase SimulaciónHospital. Una vez comience la simulación, la clase SimulaciónHospital creará e iniciará todos los hilos Paciente, Sanitario y Auxiliar que sean necesarios para comenzar el problema. Esta clase también se encarga de iniciar los hilos Pacientes cada cierto tiempo para que así no haya aglomeraciones.
Cuando empieza la simulación, los auxiliares se van a sus correspondientes puestos y los sanitarios se van a la sala de descanso para vestirse mientras los pacientes van yendo a la recepción.
Cada persona es tratada como un hilo y tendrá que hacer las acciones que les corresponda. Vamos a ver las acciones que realizan cada una de ellas. Pero, antes de seguir, me gustaría recalcar algo que pasa con todas las personas y es que todas ellas emplean la clase hospital como una especie de puente para llegar a las salas, pero realmente en la clase hospital no se hacen las acciones, si no que se hacen en la respectiva sala o puesto. Ahora sí, veamos las acciones de cada persona.

Empezando por los auxiliares, estas son las acciones que hacen:

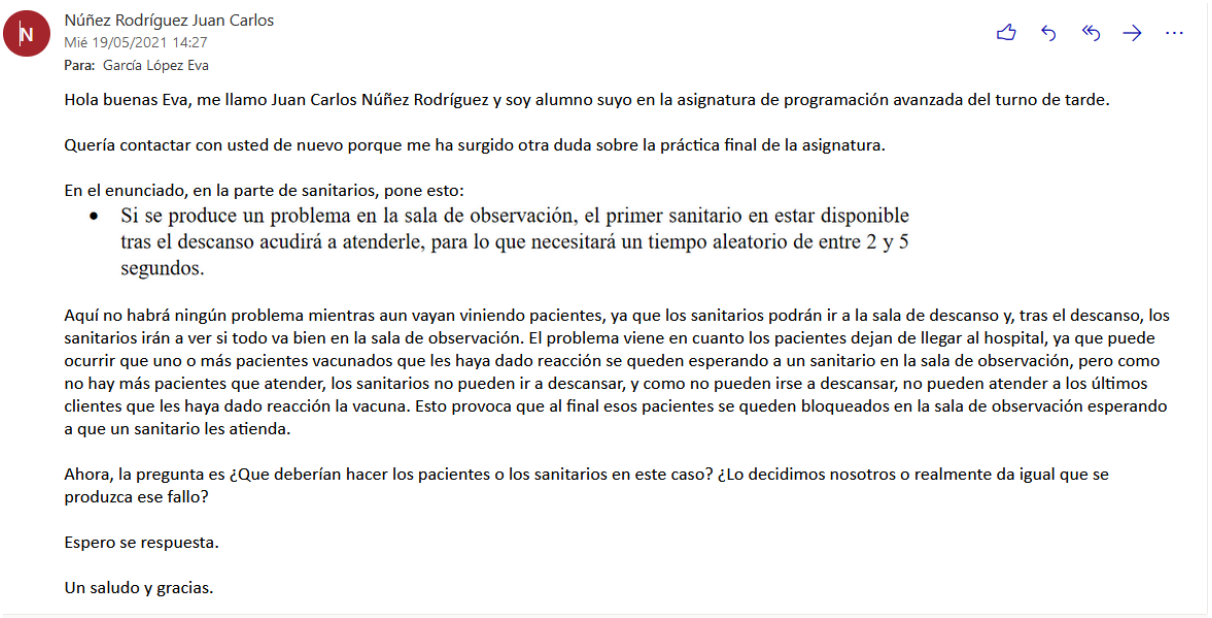
```java
@Override
public void run() {
    while (true) {
        if (id.equals("A1")) {
            hospital.atenderRecepcion(this);
            hospital.entrarSalaDescanso(this, 3000, 5000);
        } else if(id.equals("A2")){
            hospital.crearVacunas(this);
            hospital.entrarSalaDescanso(this, 1000, 4000);
        }
    }

}
```

Vemos que depende de si es el A1 o el A2 hacen una acción u otra. El auxiliar A1 se encarga de llevar la recepción. Este auxiliar se queda esperando en la recepción hasta que un paciente le pase sus datos. Una vez recibidos, el auxiliar va a buscar un puesto que esté libre y que esté abierto dentro de la sala de vacunación. Si no hubiera ninguno disponible, el auxiliar se queda esperando allí hasta que algún sanitario le avise de que ya hay uno disponible. Cuando ya tiene uno, vuelve a recepción y se lo pasa al paciente que estaba en recepción y espera al siguiente paciente. Si hubiera algún paciente sin cita, lanzaría una interrupción al paciente que haría que se fuera. Así sigue hasta que atiende a los clientes suficientes y tras esto se va a descansar. Por otra parte, el A2 va a la sala de vacunación a hacer vacunas. El número de vacunas se guarda en una variable que está sincronizada con las acciones que realizan los sanitarios. Una vez haya hecho las suficientes, se va a descansar.

Ahora vamos con los sanitarios:

```java
@Override
public void run() {
    hospital.entrarSalaDescanso(this, 1000, 3000);
    //Este sanitario se encargará de atender a los pacientes reaccionados si los otros tardan mucho en venir
    if (idSanitario.equals("SAux")) {
        try {
            Thread.sleep(120000);//Tiempo que espera el sanitario para vigilar la sala de observación
        } catch (InterruptedException ex) {
            Logger.getLogger(Sanitario.class.getName()).log(Level.SEVERE, null, ex);
        }
        hospital.vigilarSalaObservacion(this);
    } else {
        hospital.asignarPuestoVacunacion(this);
        while (true) {
            hospital.atenderPuestoVacunacion(this);
            hospital.entrarSalaDescanso(this, 5000, 8000);
            hospital.atenderSalaObservacion(this);
        }
    }
}
```

Antes de entrar en detalle con los sanitarios generales, quiero explicar para qué sirve SAux ya que es algo que no entraba en el enunciado. Este sanitario se emplea para solucionar un problema en la lógica de la práctica que explico en detalle en este correo:



Núñez Rodríguez Juan Carlos
Mié 19/05/2021 14:27
Para: García López Eva

Hola buenas Eva, me llamo Juan Carlos Núñez Rodríguez y soy alumno suyo en la asignatura de programación avanzada del turno de tarde.

Quería contactar con usted de nuevo porque me ha surgido otra duda sobre la práctica final de la asignatura.

En el enunciado, en la parte de sanitarios, pone esto:

- Si se produce un problema en la sala de observación, el primer sanitario en estar disponible tras el descanso acudirá a atenderle, para lo que necesitará un tiempo aleatorio de entre 2 y 5 segundos.

Aquí no habrá ningún problema mientras aun vayan viniendo pacientes, ya que los sanitarios podrán ir a la sala de descanso y, tras el descanso, los sanitarios irán a ver si todo va bien en la sala de observación. El problema viene en cuanto los pacientes dejan de llegar al hospital, ya que puede ocurrir que uno o más pacientes vacunados que les haya dado reacción se queden esperando a un sanitario en la sala de observación, pero como no hay más pacientes que atender, los sanitarios no pueden ir a descansar, y como no pueden irse a descansar, no pueden atender a los últimos clientes que les haya dado reacción la vacuna. Esto provoca que al final esos pacientes se queden bloqueados en la sala de observación esperando a que un sanitario les atienda.

Ahora, la pregunta es ¿Que deberían hacer los pacientes o los sanitarios en este caso? ¿Lo decidimos nosotros o realmente da igual que se produzca ese fallo?

Espero se respuesta.

Un saludo y gracias.

La función principal y única de este sanitario es atender a los pacientes que tengan una reacción a la vacuna y hayan estado esperando mucho tiempo a que les atienda un sanitario. Este sanitario solo actuará en el caso de que el resto de los sanitarios ya no puedan atender a los pacientes de la sala de observación. Esperará un tiempo prudencial y si ve que no hay sanitarios atendiendo, les atenderá él.
Siguiendo con los sanitarios generales, estos hacen varias cosas. Primero, todos los sanitarios van a “descansar”(vestirse) al principio a la sala de descanso, luego, a todos se les asigna un puesto según vayan llegando a la sala de vacunación y se quedan esperando en su puesto hasta que venga algún paciente. Cuando viene uno, el sanitario recibe sus datos y va a buscar una vacuna mientras el paciente espera a ser atendido. Si no hay, espera a que se produzcan más y cuando ya la tiene vuelve al puesto y vacuna al paciente. Este proceso se repite hasta que atienda a los suficientes pacientes y pueda ir a la sala de descanso a descansar. Una vez vuelve de allí, hace una revisión a la sala de observación por si a algún paciente le da una reacción y si hubiera alguno se queda allí atendiendolos hasta que ya no quede ninguno antes de volver a su puesto.

Finalmente están los pacientes:

```java
@Override
public void run() {
    String mensaje = "";
    hospital.entrarRecepcion(this);
    if (puestoVacunacion != null) {
        hospital.entrarSalaVacunacion(this);

        mensaje = id + " se dirige a la sala de observacion";
        hospital.imprimirMensaje(mensaje);

        hospital.entrarSalaObservacion(this);
    }

    mensaje = "El paciente " + id + " se va del hospital";
    hospital.imprimirMensaje(mensaje);
}
```

Los pacientes lo primero que hacen es dirigirse a la recepción del hospital. Allí hacen cola para entrar al hospital y cuando están en la recepción le dan sus datos al auxiliar y esperan su respuesta. Si el auxiliar lanza una interrupción, significa que el paciente ha venido sin cita por lo que el paciente se va del hospital. Si no pasa esto, el auxiliar les mandará un puesto disponible de la sala de vacunación al que tienen que ir. Si están citados o no se calcula en el constructor del paciente. Tras esto, los pacientes se dirigen al puesto indicado de la sala de vacunación y hacen lo mismo que antes, dan sus datos al sanitario y esperan. Una vez que el sanitario les vacune se van a la sala de observación y cogen el primer puesto disponible. Allí esperan una determinada cantidad de tiempo y después se calcula si tienen reacción o no. Si no tienen reacción, se van del hospital. Si tienen reacción, esperan en su puesto hasta que un sanitario les atienda.

