PROGRAMACIÓN AVANZADA

Práctica de Laboratorio – Hospital

Víctor Gamonal Sánchez

Laura Mambrilla Moreno

UNIVERSIDAD DE ALCALÁ
GRADO EN INGENIERÍA INFORMÁTICA
Curso 2019/20 – Convocatoria Extraordinaria

1. ANÁLISIS DE ALTO NIVEL

Este programa consiste en conseguir una simulación del tránsito de gente en un hospital; este consta de un total de 21 plantas (desde la 0 hasta la 20) y de 3 ascensores con una capacidad máxima de 8 personas cada uno que irán recorriendo cada una de estas plantas comprobando una a una si hay personas que desean subirse a él o si por el contrario hay gente en su interior que desea bajar en esa planta específica en la que se encuentra en ese momento, que será porque ha llegado a su destino.

Estas personas irán llegando de forma escalonada y hasta un total de 6.000 a lo largo de la ejecución del programa de forma aleatoria a cualquiera de las 21 plantas, donde pulsarán un botón y esperarán hasta que su llamada sea atendida por cualquiera de los ascensores que puedan atenderla. Estas personas cuentan con una planta origen, que es a la cual llegan, y una planta de destino, que es a la cual se dirigen, y por ende un sentido bien sea de subida o de bajada.

Los ascensores realizarán un máximo de 5.000 movimientos durante la ejecución del programa; de los tres existentes, sólo habrá dos en funcionamiento en todo momento y habrá un tercero que se encuentre parado hasta que uno de los dos anteriormente citados se estropee (evento que ocurrirá cada 20-30 segundos) y cuyo funcionamiento se verá reanudado segundos después.

Los ascensores cuentan con cuatro posibles estados: subiendo, bajando, parado y estropeado. En el caso de que estén subiendo o bajando, comprobarán en todo momento, como se ha mencionado anteriormente, que haya personas que deseen subirse a él o bajarse de él en cada una de las plantas; recogerán exclusivamente a aquellas personas cuyo sentido coincida con el de éste (es decir, mientras que haya aforo permitido en el ascensor y si éste está subiendo, por ejemplo, recogerá en una planta a aquellas personas que también deseen subir). Además, cuando uno de los ascensores se estropee, evacuará a todas las personas en su interior en la planta donde ha quedado inmovilizado, siendo posteriormente todas ellas atendidas tras pulsar el botón de llamada nuevamente en esa planta y el tercer ascensor que estaba en stand by entrará en acción.

Más allá de todo esto, se han implementado mediante un sistema distribuido dos módulos que permiten el acceso y control remoto para aquellos que lo deseen: se trata de un módulo de vigilancia y un módulo de evacuación.

En cuanto al módulo de evacuación (control), permite como su propio nombre indica evacuar a todas las personas que se encuentren en el interior del hospital (ya sea dentro del ascensor o esperando en planta) y no permite que lleguen más personas. Esta evacuación la realizarán los ascensores, quienes llevarán a todas ellas a la planta 0 del hospital hasta que no quede ninguna por desalojar.

En cuanto al módulo de vigilancia (observación), permite a quien lo desee ver el estado de los ascensores (sentido actual, personas dentro de él y sus destinos) y a todas las personas que hay esperando en cada una de las plantas en un momento dado.

2. DISEÑO GENERAL DEL SISTEMA Y DE LAS HERRAMIENTAS DE SINCRONIZACIÓN UTILIZADAS

El sistema cuenta con un diseño concurrente a través del cual múltiples tareas son realizadas a la vez.

El recurso compartido por todas estas tareas (o hilos) es el Hospital, inicializado en la clase principal dando lugar a su vez a la creación de los tres ascensores dentro de su constructor, como se detallará a continuación; además, cuenta con un Array plantas_Hospital con un total de 21 elementos que serán de tipo Planta y que simularán el hospital como tal, almacenando en cada uno de ellos un ArrayList de Personas que esperarán a ser atendidas.

La clase Planta recoge los atributos y métodos necesarios (detallados más adelante) para simular una planta de hospital.

El hilo que representa a los tres ascensores recibe el nombre de Ascensor. Estos ascensores, inicializados en el hospital previos a la creación de las personas, se encuentran inicialmente parados esperando a que éstas lleguen a las distintas plantas del hospital y pulsen los respectivos botones de llamada para atenderlas y ponerse en movimiento.

