**Simulación del funcionamiento de un hospital**

**Grado en Ingeniería de Computadores**

**Curso 2020/2021 – Convocatoria Ordinaria**

**DNI – Sánchez Librero, Fernando**

**03216738G – Montero Serrano, Adrián**

**ÍNDICE**

**-Análisis de alto nivel ………………………. Pág.3-4**

**-Diseño general del sistema**

**y herramientas**

**de sincronización usadas ………………… Pág. 5**

**-Clases principales ……………………………. Pág. 6-10**

**-Diagrama de clases …………………………. Pág. 10-14**

**-Anexo: Código fuente ………………………. Pág. 14-20**

ANÁLISIS DE ALTO NIVEL

Parte 1:

En la primera parte, se nos pide la creación de un programa que simule el funcionamiento de un hospital mediante programación concurrente, que constará de los siguientes casos:

-El hospital funcionará de manera automática, los clientes (hilos) entrarán en la recepción del hospital.

-En la recepción también encontraremos un auxiliar (hilo) que comprobará los datos de cada uno de los pacientes por orden de llegada.

-Cuando el auxiliar compruebe sus datos, el paciente pasará a una sala de vacunación libre, en la que previamente se encuentre un sanitario (hilo).

-La asignación de la sala de vacunación se realizará con métodos de sincronización para asegurarnos de que los pacientes realizan una sola acción ordenadamente.

-Una vez que el paciente se encuentre en el puesto de vacunación, el sanitario asignado a ese puesto vacunará al paciente.

-Otro auxiliar asignado a la sala de vacunación es el encargado de reponer las vacunas.

-Todos los trabajadores del hospital tienen descansos cuando completan un número de operaciones concretas. En este caso, los sanitarios cada 15 pacientes vacunados y los auxiliares, cada 10 pacientes y registrados y cada 20 vacunas repuestas respectivamente.

-Una vez el paciente ha sido vacunado, se le traslada a una sala de observación. Aquí se le asignará un puesto de observación de igual manera que se le asignaba un puesto de vacunación.

-Ya en el puesto, esperará un tiempo determinado para ver si existen reacciones a la vacuna.

-Si es así, el paciente se mantiene en el puesto de observación, hasta que un sanitario que vuelva del periodo de descanso sea notificado de la alerta y se dirija hacia dicho puesto. Allí el sanitario tratará al paciente y posteriormente abandonará el hospital.

-En caso de no encontrar ninguna reacción a la vacuna, el paciente abandonará el hospital.

Parte 2:

La segunda parte consiste en programar un acceso remoto sobre el hospital.

-El programa que habíamos creado, se convertirá en la parte “servidor” de esta conexión.

-Se creará otro programa que solicite acceso sobre nuestro programa Hospital principal.

-Una vez concedido el acceso, en la interfaz de este segundo programa se mostrará la misma información que vemos en el principal.

-Esta información será transmitida mediante conexión por puertos.

-Desde este segundo programa podremos interactuar con los puestos de vacunación pudiendo “cerrarlos” despachando así al sanitario y paciente que hubiese allí.

Diseño general del sistema y herramientas

de sincronización usadas

Para la primera parte de esta práctica hemos decidido usar principalmente listas bloqueantes (BlockingQueue) combinadas con semáforos, ya que dadas las necesidades del sistema eran los óptimos.

Dado que los pacientes, auxiliares y sanitarios son hilos, los valores compartidos a los que tienen que acceder los hemos creado como variables atómicas.

Las salas de vacunación y observación han sido desarrolladas como múltiples monitores Mesa para los hilos “Pacientes” y “Sanitarios”.

Los métodos usados por los hilos para interactuar, ya sea entre ellos o con otras instancias de objetos, son sincronizados. Esto evita que una operación sea realizada por dos hilos a la vez y conlleve un interbloqueo.

En cuanto a la interfaz, se actualiza automáticamente mediante los propios procesos que interactúan con el hospital, ya que, por cada ejecución deben esperar un tiempo determinado y esto se puede aprovechar como temporizador.