Además de esto, las personas irán imprimiendo determinados procesos que realizan en el log que tiene el hospital.

Ahora vamos a pasar a las herramientas de sincronización que he empleado.

Para sincronizar a las personas dentro de la sala de descanso lo único que he hecho ha sido crear un Object lock que he usado como cerrojo y una lista donde se van metiendo las personas que entran a la sala. Las personas que entran simplemente emplean un método synchronized con el lock cada vez que vayan a entrar o salir del arraylist.
Ejemplo de como lo he hecho:

```java
private final Object lock = new Object();
```

```
synchronized (lock) {
    personasDentro.add(persona);
    colaInterfaz.setText(personasDentro.toString());

}
```

Por otro lado, para sincronizar a los pacientes en la recepción y con el auxiliar he empleado principalmente dos herramientas de sincronización. La primera se trata de un Concurrent Linked Queue de personas en el que voy metiendo y sacando a las personas que llegan a la recepción. La segunda sirve para que el auxiliar y el paciente que esté atendiendo se comuniquen de manera adecuada y se trata de dos SynchronousQueue, uno en el que el auxiliar espera a recibir información del paciente y el paciente espera a que su información sea recibida y el segundo que funciona a la inversa. De este modo, cuando llega un nuevo paciente a recepción, ambas partes están listas para enviar o recibir la información correspondiente.

Pasando a la sala de vacunación, he empleado varias herramientas de sincronización. Para asignar los puestos de los sanitarios he utilizado un monitor para que no vayan todos a la vez y cogan el mismo puesto. Luego también he empleado dos Objects, lockPuesto y lockVacunas, utilizándolos de manera similar que en la sala de descanso. LockPuesto sirve para que el auxiliar se quede esperando cuando no hay puestos disponibles para el paciente. Dejará de esperar en cuanto un sanitario le haga un notify(). Luego está lockVacunas, que se emplea para que los sanitarios no cogan las vacunas todos a la vez y haya un orden entre el auxiliar que va creando vacunas y los sanitarios que van cogiendo. Finalmente, también he añadido un semáforo que se emplea como un contador de vacunas. El semáforo al principio tiene de espacio 0 y cuando el auxiliar hace una vacuna, este lanza un release que aumenta el espacio en 1. Los sanitarios, por su parte, intentarán hacer un acquire, y si no hay vacunas, se quedarán esperando hasta que el auxiliar haga una y lance un release.

Luego, en la sala de observación he empleado 2 herramientas. Un semáforo de 20 de capacidad para controlar a la cantidad de pacientes que pueden entrar en esta sala a la vez y un Object que actúa como un lock para ver los puestos de forma sincronizada.

Finalmente, en los puestos he empleado también varias herramientas de sincronización. He empleado monitores en determinadas funciones para que se cogan o modifiquen los datos de forma ordenada y además he empleado 3 SynchronousQueue de forma similar a cómo los uso en la recepción. Con esperaPaciente el paciente envía sus datos y el sanitario los coge, con esperaVacuna el paciente espera a que el sanitario le vacune y esperaReacción el paciente espera a que un sanitario le atienda en la sala de observación.

2ª Parte:

En esta segunda parte se han utilizado 4 nuevas clases: InterfazCliente, ClienteActualizar, ClienteCerrar y Servidor. La clase InterfazCliente será la clase principal de los usuarios que se quieran conectar a la clase Servidor. Esta clase

mostrará la interfaz del hospital con los botones de cerrar en cada puesto de la sala de vacunación y además se encargará de ir creando e inicializando los hilos ClienteCerrar cuando se quiera cerrar un puesto y ClienteActualizar para ir actualizando la interfaz.

Por otra parte, está la clase Servidor que atiende las peticiones de cerrar o actualizar que les envían los clientes. Este servidor funciona como un hilo y lo guarda la clase hospital. Si en algún momento algún hilo modifica algo de la interfaz, también actualiza los datos que tenga el servidor mediante la función actualizarVariable que tiene la clase Servidor y que sirve para actualizar el valor que se quiera de la interfaz.
Cuando el cliente quiera actualizar la pantalla, se conectará al servidor y este le devolverá todos los datos que tenga almacenado, que estarán actualizados ya que los otros hilos se han encargado de actualizarlos.

Finalmente si se quiere cerrar un puesto, lo que hace el cliente es crear un nuevo ClienteCerrar pasándole la id del puesto que se quiere cerrar. Este hilo se conectará con el servidor y este se encargará de cerrar el puesto y de lanzar una interrupción al sanitario que estuviera allí para que salga del puesto en cuanto pueda. Si el sanitario resulta que está haciendo algo en ese momento, como vacunar a un paciente, terminará de hacer lo que esté haciendo antes de irse a la sala de descanso.

Luego, centrándonos más en las herramientas que he usado, las de concurrencia han sido monitores dentro de las funciones del servidor para que los datos se fueran actualizando de forma ordenada e interrupciones para que los sanitarios dejen de esperar pacientes y salgan hacia la sala de descanso. Las herramientas de comunicación que he empleado para que cliente y servidor se conecten han sido sockets y serverSockets principalmente.

Por último, comentar que he realizado pequeños cambios en todas las clases que tengan que actualizar algo en la interfaz, principalmente para llamar a la función actualizarVariable del servidor. También he hecho unas pocas modificaciones dentro de la clase Puesto y la clase SalaVacunacion.

3. CLASES PRINCIPALES

1ª Parte:

Auxiliar: Esta clase es un hilo que se emplea para simular las acciones de uno de los auxiliares según su id. También guarda sus datos.

Atributos:

- Hospital hospital: Guarda el hospital al que pertenece
- String id: Guarda el id del auxiliar.

Sanitario: Esta clase es un hilo que se emplea para simular las acciones de un sanitario. También guarda sus datos.

Atributos:

- Hospital hospital: Guarda el hospital al que pertenece
- String idSanitario: Guarda el id del sanitario.
- Puesto puesto: Guarda el puesto al que el sanitario ha sido asignado

Paciente: Esta clase es un hilo que se emplea para simular las acciones de un paciente. También guarda sus datos. La probabilidad de venir sin cita se calcula en el constructor.

Atributos:

- Hospital hospital: Guarda el hospital al que va
- String id: Guarda el id del paciente.
- Boolean citado: Indica si el paciente está citado.
- Boolean vacunado: Indica si el paciente está vacunado.
- Boolean reacción: Indica si el paciente ha tenido una reacción.
- int porcentajeCitado: Indica el porcentaje de que el paciente no esté citado
- int porcentajeReaccion: Indica el porcentaje de que al paciente le de una reacción.
- Puesto puestoVacunacion: Indica el puesto que le ha tocado de la sala de vacunación.
- Puesto Observación: Indica el puesto que le ha tocado de la sala de observación.

Hospital: Esta clase guarda todas las salas del hospital y tiene los métodos necesarios para acceder a estas salas. Esta clase se emplea sobretodo como puente entre las personas y las salas. También sirve para imprimir mensajes en el log.

Atributos:

- Recepción recepción: Guarda la recepción del hospital.
- SalaDescanso salaDescanso: Guarda la sala de descanso del hospital.
- SalaVacunacion salaVacunacion: Guarda la sala de vacunación del hospital.
- SalaObservacion salaObservacion: Guarda la sala de observación del hospital.
- Log log: Guarda el log.

Métodos relevantes:

- void imprimirMensaje: Sirve para imprimir el mensaje que se le pase como atributo en el log. Se le puede pasar un booleano si el mensaje también se quiere imprimir por pantalla.
- Múltiples métodos que sirven solo para llamar al método adecuado de la sala correspondiente.

Recepción: Sirve para que los pacientes sean atendidos por un auxiliar y les diga un puesto de la sala de vacunación.

Atributos:

- JTextArea colaInterfaz: Sirve para imprimir por la interfaz la cola de la recepción.

- JTextField pacienteInterfaz: Sirve para imprimir por la interfaz al paciente al que esté atendiendo el auxiliar.
- JTextField auxiliarInterfaz: Sirve para imprimir por la interfaz el A1.
- Hospital hospital: Guarda el hospital.
- Queue<Paciente> colaEspera: Actúa como cola para que los pacientes entren.
- SynchronousQueue<Paciente> esperaAlAuxiliar: Sirve para que el auxiliar espere los datos del paciente.
- SynchronousQueue<Puesto> esperaPuesto:Sirve para que el paciente espere el puesto que le de el auxiliar.
- Semaphore atenderPaciente: Se usa para que los pacientes entren ordenadamente a la recepción.

Métodos relevantes:

- void añadirCola: Sirve para que los pacientes se vayan añadiendo a la cola de la recepción. También le sirve al paciente que esté en recepción para esperar la respuesta del auxiliar. Se le pasa como parámetro a la persona que vaya a entrar a la cola.
- void atenderPaciente: Sirve para que el auxiliar pueda atender a los clientes. Se le pasa al auxiliar como parámetro.
- void actualizarColaInterfaz: Sirve para actualizar la interfaz de la cola de recepción.

SalaVacunacion: Sirve para que los pacientes se vacunen en un puesto por un sanitario, para que el A2 cree vacunas y para que el A1 le busque un puesto a los pacientes.

Atributos:

- ArrayList<Puesto> puestos: Contiene los 10 puestos que hay en la sala de vacunación.
- JTextField auxiliarInterfaz: Sirve para mostrar la interfaz del A2
- JTextField vacunasInterfaz: Sirve para mostrar la interfaz de las vacunas disponibles.
- Object lockPuesto: Se usa como lock para cuando se quiere acceder a los puestos de forma sincronizada.
- Object lockVacunas: Se usa como lock para cuando se quiere acceder a las vacunas de forma sincronizada.
- Hospital hospital: Guarda el hospital.
- int vacunasDisponibles: Indica el número de vacunas disponibles
- Semaphore vacunasSemaforo: Sirve para que las vacunas vayan aumentando y disminuyendo de forma sincronizada.