El hilo que representa a todas las personas recibe el nombre de Persona. Estas personas son creadas e inicializadas en la clase principal del programa, a las cuales se les asigna un objeto Hospital, un ID que las identifica a cada una de ellas y una planta origen de la cual parten que es asignada de forma aleatoria.

Existe además un hilo implementado por nosotros, EstropeaAscensor, que será el encargado de escoger uno de entre los tres ascensores para hacer que se estropee.

La principal herramienta de comunicación y sincronización utilizada durante la creación del sistema han sido los cerrojos (locks) además de haber hecho uso de métodos synchronized que también aseguran que no haya interferencia entre hilos y evitan posibles corrupciones de memoria. Hemos hecho uso de los cerrojos en aquellos métodos donde hay una sección crítica, es decir, aquel bloque de código donde puede ocurrir que varios hilos quieran acceder a la vez, que son sobre todo métodos GETTER y SETTER, de los cuales se recaba información en tiempo real (véase para incrementar el contador de movimientos realizados, donde puede ser que el valor corrompido pueda ser 23 mientras que el real sea 20).

El flujo del programa sería algo así:

Se instancia el Hospital y a su vez, veintiún Plantas en cada una de las posiciones del array "plantas_Hospital" que hemos comentado anteriormente y los tres distintos Ascensores que también son almacenados en un array "ascensores", así como se inicializa también el hilo EstropeaAscensores. Dentro de la clase Hospital contamos con un contador de movimientos que irá incrementando cada vez que un ascensor se mueva (es decir, suba o baje de planta) y un método clave (abierto()) de tipo boolean que devolverá true siempre que ese contador sea menor que el total de movimientos que los ascensores deben realizar (5.000).

El run() de Ascensor implementa los mecanismos necesarios para su correcto funcionamiento: esto es, comprueba en cuál de los 4 posibles estados se encuentra y dependiendo de cuál sea éste, hará una cosa u otra. En el caso específico de subida y bajada, comprobará que pueda meter personas (a aquellas cuyo sentido sea igual que el del ascensor y siempre que haya capacidad) y dejar personas (a aquellas cuyo destino sea igual a la planta en la que se encuentran en ese momento) y añadirá en 1 el contador de movimientos imprimiendo también por pantalla el resultado de ese movimiento; en el caso de estar parado, buscará llamadas que atender (pulsaciones de botones) y en el caso de estar estropeado sólo evacuará a aquellas personas que se encuentren en su interior. Todo esto lo hará mientras abierto() sea true (es decir, que queden movimientos por realizar).

Después de la creación de los objetos anteriores, entran en juego las personas, tratadas como hilos e inicializadas en la clase principal (main). Éstas tienen la única función de llegar a la planta del hospital que se le asigna de forma aleatoria, pulsar el botón que hay en ella para indicar que quiere ser transportada a su destino y esperar.

Por último, existen dos interfaces, cada una de ellas recogiendo la implementación necesaria para simular un módulo de vigilancia (mediante prints y getters) y un módulo de evacuación (que consiste en establecer el destino de todas las personas en el hospital a 0 para que sean evacuadas, interrumpir la creación de más personas y parar la ejecución de los hilos) como se recogía antes, todo ello mediante el uso de RMI.

3. CLASES PRINCIPALES, ATRIBUTOS Y MÉTODOS

El programa consta de 9 clases (una de elllas el main), una interfaz que extiende de Remote.

1. Ascensor

Esta clase es un hilo, representa a la figura del ascensor. Tiene los siguientes atributos (a parte de cerrojos y condiciones, necesarios para el correcto funcionamiento del programa):

- private final int id_ascensor → qué ascensor es. En este caso, al haber tres ascensores será 1, 2 o 3.
- private String estado → puede ser parado (P), bajando (B), subiendo (S) o estropeado (E)
- private final int capacidad → según el enunciado de la práctica, este dato es 8. Es decir, la capacidad máxima de cada ascensor es de 8 personas.
- private boolean esperando → indica si el ascensor está o no esperando (en funcionamiento o no)
- private int planta_actual → la planta en la que se encuentra el ascensor en cada momento.
- private final ArrayList<Persona> personas_en_ascensor → es un arrayList donde se almacenan a las personas que están en ese ascensor, quitándolas de este cuando se bajan de él.
- private final Hospital hospital → el hospital.

Cada uno de estos atributos tienen sus métodos y funciones de get y set, ya que serán modificados en otras clases.