También hemos hecho uso de Locks para proteger las regiones críticas de código.

En la segunda parte, contamos con conexión por sockets y métodos sincronizados para acceder a ellos.

Una clase del programa principal desempeña la función de servidor, obteniendo todos los datos del Hospital a tiempo real.

Por otra parte, desde otro paquete del proyecto, otro programa simula un cliente, que encarga de establecer conexión con el servidor mediante dos puertos diferentes, por el primero obtendremos toda la información que será mostrada en la interfaz remota; con el segundo, interactuaremos con el hospital para poder cerrar los puestos de vacunación.

La transferencia de los datos se realiza con Data(Input/Output)Stream y Object(Input/Output)Stream.

El programa principal recogerá todas las acciones realizadas mediante un log (txt externo) tal y como se pide en el enunciado.

Clases principales

Por orden de creación, las clases principales son las siguientes:

**-Hospital**

La clase más importante de nuestro programa, ya que, desde aquí, tanto la interfaz como los diferentes hilos interactuaran con la información del sistema y entre ellos.

Esta clase funciona como constructor del objeto Hospital, pero a su vez servirá de “contenedor” para las diferentes colas de las salas que se encuentran en él, como por ejemplo la Cola de Recepción, o variables atómicas como el número de vacunas.

Prácticamente todos los hilos accederán y modificarán a información contenida en este objeto Hospital por medio de getters y setters.

En cuanto a métodos de sincronización también nos encontramos casi todos los Semaphores y BlockingQueue usados para dirigir a los hilos.

Esta clase también cuenta con métodos para recorrer dichas listas y liberarlas en caso necesario.

Por último, se encuentra la función que dará lugar al Log del sistema.

-**Puesto de Vacunación**

Clase constructora del objecto Puesto de Observación. Las instancias de este objeto serán nuestros monitores cuando al paciente se le asigne una sala de vacunación.

También cuenta con el método usado por los hilos “sanitario” para poner vacunas.

-**Puesto de Observación**

Al igual que Puesto de Vacunación, clase constructora del objeto del mismo nombre.

Esta clase solo cuenta con un método para insertar pacientes.

-**Auxiliar**

La clase auxiliar se utiliza para construir los dos hilos que actúan como auxiliares del hospital.

El primero, se encarga de “comprobar” los datos de los pacientes y dirigirlos hacia un puesto de vacunación libre. Cada 10 pacientes, el auxiliar 1 se toma un descanso, moviéndose a la Sala de Descanso, donde espera un tiempo determinado y después vuelve a su puesto para seguir comprobando pacientes. Aquí podemos encontrar el código correspondiente a la asignación de los pacientes en los puestos de vacunación

El segundo basa toda su actividad en la generación de vacunas para el hospital. La vacunas están representadas con un valor atómico, ya que a esta variable accederán también los hilos “Sanitarios” concurrentemente. Cada 20 vacunas generadas, toma un descanso en la Sala de Descanso y vuelve a su puesto.

-**Sanitario**

En esta clase se crean los hilos de tipo “sanitario”. En este caso 10 hilos que automáticamente se dispondrán de manera uniforme en los diferentes puestos de vacunación, a la espera de que entre un paciente para ser vacunado.

Una vez el contador interno de cada hilo haya llegado a 15 pacientes vacunados, éste tomará un descanso abandonando el puesto de vacunación e incorporándose a la Sala de descanso durante un tiempo determinado.

Una vez termina el descanso, cada sanitario realizará una iteración a todas las salas de observación en busca de algún paciente con reacción a la vacuna. En caso positivo, el sanitario se trasladará a dicho puesto de Observación y atenderá al paciente. Después, ambos abandonaran la sala de observación y el sanitario volverá a un puesto de vacunación vacío para continuar su trabajo. Si no encuentra ningún paciente con síntomas, buscará una sala vacía para seguir vacunando.

-**Paciente**

Clase para crear los hilos Paciente. Estos hilos son sobre los que se ejecutaran todos los procedimientos del hospital.