Métodos relevantes:

- Puesto buscarPuestoPaciente: Lo usa el A1 para buscar en puesto disponible en la sala. Devuelve el puesto que encuentre disponible.
- void asignarPuestoSanitario: Sirve para que los sanitarios se asignen un puesto. Se le pasa el sanitario que vaya a ser asignado a un puesto como parámetro.

- void entrarSala: Sirve para que los pacientes entren, se vacunen y salgan de la sala a la que los hayan asignado. Se le pasa al paciente como parámetro.
- void atenderPuesto: Les sirve a los sanitarios para atender a los pacientes. Aquí se emplea una variable numPacientes que indica cuántos pacientes atenderá un sanitario antes de irse a descansar. Se le pasa el sanitario como parámetro.
- void crearVacunas: Este método lo utiliza el A2 para ir creando las vacunas de forma sincronizada.

SalaObservación: Sirve para que los pacientes esperen a ver si tienen alguna reacción por la vacuna.

Atributos:

- boolean haySanitarios: Le sirve al SAux para saber si han venido sanitarios hace poco.
- ArrayList<Puesto> puestos: Guarda los 20 puestos que hay en la sala
- Hospital hospital: Guarda el hospital
- Semaphore puestosSemaforo: Sirve para que solo haya 20 pacientes a la vez en la sala.
- Object lockPuesto: Se usa como lock para cuando se quiere acceder a los puestos de forma sincronizada.

Métodos relevantes:

- void entrarSala: Les sirve a los pacientes para entrar o salir de la sala. Aquí también se calcula si el paciente le da una reacción. Se le pasa el paciente como parámetro.
- void observarPacientes: Les sirve a los sanitarios para observar a los pacientes que estén en la sala y atenderlos en caso de que hayan tenido una reacción. Se le pasa al sanitario como parámetro.
- void vigilarSala: Le sirve al SAux para vigilar si los sanitarios van observando a los pacientes. Se le pasa al sanitario como parámetro.

SalaDescanso: Sirve para que los trabajadores del hospital descansen.

Atributos:

- TextArea colaInterfaz: Sirve para mostrar por la interfaz las personas dentro de la sala.
- ArrayList<Thread> personasDentro: Guarda las personas que estén dentro de la sala.
- Object lock: Se usa como lock para que las personas se vayan metiendo de forma sincronizada al arrayList.
- Hospital hospital: Guarda el hospital.

Métodos relevantes:

- void descansar: Sirve para que las personas descansen un tiempo determinado y después se vayan. Se les pasa como parámetros el hilo que vaya a descansar y el tiempo mínimo y máximo que vayan a estar descansando.

Puesto: Se usa para representar los datos de un puesto de la sala de vacunación o de observación

Atributos:

- JTextField puestoInterfaz: Sirve para mostrar por la interfaz las personas dentro del puesto.
- String idPuesto: Guarda el id del puesto
- Sanitario sanitarioAsignado: Guarda, si lo hubiera, el sanitario que esté dentro del puesto.
- Paciente pacienteAtendido: Guarda, si lo hubiera, el paciente que esté dentro del puesto.
- boolean abierto: Indica si el puesto está abierto.
- SynchronousQueue<Paciente> esperaPaciente: Se usa para que el sanitario espere los datos de un paciente.
- SynchronousQueue<Boolean> esperaVacuna: Se usa para que el paciente espere a ser vacunado
- SynchronousQueue<Boolean> esperaReaccion: Se usa para que un paciente con reacción espere a que le atienda un sanitario.

Métodos relevantes:

- void esperarVacuna: Lo usa el paciente para esperar a que le vacunen. Se le pasa como parámetro el paciente.
- Paciente esperarPaciente: Lo usa el sanitario para esperar a un paciente. Devuelve al paciente que entre al puesto.
- void esperarReaccion: Lo usan los pacientes con reacción para esperar que les atienda un sanitario. Se le pasa como parámetro el paciente.
- void ponerVacuna: Lo usa el sanitario para ponerle la vacuna al paciente que esté esperando en el puesto.
- void atenderReaccion: Lo usa el sanitario para atender al paciente con reacción que esté esperando.
- boolean estaDisponible: Devuelve si el puesto está disponible o no para que llegue un nuevo paciente.
- void actualizarInterfaz: Actualiza la interfaz del puesto.

A parte de estas clases que son las principales, existen otras tres que utiliza el programa. La clase log donde se crea el log y se imprimen los mensajes de los procesos del programa, la clase simulacionHospital que sirve para crear e inicializar los hilos y la clase Interfaz que es la que muestra la interfaz por pantalla.

2ª Parte:(Sólo las clases nuevas o que tengan variable o métodos nuevos)

Servidor: Se encarga de dar servicio a los clientes que se conecten para hacer la función de actualizar la interfaz o de cerrar un puesto.

Atributos:

- ServerSocket servidor: Sirve para que los clientes se puedan conectar.
- Socket conexión: Sirve para obtener la conexión del cliente
- DataOutputStream salida: Nos permite mandar información al cliente

- DataInputStream entrada: Nos permite recibir información del cliente.
- Hospital hospital: Guarda el hospital.
- Varias variables String que guardan la información de cada parte de la interfaz.

Métodos relevantes:

- void run: Sirve para conectarse con los clientes y realizar las función que les pidan.
- void enviarDatosHospital: Le envía los datos de la interfaz de manera ordenada al cliente.
- void actualizarVariable: Sirve para actualizar la parte de la interfaz correspondiente.

InterfazCliente: Muestra la interfaz del cliente y genera los hilos ClienteActualizar y ClienteCerrar.

Atributos:

- ExecutorService ex: Un pool de hilos para controlar la cantidad de hilos que se generan con los botones de cerrar puesto.

Métodos relevantes:

- void empezarProblema: Crea e inicializa el hilo ClienteActualizar.

ClienteActualizar: Sirve para que el cliente se conecte al servidor y le pida los datos actualizados de la interfaz.

Atributos:

- String mensaje:Indica la acción que le va a enviar el cliente al servidor para que haga.
- Socket conexión: Sirve para obtener la conexión del servidor
- DataOutputStream salida: Nos permite mandar información al servidor.
- DataInputStream entrada: Nos permite recibir información del servidor.
- Varias variables con elementos de la interfaz para actualizarlos con lo que nos mande el servidor.

Métodos relevantes:

- Void run:Inicia la conexión con el servidor.
- void actualizarInterfaz: actualiza todos los elementos de la interfaz con la información que envía el servidor.

ClienteCerrar:

Atributos:

- String mensaje:Indica la acción que le va a enviar el cliente al servidor para que haga.
- String nombrePuesto: Indica el id del puesto que se va a cerrar.
- Socket conexión: Sirve para obtener la conexión del servidor
- DataOutputStream salida: Nos permite mandar información al servidor.

- DataInputStream entrada: Nos permite recibir información del servidor.

Métodos relevantes:

- Void run:Inicia la conexión con el servidor.

Hospital:

Atributos:

- Servidor servidor: Guarda el servidor

Puesto:

Atributos:

- boolean hayQueLimpiar: Indica si han mandado limpiar el puesto.
- boolean puestoPillado: Indica si algún cliente ha pillado algún puesto.

SalaVacunación:

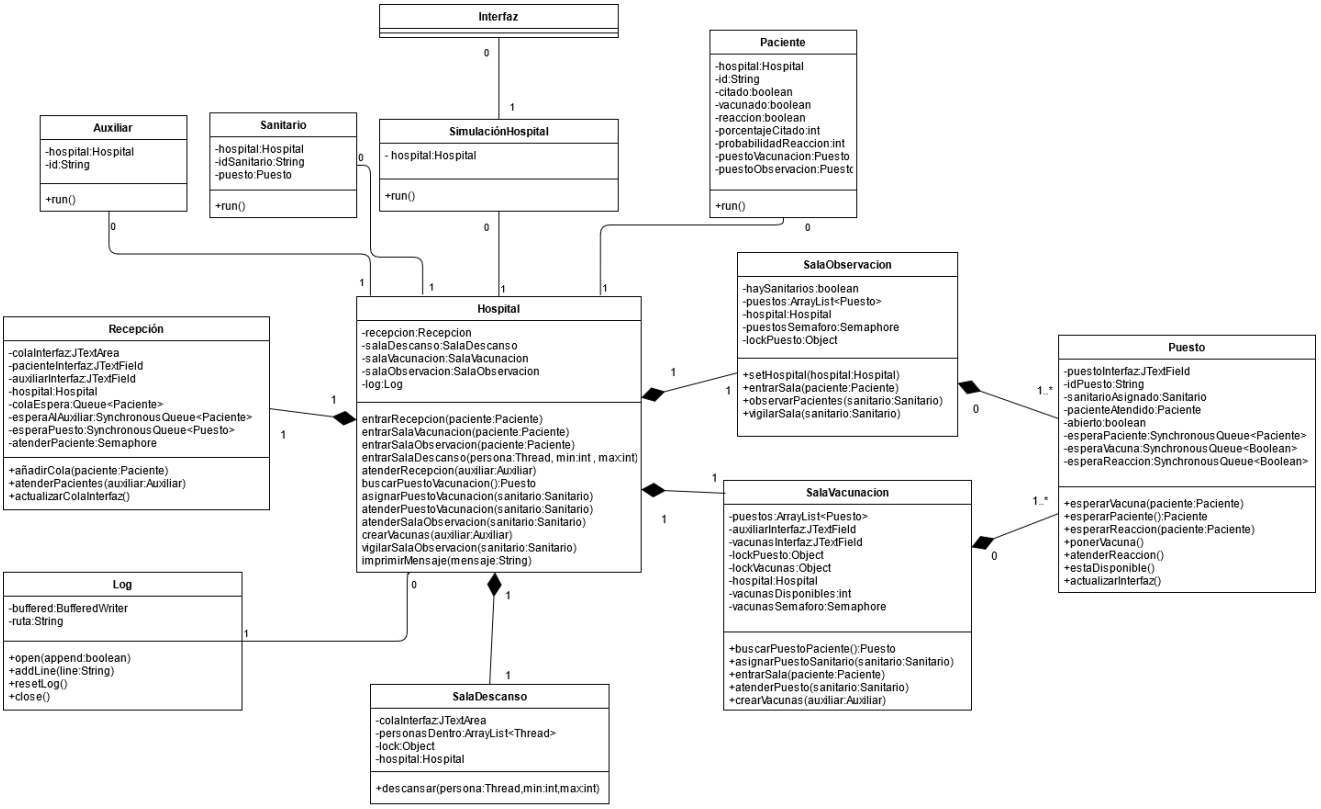
Métodos relevantes:

- void cerrarPuesto: Lo usa el servidor para cerrar un puesto específico.