Esta clase nos encontramos con los siguientes métodos para simular al máximo el comportamiento de un ascensor real:

- private void bajarDelAscensor() → cuando una persona se baja del ascensor se la elimina del arrayList personas_en_ascensor, por lo que ahora habrá un espacio más disponible.
- private void bajarAscensorEstropeado() → cuando un ascensor se estropea, todos sus ocupantes deben bajarse en la planta donde se ha estropeado, por lo que ha estas personas se las añaden donde se almacenan a las personas que están esperando en cada planta.
- private synchronized void subirAlAscensor() → se suben al ascensor aquellas personas cuyo sentido coincida con el ascensor (subir o bajar), siempre y cuando haya espacio suficiente dentro del ascensor. A la persona se le añade a personas_en_ascensor.
- private void cambioSentido () → el ascensor cambia de sentido cuando su estado es subiendo y llega a la planta 20, cuando su estado es bajando y llega

a la planta 0 y cuando se queda vacío (es decir, que está parado) y alguien en sentido opuesto ha llamado al ascensor.

- public void run() → mientras el hospital esté abierto, cada ascensor (es decir, cada hilo) dependiendo de su estado realizará una función y otra (subir o bajar personas entre otras cosas).

2. Control

En esta clase, que extiende de UnicastRemoteObject e implementa la interfaz Metodos, encontramos los métodos que se llevarán a cabo cuando cuando se pulsen los botones de los distintos módulos (vigilante y evacuación).

Como atributo tiene el hospital común al resto de clases, además de dos cerrojos. Y sus métodos son los siguientes:

- public void metodoModuloVigilante() throws RemoteException → se realizará cuando se pulse al botón correspondiente con la interfaz gráfica del módulo vigilante, imprimiendo por consola toda la información del hospital de ese justo momento, es decir: el estado de cada ascensor, el número de personas que lleva dentro cada ascensor con sus destinos y orígenes, cuántas personas hay esperando en cada planta y, también, sus orígenes y destinos. Al imprimirse en la misma consola donde se observa cada movimiento del ascensor, podría pasar desapercibido, por lo que al inicio y al final de este método se imprime una línea larga para que se vea mejor.
- public void metodoModuloEvacuacion() throws RemoteException → simula una evacuación, por lo que todos deben desalojar el hospital. Lo hacemos poniendo la variable booleana evacuacion de la clase Hospital a true y cambiando todos los destinos a 0, llamando al siguiente método.
- private void cambiarDestinoACero() → primero cambiamos el destino de todas las personas que se encuentran en los ascensores y después el de todas las personas que se encuentran en cada planta.

3. EstropeaAscensor

Esta clase, de tipo hilo, es la encargada de escoger al próximo ascensor que se estropeará.

En cuanto a sus atributos, no contiene, pues lo único destacable es que recibe como parámetro en su constructor el hospital al que pertenece.

En cuanto a sus métodos:

- private void controlarAscensores(int n) - Esta función es la encargada de establecer como estado “E” (estropeado) y poner a true el booleano de “esperando” del ascensor que recibe como parámetro; este parámetro es generado de manera aleatoria entre 1 y 3 (el número de ascensores) en el run() del propio hilo; esto se repetirá cada 20-30 segundos.

4. Hospital

Esta clase representa el recurso compartido por todos los hilos del programa.

En cuanto a sus atributos, asumiendo que cuenta con los correspondientes locks para controlar las SC:

- private int num_movimientos → el total de movimientos a realizar por los ascensores.
- private int cont_movimientos → contador de movimientos que incrementa en 1 cada vez que se mueve el ascensor.
- private int num_ascensores → total de ascensores que hay.
- private boolean evacuacion → su valor será inicialmente false y se pondrá a true permitiendo el cierre del hospital y su evacuación cuando se ejecute el módulo de evacuación.
- private Planta plantas_Hospital[] → un array con 21 elementos de tipo Planta que simula el hospital en sí.
- private Ascensor ascensores[] → array de 3 elementos donde almaceno los ascensores.
- private FileWriter archivo, BufferedWriter bw → para sacar toda la información por el archivo txt.

En cuanto a sus métodos:

- public void sumarContadorMovimientos() → para incrementar en uno el contador de movimientos e imprimir el estado actual del hospital.
- public boolean abierto() → controla que el hospital siga abierto (mientras que queden movimientos por hacer y no se haya evacuado el hospital), y una vez finalizado, se cerrará el archivo de salida.
- private synchronized boolean todasLasPlantasVacias() → comprueba, cuando se ejecuta la evacuación, que no quedan personas por evacuar en las plantas.
- private synchronized boolean todosLosAscensoresVacios() → comprueba, cuando se ejecuta la evacuación, que no quedan personas en el ascensor por evacuar.
- public synchronized void pulsarBoton(int planta_origen) → establece a true el botón de la planta que recibe como parámetro (pulsado).
- public synchronized void imprimir() → implementación para imprimir el estado del hospital (plantas, personas dentro, estado ascensores…). En este método también se escribe en archivo los mismo que va saliendo por pantalla a través de la consola.