En una primera instancia, se van creando poco a poco y entran en la cola concurrente de espera del hospital de forma espaciada en el tiempo.

Mediante un BlockingQueue(1) y un semáforo se controla la entrada, un auxiliar es el encargado de “comprobar” los datos del paciente, existiendo una pequeña probabilidad de que este no estuviese citado. Una vez comprueba los datos, el paciente pasa a otra cola donde esperará a que se le asigne una mesa. Este proceso, se realiza también mediante una BlockingQueue(1) para insertar uno a uno los pacientes en los puestos vacíos (sin otro paciente en ellos y con un sanitario esperando). Estos puestos vacíos se adquieren de otra cola donde se depositan todos al comienzo de la ejecución, y cada vez que un paciente abandona un puesto, dicha sala de vacunación vuelve a la cola de puestos vacíos esperando ser seleccionada por el auxiliar uno para el próximo paciente.

Cuando el paciente se encuentra en el puesto de vacunación el sanitario ejecuta sobre él el método ponerVacuna, que decrementa la variable atómica vacunas.

Acto seguido, el sanitario despacha al paciente y lo redirige a una BlockingQueue que selecciona un puesto de observación libre, de igual manera que se hacía con los de vacunación.

Cuando el paciente se encuentra en una sala de Observación, esperará 10 segundos para verificar que no tiene ninguna reacción a la vacuna. Un 5% de los pacientes tendrán reacción. En este caso, los pacientes activan un flag de la sala y quedan a la espera de que un sanitario los atienda.

Después de que el sanitario llegue a dicha sala (una vez termine su jornada y haya descansado) , el paciente será tratado y el flag de la sala volverá a su estado inicial. El paciente se marchará de la sala de Observación y del hospital. Se dará por finalizada su ejecución y se destruirá el hilo.

-**ServidorSockets**

En esta clase se desarrolla el constructor del servidor por sockets, para transmitir la información servidor-cliente del hospital.

Se crean dos instancias de esta clase, la primera (puerto 5000) se ocupará de recibir las señales E/S de la interfaz Cliente.

Cuenta con una rutina de casos en los que dependiendo de que botón pulsemos, cerrará el puesto asignado. Liberando así al sanitario (lo manda a la sala de descanso) y paciente (lo expulsa del hospital) que hubiese en éste.

El segundo hilo (puerto 5001) se encarga de transmitir los datos del hospital hacia la interfaz, de manera que se puedan mostrar en pantalla remotamente. Se transmite toda la información actual mediante un ArrayList. Dentro de este podremos encontrar todos los Arrays o Strings correspondientes a cada uno de los Jtext de la interfaz, ya que solo buscamos representar los valores por pantalla.

Ambos procedimientos se realizan con llamadas a Data(Input/Output)Stream y Object(Input/Output)Stream.

-**Cliente**

Como contraparte del servidor, esta clase genera los hilos que se encargarán de solicitar la información del servidor realizando una conexión mediante sockets.

Este hilo se encarga de recoger la información recibida mediante el ArrayList y la proyecta hacia la interfaz, colocando así todos los datos. Repetirá esta acción cada segundo, simulando una actualización en tiempo real de la interfaz.

**----INTERFAZ GRÁFICA----**

-**Menú Principal (MAIN)**

  
 Interfaz Swing que cuenta con 4 para seleccionar las diferentes ventanas que tiene el programa.

Esta interfaz es a su vez el inicializador de todos los hilos e instancias, ya que posee la el Main del programa. Toda la interfaz está modularmente separada de la lógica para mayor simpleza a la hora de modificar los datos.

También se encarga de rellenar las colas de los puestos vacíos, pasando todos los valores recién creados por los constructores.

-**Recepción / salaDescanso / SalaVacunación / SalaObservación**

Estas clases cuentan solo con el diseño de la interfaz gráfica y los componentes contenedores donde se introduce la información, no tienen lógica asignada ya que de eso se encargan los hilos creados anteriormente.