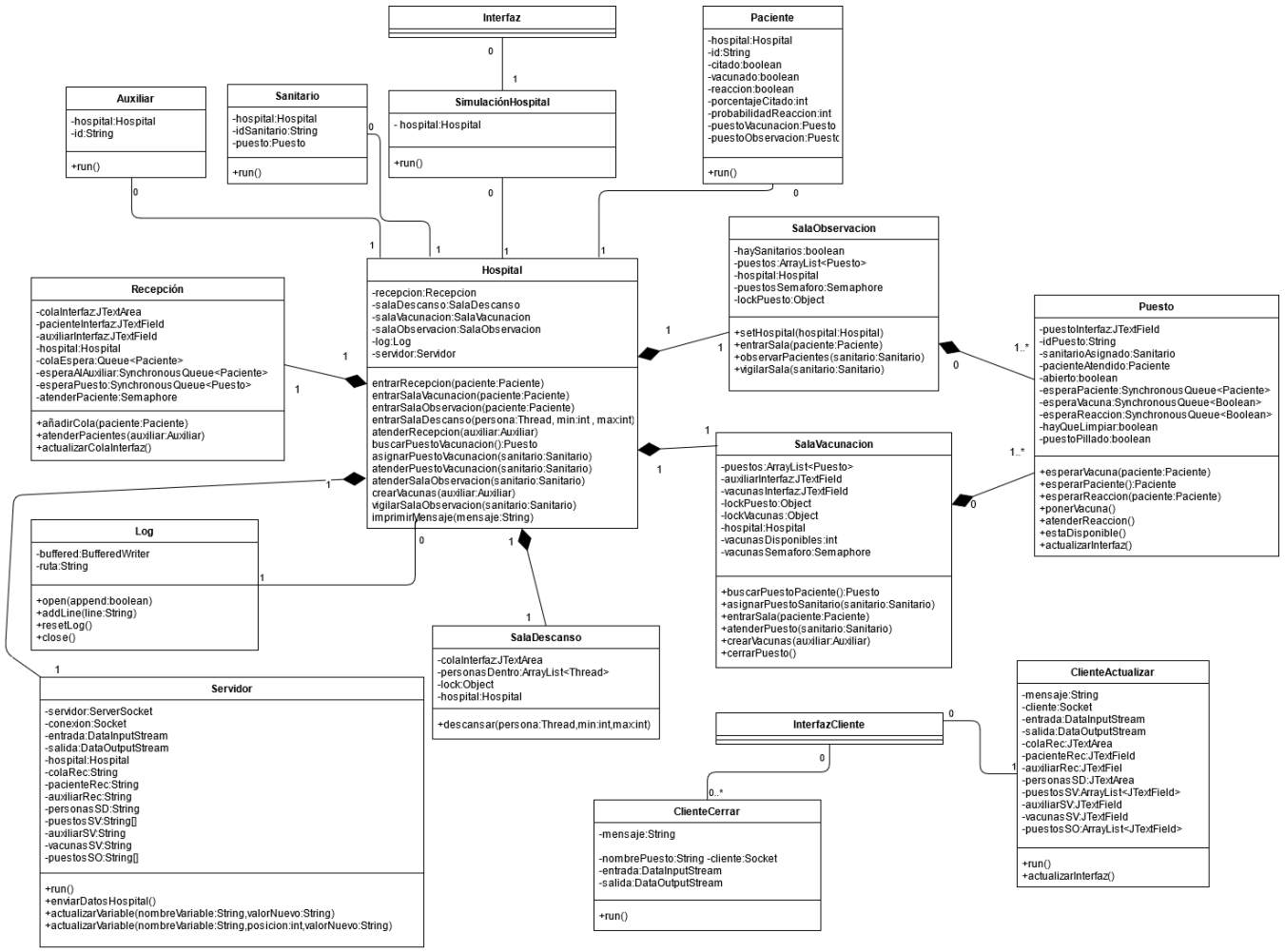
4. DIAGRAMA DE CLASES

1ª Parte:

En este diagrama de clases se obvian los métodos getter, setter o toString que puedan tener las clases.



2ª Parte:



5. ANEXO

1ª Parte:

Clase Auxiliar:

```java
package simulacionhospital;

import java.util.logging.Level;
import java.util.logging.Logger;

public class Auxiliar extends Thread {

    private Hospital hospital;
    private String id;

    public Auxiliar(Hospital hospital, String id) {
        this.hospital = hospital;
        this.id = id;
    }

    @Override
    public void run() {
        while (true) {
            if (id.equals("A1")) {
                hospital.atenderRecepcion(this);
                hospital.entrarSalaDescanso(this, 3000, 5000);
            } else if(id.equals("A2")){
                hospital.crearVacunas(this);
                hospital.entrarSalaDescanso(this, 1000, 4000);
            }
        }

    }

    public String getIdAuxiliar() {
        return id;
    }

    @Override
    public String toString() {
        return id;
    }


}
```

Clase Sanitario:

```java
package simulacionhospital;

import java.util.logging.Level;
import java.util.logging.Logger;

public class Sanitario extends Thread {

    private Hospital hospital;
    private String idSanitario;
    private Puesto puesto = null;

    public Sanitario(Hospital hospital, String id) {
        this.hospital = hospital;
        this.idSanitario = id;
    }

    @Override
    public void run() {
        hospital.entrarSalaDescanso(this, 1000, 3000);
        //Este sanitario se encargará de atender a los pacientes reaccionados si los otros tardan mucho en venir
        if (idSanitario.equals("SAux")) {
            try {
                Thread.sleep(120000);//Tiempo que espera el sanitario para vigilar la sala de observación
            } catch (InterruptedException ex) {
                Logger.getLogger(Sanitario.class.getName()).log(Level.SEVERE, null, ex);
            }
            hospital.vigilarSalaObservacion(this);
        } else {
            hospital.asignarPuestoVacunacion(this);
            while (true) {
                hospital.atenderPuestoVacunacion(this);
                hospital.entrarSalaDescanso(this, 5000, 8000);
                hospital.atenderSalaObservacion(this);
            }
        }
    }

    public Puesto getPuesto() {
        return puesto;
    }

    public void setPuesto(Puesto puesto) {
        this.puesto = puesto;
    }

    public String getIdSanitario() {
        return idSanitario;
    }

    @Override
    public String toString() {
        return idSanitario;
    }

}
```

Clase Paciente:

```java
package simulacionhospital;

import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Paciente extends Thread {

    private Hospital hospital;
    private String id;
    private boolean citado = true;
    private boolean vacunado = false;
    private boolean reaccion = false;
    private int porcentajeCitado = 1;
    private int probabilidadReaccion = 5;
    private Puesto puestoVacunacion = null;
    private Puesto puestoObservacion = null;

    public Paciente(Hospital hospital, String id) {
        this.hospital = hospital;
        this.id = id;
        if (porcentajeCitado > Math.random() * 100) {
            citado = false;
        }
    }

    @Override
    public void run() {
        String mensaje = "";
        hospital.entrarRecepcion(this);
        if (puestoVacunacion != null) {
            hospital.entrarSalaVacunacion(this);

            mensaje = id + " se dirige a la sala de observacion";

            hospital.imprimirMensaje(mensaje);

            hospital.entrarSalaObservacion(this);
        }

        mensaje = "El paciente " + id + " se va del hospital";
        hospital.imprimirMensaje(mensaje);
    }

    public Puesto getPuestoVacunacion() {
        return puestoVacunacion;
    }

    public void setPuestoVacunacion(Puesto puestoVacunacion) {
        this.puestoVacunacion = puestoVacunacion;
    }

    public Puesto getPuestoObservacion() {
        return puestoObservacion;
    }

    public void setPuestoObservacion(Puesto puestoObservacion) {
        this.puestoObservacion = puestoObservacion;
    }

    public boolean isCitado() {
        return citado;
    }

    public String getIdPaciente() {
        return id;
    }

    public void setVacunado(boolean vacunado) {
```

```
69          this.vacunado = vacunado;
70      }
71
72      public int getProbabilidadReaccion() {
73          return probabilidadReaccion;
74      }
75
76      public boolean isReaccion() {
77          return reaccion;
78      }
79
80      public void setReaccion(boolean reaccion) {
81          this.reaccion = reaccion;
82      }
83
84      @Override
86      public String toString() {
87          return id;
88      }
89  }
90
```