5. Metodos

Esta clase sirve de nexo entre el Hospital y las interfaces de los módulos de vigilancia y evacuación.

- void metodoModuloVigilante() throws RemoteException
- void metodoModuloEvacuacion() throws RemoteException

6. Persona

Esta clase es un hilo. Como representa a una persona, tiene los siguientes atributos:

- private String id_persona → se compone de la letra P seguida del número correspondiente con el puesto en el que se ha creado esa persona.
- private int origen → planta de origen.
- private int destino → planta a la que desea ir esa persona.
- private String sentido → si la persona quiere subir, es decir, su destino es mayor que su origen, su sentido será S. Si quiere bajar, es decir, destino menor que origen, su sentido será B.
- private boolean esta_en_ascensor = false → indica si la persona está montada o no en el ascensor.
- private Hospital hospital → el hospital.

Cada atributo tiene sus funciones y métodos de get y set para poder utilizarlos y cambiarlos durante la ejecución del programa.

Al tratarse de ser un hilo, esta clase tiene un método run(), el cual hará que finalmente consiga subirse al ascensor.

NOTA → En ocasiones sale por pantalla un id de persona con "P" duplicadas, como por ejemplo PPP38. Esto lo hemos hecho a posta para que resalten las personas que salían de un ascensor estropeado.

7. Planta

Esta clase, como su propio nombre indica, representa una planta del hospital.

Atributos (además de sus correspondientes locks):

- private int id_planta → la planta que es.
- private boolean pulsado → para comprobar si el botón de esa planta está o no pulsado.
- private ArrayList <Persona> personas → cada una de las plantas tendrá un ArrayList que contenga a las personas esperando dentro de ellas.

Métodos:

- public int getId_planta()
- public void set_Id_planta()
- … demás GETTER y SETTER correspondientes, todos ellos con sus cerrojos implementados.

8. main

Esta clase, que es la principal, se encarga de inicializar tanto el Hospital como el Control (la parte de programación distribuida, RMI) e inicializa cada uno de los hilos que representan a las personas, siempre y cuando no haya sido pulsado el botón de evacuación del hospital.