Estas son las salas:

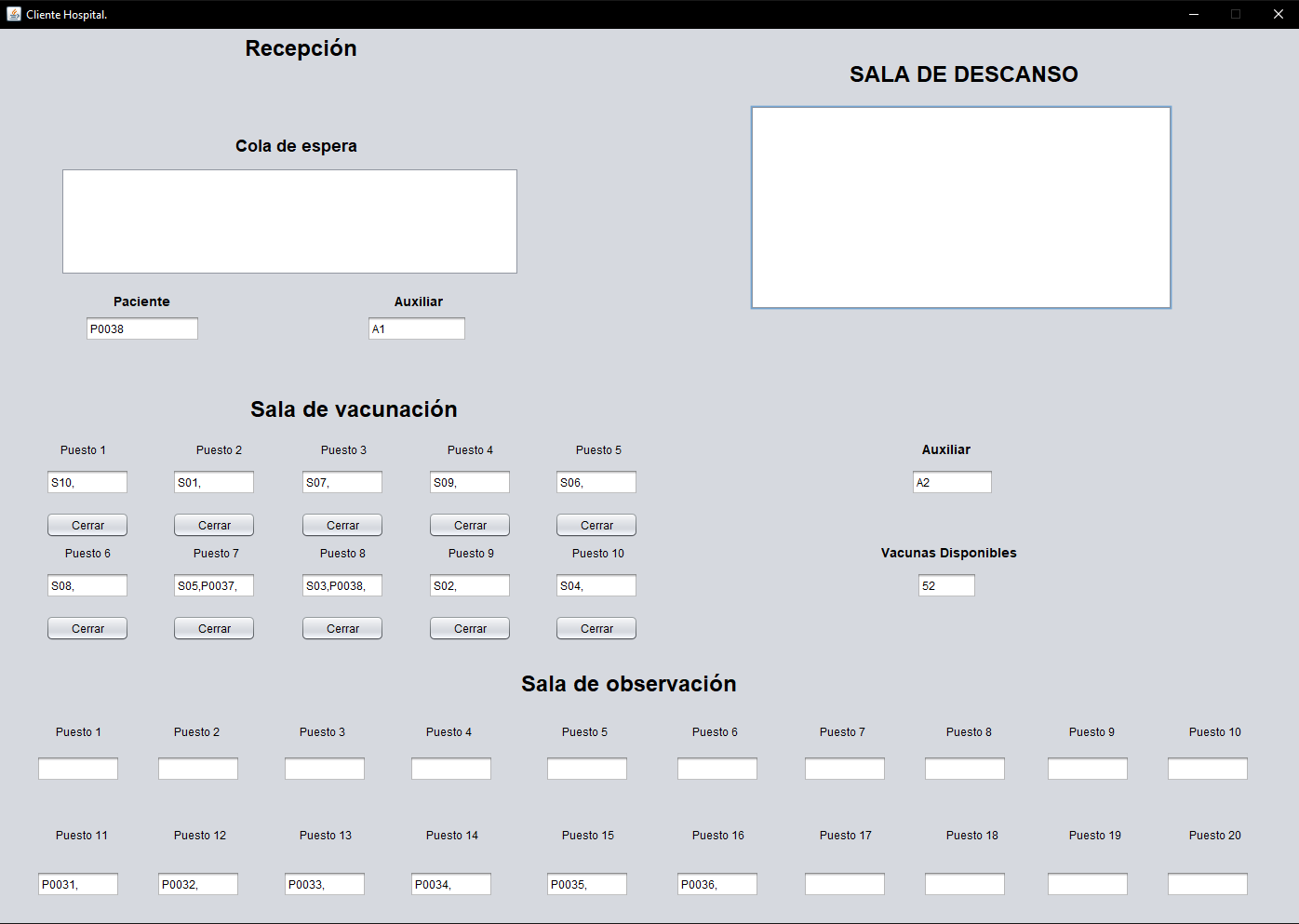




****

****

-InterfazCliente



**Diagrama de clases**

****

****