Clase Recepción:

```java
package simulacionhospital;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Queue;
import java.util.concurrent.ConcurrentLinkedQueue;
import java.util.concurrent.Semaphore;
import java.util.concurrent.SynchronousQueue;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JTextArea;
import javax.swing.JTextField;

public class Recepcion {

    private JTextArea colaInterfaz;
    private JTextField pacienteInterfaz;
    private JTextField auxiliarInterfaz;
    private Hospital hospital;
    private final Queue<Paciente> colaEspera = new ConcurrentLinkedQueue<Paciente>();
    private SynchronousQueue<Paciente> esperaAlAuxiliar = new SynchronousQueue<>();
    private SynchronousQueue<Puesto> esperaPuesto = new SynchronousQueue<>();
    private Semaphore atenderPaciente = new Semaphore(1, true);

    public Recepcion(JTextArea colaInterfaz, JTextField pacienteInterfaz, JTextField auxiliarInterfaz) {
        this.colaInterfaz = colaInterfaz;
        this.pacienteInterfaz = pacienteInterfaz;
        this.auxiliarInterfaz = auxiliarInterfaz;
    }

    public void añadirCola(Paciente paciente) {
        synchronized (colaEspera) {
            colaEspera.offer(paciente);
            actualizarColaInterfaz();
        }
        try {
            atenderPaciente.acquire();
            synchronized (colaEspera) {
                colaEspera.poll();
                actualizarColaInterfaz();
            }

            pacienteInterfaz.setText(paciente.getIdPaciente());

            esperaAlAuxiliar.put(paciente);
            paciente.setPuestoVacunacion(esperaPuesto.take());
        } catch (InterruptedException ex) {
            String mensaje="El paciente "+paciente.getIdPaciente()+" pide disculpas por venir sin cita y se marcha";
            hospital.imprimirMensaje(mensaje, true);
        } finally {
            pacienteInterfaz.setText("");
            atenderPaciente.release();
        }
    }

    public void atenderPacientes(Auxiliar auxiliar) {
        Paciente paciente;
        Puesto puesto;
        String mensaje="";
        int numPacientes = 10;//Número de pacientes que atenderá el auxiliar antes de descansar
        auxiliarInterfaz.setText(auxiliar.getIdAuxiliar());
        for (int i = 0; i < numPacientes; i++) {
            try {
                paciente = esperaAlAuxiliar.take();
                Thread.sleep(500 + (int) (500 * Math.random()));//tiempo en revisar los datos del paciente
                if (paciente.isCitado()) {
                    puesto=hospital.buscarPuestoVacunacion();
                    esperaPuesto.put(puesto);
```

```
                    mensaje="Paciente " + paciente.getIdPaciente() + " será vacunado en el puesto"
                        + puesto.getIdPuesto() +
                            " por el sanitario " + puesto.getSanitarioAsignado().getIdSanitario();
                    hospital.imprimirMensaje(mensaje, true);
                } else {
                    paciente.interrupt();
                    mensaje="Paciente "+ paciente.getIdPaciente() + " ha acudido sin cita.";
                    hospital.imprimirMensaje(mensaje, true);
                }

            } catch (InterruptedException ex) {
                Logger.getLogger(Recepcion.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
        auxiliarInterfaz.setText("");
    }

    public synchronized void actualizarColaInterfaz() {
        colaInterfaz.setText(colaEspera.toString());
    }

    public void setHospital(Hospital hospital) {
        this.hospital = hospital;
    }
}
```

Clase SalaDescanso:

```
package simulacionhospital;

import java.io.IOException;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JTextArea;

public class SalaDescanso {

    private JTextArea colaInterfaz;
    private ArrayList<Thread> personasDentro = new ArrayList<>();
    private final Object lock = new Object();
    private Hospital hospital;

    public SalaDescanso(JTextArea colaInterfaz) {
        this.colaInterfaz = colaInterfaz;
    }

    public void descansar(Thread persona, int min, int max) {
        String mensaje="";
        synchronized (lock) {
            personasDentro.add(persona);
            colaInterfaz.setText(personasDentro.toString());

        }
        try {
            mensaje=persona.toString()+" comienza su descanso.";
            hospital.imprimirMensaje(mensaje);

            Thread.sleep(min + (int) ((max - min) * Math.random()));

            mensaje=persona.toString()+" termina su descanso.";
            hospital.imprimirMensaje(mensaje);
```

```
        } catch (InterruptedException ex) {
            Logger.getLogger(SalaDescanso.class.getName()).log(Level.SEVERE, null, ex);
        }

        synchronized (lock) {
            personasDentro.remove(persona);
            colaInterfaz.setText(personasDentro.toString());
        }
    }

    public void setHospital(Hospital hospital) {
        this.hospital = hospital;
    }
}
```

Clase SalaVacunación:

```java
package simulacionhospital;

import java.io.IOException;
import java.util.ArrayList;
import java.util.concurrent.Semaphore;
import java.util.concurrent.SynchronousQueue;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JTextField;

public class SalaVacunacion {

    private ArrayList<Puesto> puestos = new ArrayList<>();
    private JTextField auxiliarInterfaz;
    private JTextField vacunasInterfaz;
    private final Object lockPuesto = new Object();
    private final Object lockVacunas = new Object();
    private Hospital hospital;
    private int vacunasDisponibles = 0;
    private Semaphore vacunasSemaforo = new Semaphore(vacunasDisponibles, true);

    public SalaVacunacion(ArrayList<JTextField> puestosInterfaz, JTextField auxiliarInterfaz, JTextField vacunasInterfaz)
        for (int i = 1; i <= puestosInterfaz.size(); i++) {
            this.puestos.add(new Puesto("SVP" + i, puestosInterfaz.get(i - 1)));
        }
        this.auxiliarInterfaz = auxiliarInterfaz;
        this.vacunasInterfaz = vacunasInterfaz;
        vacunasInterfaz.setText(String.valueOf(vacunasDisponibles));
    }

    public Puesto buscarPuestoPaciente() {
        Puesto puesto = null;
        while (puesto == null) {
            for (Puesto p : puestos) {
                if (p.estaDisponible()) {
                    puesto = p;
                    break;
                }
            }
            if (puesto == null) {
                synchronized (lockPuesto) {
                    try {
                        lockPuesto.wait();
                    } catch (InterruptedException ex) {
                        Logger.getLogger(SalaVacunacion.class.getName()).log(Level.SEVERE, null, ex);
                    }
                }
            }

        }
        return puesto;
    }

    public synchronized void asignarPuestoSanitario(Sanitario sanitario) {
        String mensaje="";
        for (Puesto p : puestos) {
            if (p.getSanitarioAsignado() == null) {
                p.setSanitarioAsignado(sanitario);
                sanitario.setPuesto(p);

                mensaje="Al sanitario " + sanitario.getIdSanitario()
                        + " se le ha asignado el puesto " + p.getIdPuesto();
                hospital.imprimirMensaje(mensaje);

                break;
            }
```

```
67          }
68      }
69
70      public void entrarSala(Paciente paciente) {
71          paciente.getPuestoVacunacion().setPacienteAtendido(paciente);
72          paciente.getPuestoVacunacion().actualizarInterfaz();
73
74          paciente.getPuestoVacunacion().esperarVacuna(paciente);
75
76          paciente.getPuestoVacunacion().setPacienteAtendido(null);
77          paciente.getPuestoVacunacion().actualizarInterfaz();
78      }
79
80      public void atenderPuesto(Sanitario sanitario) {
81          Paciente paciente;
            String mensaje = "";
83          int numPacientes = 15;//Número de pacientes que atenderá el sanitario antes de descansar
84
85          sanitario.getPuesto().setSanitarioAsignado(sanitario);
86          sanitario.getPuesto().actualizarInterfaz();
87          sanitario.getPuesto().setAbierto(true);
88
89          synchronized (lockPuesto) {
90              lockPuesto.notifyAll();//Avisa al auxiliar que hay un sanitario disponible
91          }
92
93          for (int i = 0; i < numPacientes; i++) {
94              try {
95                  paciente = sanitario.getPuesto().esperarPaciente();
96
97                  mensaje=sanitario.getIdSanitario() + " va a por vacunas.";
98                  hospital.imprimirMensaje(mensaje);
99
```

```
                vacunasSemaforo.acquire();

                mensaje=sanitario.getIdSanitario() + " ha conseguido vacunas.";
                hospital.imprimirMensaje(mensaje);

                synchronized (lockVacunas) {
                    vacunasDisponibles--;
                    vacunasInterfaz.setText(String.valueOf(vacunasDisponibles));
                }
                Thread.sleep(3000 + (int) (2000 * Math.random()));
                sanitario.getPuesto().ponerVacuna();

                mensaje = "Paciente " + paciente.getIdPaciente() + " ha sido vacunado en el puesto"
                        + paciente.getPuestoVacunacion().getIdPuesto() + " por el sanitario " + sanitario.getIdSanitario()
                hospital.imprimirMensaje(mensaje);
            } catch (InterruptedException ex) {
                Logger.getLogger(SalaVacunacion.class.getName()).log(Level.SEVERE, null, ex);
            }

            synchronized (lockPuesto) {
                lockPuesto.notifyAll();//Avisa al auxiliar que hay un puesto libre
            }
        }

        sanitario.getPuesto().setSanitarioAsignado(null);
        sanitario.getPuesto().actualizarInterfaz();
        sanitario.getPuesto().setAbierto(false);

    }

    public void crearVacunas(Auxiliar auxiliar) {
        int numVacunas = 20;//Vacunas que hay que realizar antes de irse a descansar
        int tasaProduccion = 1;//Número de vacunas elaboradas en cada ronda
        auxiliarInterfaz.setText(auxiliar.getIdAuxiliar());
        while (numVacunas > 0) {
            try {
                Thread.sleep(500 + (int) (500 * Math.random()));
            } catch (InterruptedException ex) {
                Logger.getLogger(SalaVacunacion.class.getName()).log(Level.SEVERE, null, ex);
            }

            for (int i = 0; i < tasaProduccion; i++) {
                synchronized (lockVacunas) {
                    vacunasDisponibles++;
                    vacunasInterfaz.setText(String.valueOf(vacunasDisponibles));
                }

                vacunasSemaforo.release();
                numVacunas--;
            }
        }
        auxiliarInterfaz.setText("");
    }

    public void setHospital(Hospital hospital) {
        this.hospital = hospital;
    }

}
```