9. ModuloEvacuacion

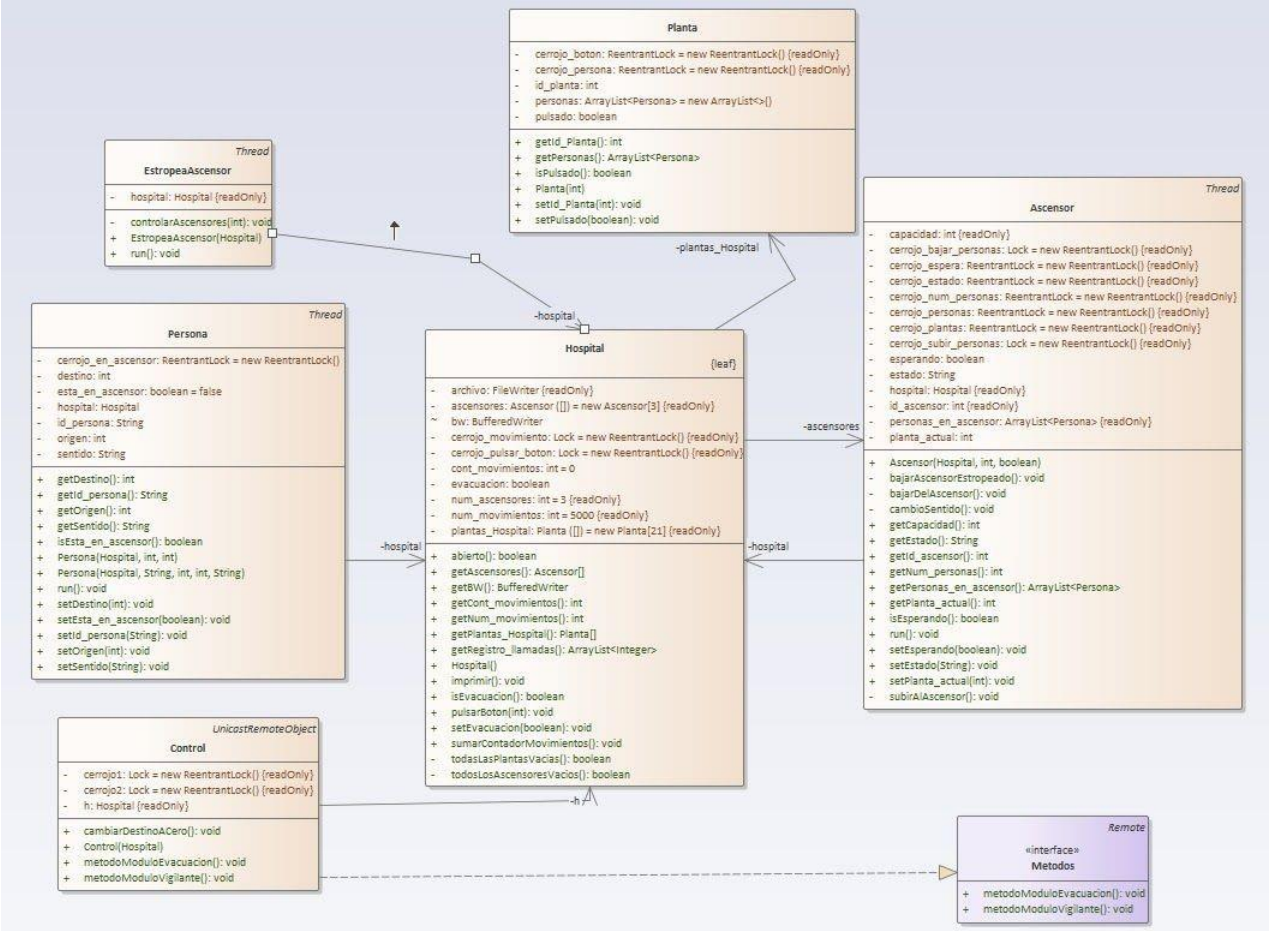
Esta clase corresponde a una interfaz gráfica donde encontramos un botón que al pulsarlo hace que se ejecute remotamente el método metodoModuloEvacuacion() de la clase Control, provocando la evacuación del hospital.

10. ModuloVigilante

Esta clase corresponde a una interfaz gráfica donde encontramos un botón que al pulsarlo hace que se ejecute remotamente el método metodoModuloVigilante() de la clase Control, mostrando por pantalla el estado puntual de todo el hospital.

4. DIAGRAMA DE CLASES

Se adjunta archivo .JPEG para comprobarlo con mejor detalle.



5. CÓDIGO FUENTE (anexo)

// Ascensor

```
package ascensor;

import java.io.IOException;
import static java.lang.Thread.sleep;
import java.util.ArrayList;
import java.util.Random;
import java.util.concurrent.locks.Lock;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.concurrent.locks.ReentrantLock;

/**
 *
 * @author Laura y Victor
 */
public class Ascensor extends Thread
{

    private final int id_ascensor;
    private String estado; //Parado, Bajando, Subiendo, Estropeado
    private final int capacidad;  // = 8
    private boolean esperando;
    private int planta_actual;
    private final ArrayList<Persona> personas_en_ascensor;
    private final Hospital hospital;

    // LOCKS
    private final ReentrantLock cerrojo_estado = new ReentrantLock();
    private final ReentrantLock cerrojo_num_personas = new ReentrantLock();
    private final ReentrantLock cerrojo_plantas = new ReentrantLock();
    private final ReentrantLock cerrojo_espera = new ReentrantLock();
    private final ReentrantLock cerrojo_personas = new ReentrantLock();

    private final Lock cerrojo_bajar_personas = new ReentrantLock();
    private final Lock cerrojo_subir_personas = new ReentrantLock();


    public Ascensor(Hospital h, int id, boolean esperando)
    {
        this.hospital = h;
        this.id_ascensor = id;
        this.estado = "P";
        this.capacidad = 8;
        this.esperando = esperando;
        this.planta_actual = (int) (21 * Math.random());
        this.personas_en_ascensor = new ArrayList<>();
    }
```

```
@Override
public void run()
{
    try {
        while (hospital.abierto())
        {
            if ("S".equals(getEstado()))
            {
                if (getPlanta_actual() < 21)
                {
                    setPlanta_actual(getPlanta_actual() + 1);
                }
                bajarDelAscensor();
                subirAlAscensor();
                try
                {
                    hospital.sumarContadorMovimientos();
                }
                catch (IOException ex)
                {
                    Logger.getLogger(Ascensor.class.getName()).log(Level.SEVERE, null, ex);
                }
                try
                {
                    sleep(500);
                }
                catch (InterruptedException e) {
                    System.out.println(e.getMessage());
                }
            }
            else if ("B".equals(getEstado()))
            {
                if (getPlanta_actual() >= 0)
                {
                   setPlanta_actual(getPlanta_actual() - 1);
                }
                bajarDelAscensor();
                subirAlAscensor();
                try
                {
                    hospital.sumarContadorMovimientos();
                }
                catch (IOException ex)
                {
                    Logger.getLogger(Ascensor.class.getName()).log(Level.SEVERE, null, ex);
                }
                try
                {
                    sleep(500);
                }
                catch (InterruptedException e) {
                    System.out.println(e.getMessage());
                }
            }
            else if ("P".equals(getEstado()))
            {
                if(!isEsperando())
```

```
                {
                    bajarDelAscensor();
                    subirAlAscensor();
                }
                try
                {
                    sleep(300);
                }
                catch (InterruptedException e)
                {
                    System.out.println(e.getMessage());
                }
            }
            else if ("E".equals(getEstado()))
            {
                bajarAscensorEstropeado();
                try
                {
                    sleep(new Random().nextInt(5001) + 10000);
                }
                catch (InterruptedException e)
                {
                    System.out.println(e.getMessage());
                }
                setEstado("P");
            }
            cambioSentido();
        }
    }
    catch (IOException ex)
    {
        Logger.getLogger(Ascensor.class.getName()).log(Level.SEVERE, null, ex);
    }
}

////////////////////////////////////////////////////////////////////////
//GETTERS Y SETTERS
////////////////////////////////////////////////////////////////////////

/**
 * Get the value of estado
 *
 * @return the value of estado
 */
public String getEstado()
{
    String estado_aux;
    cerrojo_estado.lock();
    try
    {
        estado_aux = estado;
    }
    finally
    {
        cerrojo_estado.unlock();
    }
    return estado_aux;
}
```

```
/**
 * Set the value of estado
 *
 * @param estado new value of estado
 */
public void setEstado(String estado)
{
   cerrojo_estado.lock();
    try
    {
        this.estado = estado;
    }
    finally
    {
        cerrojo_estado.unlock();
    }
}

/**
 * Get the value of esperando
 *
 * @return the value of esperando
 */
public boolean isEsperando()
{
    boolean espera;
    cerrojo_espera.lock();
    try
    {
        espera = esperando;
    }
    finally
    {
        cerrojo_espera.unlock();
    }
    return espera;
}

/**
 * Set the value of esperando
 *
 * @param esperando new value of esperando
 */
public void setEsperando(boolean esperando)
{
    cerrojo_espera.lock();
    try
    {
        this.esperando = esperando;
    }
    finally
    {
        cerrojo_espera.unlock();
    }
}

/**
```

```
     * Get the value of capacidad
     *
     * @return the value of capacidad
     */
    public int getCapacidad()
    {
        return capacidad;
    }

    /**
     * Get the value of planta_actual
     *
     * @return the value of planta_actual
     */
    public int getPlanta_actual()
    {
        int planta;
        cerrojo_plantas.lock();
        try
        {
            planta = planta_actual;
        }
        finally
        {
            cerrojo_plantas.unlock();
        }
        return planta;
    }

    /**
     * Set the value of planta_actual
     *
     * @param planta_actual new value of planta_actual
     */
    public void setPlanta_actual(int planta_actual)
    {
        cerrojo_plantas.lock();
        try
        {
            this.planta_actual = planta_actual;
        }
        finally
        {
            cerrojo_plantas.unlock();
        }
    }

    /**
     * Get the value of num_personas
     *
     * @return the value of num_personas
     */
    public int getNum_personas()
    {
        int num;
        cerrojo_num_personas.lock();
        try
        {
```

```
            num = personas_en_ascensor.size();
        }
        finally
        {
            cerrojo_num_personas.unlock();
        }
        return num;
    }

    /**
     * Get the value of id
     *
     * @return the value of id
     */
    public int getId_ascensor()
    {
        return id_ascensor;
    }

    /**
     * Get the value of personas_en_ascensor
     *
     * @return the value of personas_en_ascensor
     */