Clase SalaObservacion:

```java
package simulacionhospital;

import java.io.IOException;
import java.util.ArrayList;
import java.util.concurrent.Semaphore;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JTextField;

public class SalaObservacion {

    private boolean haySanitarios = false;
    private ArrayList<Puesto> puestos = new ArrayList<>();
    private Hospital hospital;
    private Semaphore puestosSemaforo = new Semaphore(20, true);
    private final Object lockPuesto = new Object();

    public SalaObservacion(ArrayList<JTextField> puestosInterfaz) {
        for (int i = 1; i <= puestosInterfaz.size(); i++) {
            this.puestos.add(new Puesto("SOP" + i, puestosInterfaz.get(i - 1)));
        }
    }

    public void setHospital(Hospital hospital) {
        this.hospital = hospital;
    }

    public void entrarSala(Paciente paciente) {
        String mensaje="";
        try {
            puestosSemaforo.acquire();
            synchronized (lockPuesto) {
                for (Puesto p : puestos) {
                    if (p.getPacienteAtendido() == null) {
                        p.setPacienteAtendido(paciente);
                        paciente.setPuestoObservacion(p);
                        paciente.getPuestoObservacion().actualizarInterfaz();
                        break;
                    }
                }
            }

            Thread.sleep(10000);

            if (paciente.getProbabilidadReaccion() > Math.random() * 100) {
                paciente.setReaccion(true);

                mensaje="El paciente "+paciente.getIdPaciente()+" ha tenido una reacción";
                hospital.imprimirMensaje(mensaje);

                paciente.getPuestoObservacion().esperarReaccion(paciente);
            }

            synchronized (lockPuesto) {
                paciente.getPuestoObservacion().setPacienteAtendido(null);
                paciente.getPuestoObservacion().actualizarInterfaz();
            }
        } catch (InterruptedException ex) {
            Logger.getLogger(SalaObservacion.class.getName()).log(Level.SEVERE, null, ex);
        } finally {
            puestosSemaforo.release();
        }
    }

    public void observarPacientes(Sanitario sanitario) {
        Puesto puestoPaciente;
```

```
67      haySanitarios = true;
68      do {
69          puestoPaciente = null;
70          synchronized (lockPuesto) {
71              for (Puesto p : puestos) {
72                  if (p.getPacienteAtendido() != null && p.getPacienteAtendido().isReaccion()
73                          && p.getSanitarioAsignado() == null) {
74                      p.setSanitarioAsignado(sanitario);
75                      p.actualizarInterfaz();
76                      puestoPaciente = p;
77                      break;
78                  }
79              }
80          }
81
82          if (puestoPaciente != null) {
83              try {
84                  Thread.sleep(2000 + (int) (3000 * Math.random()));
85                  hospital.getLog().addLine("La reacción del Paciente "+puestoPaciente.getPacienteAtendido().getIdPaciente()+
86                          " ha sido atendida por el sanitario "+sanitario.getIdSanitario());
87              } catch (InterruptedException ex) {
88                  Logger.getLogger(SalaObservacion.class.getName()).log(Level.SEVERE, null, ex);
89              } catch (IOException ex) {
90                  Logger.getLogger(Recepcion.class.getName()).log(Level.SEVERE, null, ex);
91              }
92              puestoPaciente.atenderReaccion();
93              synchronized (lockPuesto) {
94                  puestoPaciente.setSanitarioAsignado(null);
95                  puestoPaciente.actualizarInterfaz();
96              }
97          }
98      } while (puestoPaciente != null);
99
100     }
101
102     public void vigilarSala(Sanitario sanitario) {
103         if (!haySanitarios) {
104             observarPacientes(sanitario);
105         }
106
107         haySanitarios = false;
108     }
109 }
110
```

Clase Puesto:

```
1   package simulacionhospital;
2
3   import java.util.Arrays;
4   import java.util.concurrent.SynchronousQueue;
5   import java.util.logging.Level;
6   import java.util.logging.Logger;
7   import javax.swing.JTextField;
8
9   public class Puesto {
10
11      private JTextField puestoInterfaz;
12      private String idPuesto;
13      private Sanitario sanitarioAsignado = null;
14      private Paciente pacienteAtendido = null;
15      private boolean abierto = false;
16      private SynchronousQueue<Paciente> esperaPaciente = new SynchronousQueue<>();
17      private SynchronousQueue<Boolean> esperaVacuna = new SynchronousQueue<>();
18      private SynchronousQueue<Boolean> esperaReaccion = new SynchronousQueue<>();
19
20      public Puesto(String idPuesto, JTextField puestoInterfaz) {
21          this.puestoInterfaz = puestoInterfaz;
22          this.idPuesto = idPuesto;
23      }
24
25      public void esperarVacuna(Paciente paciente) {
26          try {
27              esperaPaciente.put(paciente);
28              paciente.setVacunado(esperaVacuna.take());
29          } catch (InterruptedException ex) {
30              Logger.getLogger(SalaVacunacion.class.getName()).log(Level.SEVERE, null, ex);
31          }
32      }
33
34      public Paciente esperarPaciente(){
```

```
        Paciente paciente=null;
        try {
            paciente= esperaPaciente.take();
        } catch (InterruptedException ex) {
            Logger.getLogger(Puesto.class.getName()).log(Level.SEVERE, null, ex);
        }
        return paciente;
    }

    public void esperarReaccion(Paciente paciente){
        try {
            paciente.setReaccion(esperaReaccion.take());
        } catch (InterruptedException ex) {
            Logger.getLogger(SalaVacunacion.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public void ponerVacuna(){
        try {
            esperaVacuna.put(true);
        } catch (InterruptedException ex) {
            Logger.getLogger(Puesto.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public void atenderReaccion(){
        try {
            esperaReaccion.put(false);
        } catch (InterruptedException ex) {
            Logger.getLogger(Puesto.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public synchronized boolean estaDisponible() {
        boolean respuesta = true;
        if (!abierto || pacienteAtendido != null) {
            respuesta = false;
        }
        return respuesta;
    }

    public synchronized void actualizarInterfaz() {
        String[] frase = {"", ""};
        if (sanitarioAsignado != null) {
            frase[0] = sanitarioAsignado.getIdSanitario();
        }
        if (pacienteAtendido != null) {
            frase[1] = pacienteAtendido.getIdPaciente();
        }

        puestoInterfaz.setText(Arrays.toString(frase));
    }

    public Sanitario getSanitarioAsignado() {
        return sanitarioAsignado;
    }

    public void setSanitarioAsignado(Sanitario sanitarioAsignado) {
        this.sanitarioAsignado = sanitarioAsignado;
    }

    public synchronized Paciente getPacienteAtendido() {
        return pacienteAtendido;
    }

    public synchronized void setPacienteAtendido(Paciente pacienteAtendido) {
        this.pacienteAtendido = pacienteAtendido;
    }

    public synchronized boolean isAbierto() {
        return abierto;
    }

    public synchronized void setAbierto(boolean abierto) {
        this.abierto = abierto;
    }

    public String getIdPuesto() {
        return idPuesto;
    }

}
```

Clase Hospital:

```java
package simulacionhospital;

import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Hospital {

    private Recepcion recepcion;
    private SalaDescanso salaDescanso;
    private SalaVacunacion salaVacunacion;
    private SalaObservacion salaObservacion;
    private Log log;

    public Hospital(Recepcion recepcion, SalaDescanso salaDescanso,
            SalaVacunacion salaVacunacion, SalaObservacion salaObservacion) {
        this.recepcion = recepcion;
        this.salaDescanso = salaDescanso;
        this.salaVacunacion = salaVacunacion;
        this.salaObservacion = salaObservacion;
        recepcion.setHospital(this);
        salaDescanso.setHospital(this);
        salaVacunacion.setHospital(this);
        salaObservacion.setHospital(this);
        try {
            log = new Log("evolucionHospital.txt", true);
        } catch (IOException ex) {
            Logger.getLogger(Hospital.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public void entrarRecepcion(Paciente paciente) {
        recepcion.añadirCola(paciente);
    }

    public void entrarSalaVacunacion(Paciente paciente) {
        salaVacunacion.entrarSala(paciente);
    }

    public void entrarSalaObservacion(Paciente paciente) {
        salaObservacion.entrarSala(paciente);
    }

    public void entrarSalaDescanso(Thread persona, int min, int max) {
        salaDescanso.descansar(persona, min, max);
    }

    public void atenderRecepcion(Auxiliar auxiliar) {
        recepcion.atenderPacientes(auxiliar);
    }

    public Puesto buscarPuestoVacunacion() {
        Puesto puesto;
        puesto = salaVacunacion.buscarPuestoPaciente();
        return puesto;
    }

    public void asignarPuestoVacunacion(Sanitario sanitario) {
        salaVacunacion.asignarPuestoSanitario(sanitario);
    }

    public void atenderPuestoVacunacion(Sanitario sanitario) {
        salaVacunacion.atenderPuesto(sanitario);
    }

    public void atenderSalaObservacion(Sanitario sanitario) {
```

```
        salaObservacion.observarPacientes(sanitario);
    }

    public void crearVacunas(Auxiliar auxiliar) {
        salaVacunacion.crearVacunas(auxiliar);
    }

    public void vigilarSalaObservacion(Sanitario sanitario) {
        salaObservacion.vigilarSala(sanitario);
    }

    public Log getLog() {
        return log;
    }

    public void imprimirMensaje(String mensaje) {
        try {
            getLog().addLine(mensaje);
        } catch (IOException ex) {
            Logger.getLogger(Recepcion.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public void imprimirMensaje(String mensaje, boolean imprimirPantalla) {
        if(imprimirPantalla){
            System.out.println(mensaje);
        }
        try {
            getLog().addLine(mensaje);
        } catch (IOException ex) {
            Logger.getLogger(Recepcion.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
```

Clase Log:

```
package simulacionhospital;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;

public class Log {

    private BufferedWriter buffered;
    private String ruta;

    public Log(String ruta) throws IOException {
        this.ruta = ruta;
        this.open(true);
    }

    public Log(String ruta, boolean reset) throws IOException {
        this.ruta = ruta;
        this.open(!reset);
    }

    private synchronized void open(boolean append) throws IOException {
        this.buffered = new BufferedWriter(new FileWriter(this.ruta, append));
    }

    public synchronized void addLine(String line) throws IOException {

        SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/YYYY HH:mm:ss");
        String formatoFecha = sdf.format(new Date());
        this.open(true);
        this.buffered.write("[" + formatoFecha + "] " + line + "\n");
        this.close();
    }

    public synchronized void resetLog() throws IOException{
        this.open(false);
        this.close();
    }

    private synchronized void close() throws IOException{
        this.buffered.close();
    }

}
```