    public ArrayList<Persona> getPersonas_en_ascensor()
    {
        ArrayList<Persona> personas_aux = new ArrayList <>();
        cerrojo_personas.lock();
        try
        {
            personas_aux = personas_en_ascensor;
        }
        finally
        {
            cerrojo_personas.unlock();
        }
        return personas_aux;
    }


    ////////////////////////////////////////////////////////////////////////////
    // FUNCIONES
    ////////////////////////////////////////////////////////////////////////////

    /**
     * Se bajan las personas con ese destino
     * @param a
     */
    private void bajarDelAscensor()
    {
        cerrojo_bajar_personas.lock();
        try
        {
            int desplazamiento = 0;
            int veces = getPersonas_en_ascensor().size();
            for (int i = 0; i < veces; i++)
            {
                Persona p = getPersonas_en_ascensor().get(i - desplazamiento);
```

```
                if (p.getDestino() == getPlanta_actual())
                {
                    getPersonas_en_ascensor().remove(p);
                    desplazamiento++;
                }
            }
        }
        finally
        {
            cerrojo_bajar_personas.unlock();
        }
    }

    /**
     * Se bajan todos los ocupantes del ascensor, pues el ascensor se ha estropeado
     * añadir esas personas a la planta actual
     */
    private void bajarAscensorEstropeado()
    {
        cerrojo_bajar_personas.lock();
        try
        {
            int desplazamiento = 0;
            int veces = getPersonas_en_ascensor().size();
            for (int i = 0; i < veces; i++)
            {
                Persona p = getPersonas_en_ascensor().get(i - desplazamiento);
                        Persona p_aux = new Persona (hospital, ("P" + p.getId_persona()), getPlanta_actual(),
p.getDestino(), p.getSentido());

                hospital.getPlantas_Hospital()[getPlanta_actual()].getPersonas().add(p_aux);
                p_aux.start();
                desplazamiento++;
            }
        }
        finally
        {
            cerrojo_bajar_personas.unlock();
        }
    }

    /** Se suben al ascensor aquellas personas en la planta actual en la que nos encontramos
     *  Siempre y cuando la capacidad lo permita
     * @param a
     */
    private synchronized void subirAlAscensor()
    {
        for (int i = 0; i < hospital.getPlantas_Hospital()[getPlanta_actual()].getPersonas().size(); i++)
        {
            Persona p = hospital.getPlantas_Hospital()[getPlanta_actual()].getPersonas().get(i);

            if (getNum_personas() < getCapacidad()) // si puedo meter gente
            {
                if (p.getSentido().equals("S")) // Si esa persona sube
                {
                    if (getEstado().equals("S")) // si ascensor esta subiendo
                    {
```

```
                    cerrojo_subir_personas.lock();
                    try
                    {
                        p.setEsta_en_ascensor(true);
                        getPersonas_en_ascensor().add(p); //meto a la persona al ascensor
                            hospital.getPlantas_Hospital()[getPlanta_actual()].getPersonas().remove(p);// saco a la
persona de la planta
                    }
                    finally
                    {
                        cerrojo_subir_personas.unlock();
                    }
                }
                else if (getEstado().equals("P") || getNum_personas() == 0) //si esta parado o no tiene personas
                {
                    setEstado("S");
                    cerrojo_subir_personas.lock();
                    try
                    {
                        p.setEsta_en_ascensor(true);
                        getPersonas_en_ascensor().add(p); //meto a la persona al ascensor
                            hospital.getPlantas_Hospital()[getPlanta_actual()].getPersonas().remove(p);// saco a la
persona de la planta
                    }
                    finally
                    {
                        cerrojo_subir_personas.unlock();
                    }
                }
            }
            else // Baja
            {
               if (getEstado().equals("B")) // si esta bajando
                {
                    cerrojo_subir_personas.lock();
                    try
                    {
                        p.setEsta_en_ascensor(true);
                        getPersonas_en_ascensor().add(p); //meto a la persona al ascensor
                            hospital.getPlantas_Hospital()[getPlanta_actual()].getPersonas().remove(p);// saco a la
persona de la planta
                    }
                    finally
                    {
                        cerrojo_subir_personas.unlock();
                    }
                }
                else if (getEstado().equals("P") || getNum_personas() == 0) //si esta parado o no tiene personas
                {
                    setEstado("B");
                    cerrojo_subir_personas.lock();
                    try
                    {
                        p.setEsta_en_ascensor(true);
                        getPersonas_en_ascensor().add(p); //meto a la persona al ascensor
                            hospital.getPlantas_Hospital()[getPlanta_actual()].getPersonas().remove(p);// saco a la
persona de la planta
                    }
```

```
                finally
                {
                    cerrojo_subir_personas.unlock();
                }
            }
        }
    }
    hospital.getPlantas_Hospital()[getPlanta_actual()].setPulsado(false);
}

/**
 * cambio de sentido del ascensor en los siguientes casos:
 * "S" y llega a la planta 20
 * "B" y llega a la planta 0
 * @param a
 */
private void cambioSentido ()
{
    if (("S".equals(getEstado())))
    {
        if (getNum_personas() == 0)
        {
            setEstado("P");
        }
        else
        {
            if (getPlanta_actual() != 20)
            {
                setEstado("S");
            }
            else
            {
                setEstado("B");
            }
        }
    }
    else if (("B".equals(getEstado())))
    {
        if (getNum_personas() == 0)
        {
            setEstado("P");
        }
        else
        {
            if (getPlanta_actual() != 0)
            {
                setEstado("B");
            }
            else
            {
                setEstado("S");
            }
        }
    }
    else if (("P".equals(getEstado())))
    {
        if (isEsperando())
```

```
            {
                setEstado("P");
            }
            else
            {
                if (hospital.getRegistro_llamadas().isEmpty())
                {
                    setEstado("P");
                }
                else if (hospital.getRegistro_llamadas().get(0) >= getPlanta_actual())
                {
                    setEstado("S");
                }
                else
                {
                    setEstado("B");
                }
            }
        }
    }
}
```