Clase SimulacionHospital:

```java
package simulacionhospital;

public class SimulacionHospital extends Thread {

    private Hospital hospital;

    public SimulacionHospital(Hospital hospital) {
        this.hospital = hospital;
    }

    @Override
    public void run() {
        Sanitario sanitario;
        Auxiliar auxiliar;
        Paciente paciente;

        for (int i = 1; i <= 10; i++) {
            sanitario = new Sanitario(hospital, "S" + String.format("%02d", i));
            sanitario.start();
        }
        sanitario = new Sanitario(hospital, "SAux");
        sanitario.start();

        for (int i = 1; i <= 2; i++) {
            auxiliar = new Auxiliar(hospital, "A" + i);
            auxiliar.start();
        }
        for (int i = 1; i <= 2000; i++) {
            paciente = new Paciente(hospital, "P" + String.format("%04d", i));
            paciente.start();
            try {
                Thread.sleep(1000 + (int) (2000 * Math.random()));
            } catch (Exception e) {
                System.out.println("Error en la espera entre pacientes");
            }
        }
    }
}
```

Clase Interfaz:(Solo la parte importante para la práctica)

```
1012     /**
1013      * @param args the command line arguments
1014      */
1015     public static void main(String args[]) {
1016         /* Set the Nimbus look and feel */
1017         Look and feel setting code (optional)
1038
1039         /* Create and display the form */
1040         java.awt.EventQueue.invokeLater(new Runnable() {
1041             public void run() {
1042                 Interfaz interfaz= new Interfaz();
1043                 interfaz.setVisible(true);
1044                 interfaz.empezarProblema();
1045             }
1046         });
1047     }
1048
1049     public void empezarProblema(){
1050         ArrayList<JTextField> puestosSalaVac=new ArrayList<>();
1051         ArrayList<JTextField> puestosSalaObs=new ArrayList<>();
1052         puestosSalaVac.add(SVP1);
1053         puestosSalaVac.add(SVP2);
1054         puestosSalaVac.add(SVP3);
1055         puestosSalaVac.add(SVP4);
1056         puestosSalaVac.add(SVP5);
1057         puestosSalaVac.add(SVP6);
1058         puestosSalaVac.add(SVP7);
1059         puestosSalaVac.add(SVP8);
1060         puestosSalaVac.add(SVP9);
1061         puestosSalaVac.add(SVP10);
1062         puestosSalaObs.add(SOP1);
1063         puestosSalaObs.add(SOP2);
1064         puestosSalaObs.add(SOP3);
1065         puestosSalaObs.add(SOP4);
1066         puestosSalaObs.add(SOP5);
1067         puestosSalaObs.add(SOP6);
1068         puestosSalaObs.add(SOP7);
1069         puestosSalaObs.add(SOP8);
1070         puestosSalaObs.add(SOP9);
1071         puestosSalaObs.add(SOP10);
1072         puestosSalaObs.add(SOP11);
1073         puestosSalaObs.add(SOP12);
1074         puestosSalaObs.add(SOP13);
1075         puestosSalaObs.add(SOP14);
1076         puestosSalaObs.add(SOP15);
1077         puestosSalaObs.add(SOP16);
1078         puestosSalaObs.add(SOP17);
1079         puestosSalaObs.add(SOP18);
1080         puestosSalaObs.add(SOP19);
1081         puestosSalaObs.add(SOP20);
1082
1083         Recepcion recepcion= new Recepcion(colaRecepcion,pacienteRecepcion,auxiliarRecepcion);
1084         SalaDescanso salaDescanso= new SalaDescanso(colaSalaDescanso);
1085         SalaVacunacion salaVacunacion= new SalaVacunacion(puestosSalaVac,SVAux,SVVacunas);
1086         SalaObservacion salaObservacion= new SalaObservacion(puestosSalaObs);
1087         Hospital hospital= new Hospital(recepcion,salaDescanso,salaVacunacion,salaObservacion);
1088         SimulacionHospital sh=new SimulacionHospital(hospital);
1089         sh.start();
1090     }
```

2ª Parte:(Sólo las clases nuevas y funciones que más hayan cambiado)

Clase Servidor:

```java
package simulacionhospitalparte2;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Servidor extends Thread {

    private ServerSocket servidor;
    private Socket conexion;
    private DataOutputStream salida;
    private DataInputStream entrada;

    private Hospital hospital;
    private String colaRec = "", pacienteRec = "", auxiliarRec = "";
    private String personasSD = "";
    private String[] puestosSV = new String[10];
    private String auxiliarSV = "", vacunasSV = "";
    private String[] puestosSO = new String[20];

    public Servidor(Hospital hospital) {
        try {
            servidor = new ServerSocket(5000); //Creamos un ServerSocketen el Puerto 5000
        } catch (IOException ex) {
            System.out.println("PROBLEMA AL CREAR EL SERVIDOR");
            Logger.getLogger(Servidor.class.getName()).log(Level.SEVERE, null, ex);
        }

        this.hospital=hospital;


        for (int i = 0; i < puestosSV.length; i++) {
            puestosSV[i] = "";
        }

        for (int i = 0; i < puestosSO.length; i++) {
            puestosSO[i] = "";
        }
    }

    @Override
    public void run() {
        try {
            while (true) {
                conexion = servidor.accept(); //Esperamos una conexión

                entrada = new DataInputStream(conexion.getInputStream()); //Abrimos los canales de E/S
                salida = new DataOutputStream(conexion.getOutputStream());

                String mensaje = entrada.readUTF(); //Leemos el mensaje del cliente

                if (mensaje.equals("Actualizar")) {
                    enviarDatosHospital();
                } else if (mensaje.equals("Cerrar")) {
                    String idPuesto=entrada.readUTF();
                    hospital.getSalaVacunacion().cerrarPuesto(idPuesto);
                }

                entrada.close();//Cerramos los flujos de entrada y salida
                salida.close();
                conexion.close(); //Y cerramos la conexión
            }
        } catch (IOException e) {
        }
    }
```

```
69
70      public synchronized void enviarDatosHospital() {
71          try {
72              salida.writeUTF(colaRec);
73              salida.writeUTF(pacienteRec);
74              salida.writeUTF(auxiliarRec);
75              salida.writeUTF(personasSD);
76              for (String puestoSV : puestosSV) {
77                  salida.writeUTF(puestoSV);
78              }
79              salida.writeUTF(auxiliarSV);
80              salida.writeUTF(vacunasSV);
81              for (String puestoSO : puestosSO) {
82                  salida.writeUTF(puestoSO);
83              }
84          } catch (IOException ex) {
85              Logger.getLogger(Servidor.class.getName()).log(Level.SEVERE, null, ex);
86          }
87      }
88
89      //Para las variables que son String
90      public synchronized void actualizarVariable(String nombreVariable, String valorNuevo) {
91          switch (nombreVariable) {
92              case "colaRec":
93                  colaRec = valorNuevo;
94                  break;
95              case "pacienteRec":
96                  pacienteRec = valorNuevo;
97                  break;
98              case "auxiliarRec":
99                  auxiliarRec = valorNuevo;
100                 break;
101             case "personasSD":
102                 personasSD = valorNuevo;
103                 break;
104             case "auxiliarSV":
105                 auxiliarSV = valorNuevo;
106                 break;
107             case "vacunasSV":
108                 vacunasSV = valorNuevo;
109                 break;
110         }
111     }
112
113     //Para las variables que son String[]
114     public synchronized void actualizarVariable(String nombreVariable, int posicion, String valorNuevo) {
115         switch (nombreVariable) {
116             case "puestosSV":
117                 puestosSV[posicion] = valorNuevo;
118                 break;
119             case "puestosSO":
120                 puestosSO[posicion] = valorNuevo;
121                 break;
122         }
123     }
124 }
125
```

Clase ClienteActualizar:

```java
package simulacionhospitalparte2;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.InetAddress;
import java.net.Socket;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JTextArea;
import javax.swing.JTextField;

public class ClienteActualizar extends Thread {

    private final String mensaje = "Actualizar";
    private Socket cliente;
    private DataInputStream entrada;
    private DataOutputStream salida;

    private JTextArea colaRec;
    private JTextField pacienteRec, auxiliarRec;
    private JTextArea personasSD;
    private ArrayList<JTextField> puestosSV;
    private JTextField auxiliarSV, vacunasSV;
    private ArrayList<JTextField> puestosSO;

    public ClienteActualizar(JTextArea colaRec, JTextField pacienteRec, JTextField auxiliarRec,
            JTextArea personasSD, ArrayList<JTextField> puestosSV,
            JTextField auxiliarSV, JTextField vacunasSV, ArrayList<JTextField> puestosSO) {
        this.colaRec = colaRec;
        this.pacienteRec = pacienteRec;
        this.auxiliarRec = auxiliarRec;
        this.personasSD = personasSD;
        this.puestosSV = puestosSV;
        this.auxiliarSV = auxiliarSV;
        this.vacunasSV = vacunasSV;
        this.puestosSO = puestosSO;
    }


    @Override
    public void run() {

        while (true) {
            try {
                Thread.sleep(1000);
                cliente = new Socket(InetAddress.getLocalHost(), 5000); //Creamos el socket para conectarnos al puerto 500
                entrada = new DataInputStream(cliente.getInputStream()); //Creamos los canales de E/S
                salida = new DataOutputStream(cliente.getOutputStream());

                salida.writeUTF(mensaje); //Enviamos un mensaje al servidor

                actualizarInterfaz();

                entrada.close(); //Cerramos los flujos de entrada y salida
                salida.close();
                cliente.close();//Cerramos la conexión
            } catch (IOException e) {
                System.out.println("Error: " + e.getMessage());
            } catch (InterruptedException ex) {
                Logger.getLogger(ClienteActualizar.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }

```

```
    public void actualizarInterfaz() {
        try {
            colaRec.setText(entrada.readUTF());
            pacienteRec.setText(entrada.readUTF());
            auxiliarRec.setText(entrada.readUTF());
            personasSD.setText(entrada.readUTF());

            for(JTextField puesto:puestosSV){
                puesto.setText(entrada.readUTF());
            }

            auxiliarSV.setText(entrada.readUTF());
            vacunasSV.setText(entrada.readUTF());

            for(JTextField puesto:puestosSO){
                puesto.setText(entrada.readUTF());
            }
        } catch (IOException ex) {
            System.out.println("Se perdio la conexión con el servidor.");
        }

    }
}
```

clase ClienteCerrar:

```
package simulacionhospitalparte2;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.InetAddress;
import java.net.Socket;
import java.util.logging.Level;
import java.util.logging.Logger;

public class ClienteCerrar implements Runnable {

    private final String mensaje = "Cerrar";
    private String nombrePuesto;
    private Socket cliente;
    private DataInputStream entrada;
    private DataOutputStream salida;

    public ClienteCerrar(String nombrePuesto) {
        this.nombrePuesto = nombrePuesto;
    }

    @Override
    public void run() {
            try {
                cliente = new Socket(InetAddress.getLocalHost(), 5000); //Creamos el socket para conectarnos al puerto 500
                entrada = new DataInputStream(cliente.getInputStream()); //Creamos los canales de E/S
                salida = new DataOutputStream(cliente.getOutputStream());

                salida.writeUTF(mensaje); //Enviamos un mensaje al servidor
                salida.writeUTF(nombrePuesto);//Le enviamos el puesto a cerrar

                entrada.close(); //Cerramos los flujos de entrada y salida
                salida.close();
                cliente.close();//Cerramos la conexión
            } catch (IOException e) {
                System.out.println("Error: " + e.getMessage());
            }
    }
}
```

clase InterfazCliente:(La parte relevante)

```
1129    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
1130        ex.execute(new ClienteCerrar("SVP1"));
1131    }
1132
1133    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
1134        ex.execute(new ClienteCerrar("SVP2"));
1135    }
1136
1137    private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
1138        ex.execute(new ClienteCerrar("SVP3"));
1139    }
1140
1141    private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
1142        ex.execute(new ClienteCerrar("SVP4"));
1143    }
1144
1145    private void jButton5ActionPerformed(java.awt.event.ActionEvent evt) {
1146        ex.execute(new ClienteCerrar("SVP5"));
1147    }
1148
1149    private void jButton6ActionPerformed(java.awt.event.ActionEvent evt) {
1150        ex.execute(new ClienteCerrar("SVP6"));
1151    }
1152
1153    private void jButton7ActionPerformed(java.awt.event.ActionEvent evt) {
1154        ex.execute(new ClienteCerrar("SVP7"));
1155    }
1156
1157    private void jButton8ActionPerformed(java.awt.event.ActionEvent evt) {
1158        ex.execute(new ClienteCerrar("SVP8"));
1159    }
1160
1161    private void jButton9ActionPerformed(java.awt.event.ActionEvent evt) {
1162        ex.execute(new ClienteCerrar("SVP9"));
1163    }
1164
1165    private void jButton10ActionPerformed(java.awt.event.ActionEvent evt) {
1166        ex.execute(new ClienteCerrar("SVP10"));
1167    }
1168
1169    /**
1170     * @param args the command line arguments
1171     */
1172    public static void main(String args[]) {
1173        /* Set the Nimbus look and feel */
1174        Look and feel setting code (optional)
1195        //</editor-fold>
1196        //</editor-fold>
1197        //</editor-fold>
1198
1199        /* Create and display the form */
            java.awt.EventQueue.invokeLater(new Runnable() {
1201            public void run() {
1202                InterfazCliente interfaz = new InterfazCliente();
1203                interfaz.setVisible(true);
1204                interfaz.empezarProblema();
1205            }
1206        });
1207    }
1208
1209    public void empezarProblema() {
1210        ArrayList<JTextField> puestosSalaVac = new ArrayList<>();
1211        ArrayList<JTextField> puestosSalaObs = new ArrayList<>();
1212        puestosSalaVac.add(SVP1);
1213        puestosSalaVac.add(SVP2);
```

```
1214            puestosSalaVac.add(SVP3);
1215            puestosSalaVac.add(SVP4);
1216            puestosSalaVac.add(SVP5);
1217            puestosSalaVac.add(SVP6);
1218            puestosSalaVac.add(SVP7);
1219            puestosSalaVac.add(SVP8);
1220            puestosSalaVac.add(SVP9);
1221            puestosSalaVac.add(SVP10);
1222            puestosSalaObs.add(SOP1);
1223            puestosSalaObs.add(SOP2);
1224            puestosSalaObs.add(SOP3);
1225            puestosSalaObs.add(SOP4);
1226            puestosSalaObs.add(SOP5);
1227            puestosSalaObs.add(SOP6);
1228            puestosSalaObs.add(SOP7);
1229            puestosSalaObs.add(SOP8);
1230            puestosSalaObs.add(SOP9);
1231            puestosSalaObs.add(SOP10);
1232            puestosSalaObs.add(SOP11);
1233            puestosSalaObs.add(SOP12);
1234            puestosSalaObs.add(SOP13);
1235            puestosSalaObs.add(SOP14);
1236            puestosSalaObs.add(SOP15);
1237            puestosSalaObs.add(SOP16);
1238            puestosSalaObs.add(SOP17);
1239            puestosSalaObs.add(SOP18);
1240            puestosSalaObs.add(SOP19);
1241            puestosSalaObs.add(SOP20);
1242
1243            ClienteActualizar cliente = new ClienteActualizar(colaRecepcion, pacienteRecepcion, auxiliarRecepcion,
1244                    colaSalaDescanso, puestosSalaVac, SVAux, SVVacunas, puestosSalaObs);
1245
1246            cliente.start();
```

Clase SalaVacunacion:

- método atenderPuesto:

```
82      public void atenderPuesto(Sanitario sanitario) {
83          Paciente paciente = null;
            String mensaje = "";
85          int numPacientes = 15;//Número de pacientes que atenderá el sanitario antes de descansar
86
87          sanitario.getPuesto().setSanitarioAsignado(sanitario);
88          sanitario.getPuesto().actualizarInterfaz(hospital);
89          sanitario.getPuesto().setHayQueLimpiar(false);
90          sanitario.getPuesto().setAbierto(true);
91
92          synchronized (lockPuesto) {
93              lockPuesto.notifyAll();//Avisa al auxiliar que hay un sanitario disponible
94          }
95
96          for (int i = 0; i < numPacientes; i++) {
97              try {
98                  paciente = sanitario.getPuesto().esperarPaciente();
99              } catch (InterruptedException ex) {
100             }
101
102             if (paciente != null) {
103                 int tiempoRandom = 3000 + (int) (2000 * Math.random());
104
105                 mensaje = sanitario.getIdSanitario() + " va a por vacunas.";
106                 hospital.imprimirMensaje(mensaje);
107
108                 boolean colaVacunaHecha = false;
109                 boolean vacunaConseguida = false;
110                 boolean pacienteVacunado = false;
111
112                 while (!pacienteVacunado) {
113                     if (paciente.isVacunado()) {
114                         break;
```

```
            }
            try {
                if (!colaVacunaHecha) {
                    vacunasSemaforo.acquire();
                    colaVacunaHecha = true;
                }
                if (!vacunaConseguida) {
                    synchronized (lockVacunas) {
                        vacunasDisponibles--;
                        vacunasInterfaz.setText(String.valueOf(vacunasDisponibles));
                        hospital.getServidor().actualizarVariable("vacunasSV", String.valueOf(vacunasDisponibles));
                    }
                    vacunaConseguida = true;
                }
                mensaje = sanitario.getIdSanitario() + " ha conseguido vacunas.";
                hospital.imprimirMensaje(mensaje);

                Thread.sleep(tiempoRandom);
                mensaje = "Paciente " + paciente.getIdPaciente() + " está a punto de ser vacunado en el puesto "
                        + paciente.getPuestoVacunacion().getIdPuesto() + " por el sanitario " + sanitario.getIdSanitario();
                hospital.imprimirMensaje(mensaje);
                sanitario.getPuesto().ponerVacuna();
                pacienteVacunado = true;
            } catch (InterruptedException ex) {
            }
        }

        mensaje = "Paciente " + paciente.getIdPaciente() + " ha sido vacunado en el puesto"
                + paciente.getPuestoVacunacion().getIdPuesto() + " por el sanitario " + sanitario.getIdSanitario();
        hospital.imprimirMensaje(mensaje);
    }

    if (sanitario.getPuesto().isHayQueLimpiar()) {
```

```
                mensaje = "Se ha cerrado el puesto " + sanitario.getPuesto().getIdPuesto() + " para limpiar";
                hospital.imprimirMensaje(mensaje);
                break;
            }

            synchronized (lockPuesto) {
                lockPuesto.notifyAll();//Avisa al auxiliar que hay un puesto libre
            }
        }

        sanitario.getPuesto().setSanitarioAsignado(null);
        sanitario.getPuesto().actualizarInterfaz(hospital);
        sanitario.getPuesto().setAbierto(false);

    }
```

- método cerrarPuesto:

```
    public void cerrarPuesto(String idPuesto) {
        for (Puesto p : puestos) {
            if (p.getIdPuesto().equals(idPuesto)) {
                if (!p.isHayQueLimpiar()) {
                    p.setHayQueLimpiar(true);
                    p.setAbierto(false);
                    if (p.getSanitarioAsignado() != null) {
                        p.getSanitarioAsignado().interrupt();
                    }
                }
                break;
            }
        }
    }

```


Design Preview [InterfazCliente]
RECEPCIÓN
Paciente
Auxiliar
SALA DE DESCANSO
SALA DE VACUNACIÓN
Puesto 1
Puesto 2
Puesto 3
Puesto 4
Puesto 5
CERRAR
CERRAR
CERRAR
CERRAR
CERRAR
Puesto 6
Puesto 7
Puesto 8
Puesto 9
Puesto 10
CERRAR
CERRAR
CERRAR
CERRAR
CERRAR
Auxiliar
Vacunas disponibles
SALA DE OBSERVACIÓN
Puesto 1
Puesto 2
Puesto 3
Puesto 4
Puesto 5
Puesto 6
Puesto 7
Puesto 8
Puesto 9
Puesto 10
Puesto 11
Puesto 12
Puesto 13
Puesto 14
Puesto 15
Puesto 16
Puesto 17
Puesto 18
Puesto 19
Puesto 20