// Control

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package ascensor;

import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

/**
 *
 * @author Laura y Victor
 */
public class Control extends UnicastRemoteObject implements Metodos
{
    private final Hospital h;
    private final Lock cerrojo1 = new ReentrantLock();
    private final Lock cerrojo2 = new ReentrantLock();

    public Control(Hospital h) throws RemoteException
    {
        this.h = h;
    }

    ////////////////////////////////////////////////////////////////////////
    //modulo vigilante
    ////////////////////////////////////////////////////////////////////////

    /**
     * Este módulo permitirá consultar de forma remota el estado de los dos ascensores del hospital,
```

```
     * mostrando su estado de forma automática con una periodicidad de 1 segundo.
     * La información relativa a la situación de cada ascensor mostrada será: piso en el que se
     * encuentra el ascensor, cuántas personas se encuentran en su interior, así como sus destinos y
     * las personas que se encuentran esperando en cada planta.
     * @throws java.rmi.RemoteException
     */
    @Override
    public void metodoModuloVigilante() throws RemoteException
    {
        cerrojo1.lock();
        try
        {

System.out.println("_____
_____MODULO
VIGILANTE_____
_____");
            for (int i = 0; i < h.getAscensores().length; i++)
            {
                        System.out.println("ASCENSOR: " + h.getAscensores()[i].getId_ascensor() + " - Estado: " +
h.getAscensores()[i].getEstado() + " - Nº personas: " + h.getAscensores()[i].getNum_personas() + " - **
Planta Actual ** - " + h.getAscensores()[i].getPlanta_actual());

                System.out.println("// PERSONAS DENTRO ");
                for (int j = 0; j < h.getAscensores()[i].getPersonas_en_ascensor().size(); j++)
                {
                    System.out.println(h.getAscensores()[i].getPersonas_en_ascensor().get(j).getId_persona() + " -
Origen: " + h.getAscensores()[i].getPersonas_en_ascensor().get(j).getOrigen() + " - Destino: " +
h.getAscensores()[i].getPersonas_en_ascensor().get(j).getDestino());
                }
                System.out.println("_____");
            }

            for (int i = h.getPlantas_Hospital().length - 1; i >= 0; i--) //recorremos al revés
            {
                System.out.println("-|-|-|- " + i + " -|-|-|-");
                for (int j = 0; j < h.getPlantas_Hospital()[i].getPersonas().size(); j++)
                {
                    Persona p = h.getPlantas_Hospital()[i].getPersonas().get(j);
                            System.out.println(p.getId_persona() + " - Origen: " + p.getOrigen() + " - Destino: " +
p.getDestino());
                }
                System.out.println("");
            }

System.out.println("_____
_____
_____\n");
        }
        finally
        {
            cerrojo1.unlock();
        }
    }

    ////////////////////////////////////////////////////////////////////////
    //modulo evacuacion
```

```
    ////////////////////////////////////////////////////////////////////////////

    /**
     * dejarán de crearse nuevas personas y, automáticamente, la planta de destino
     * de todas las personas que estén esperando a un ascensor será la planta baja,
     * para abandonar el hospital.
     * @throws RemoteException
     */
    @Override
    public void metodoModuloEvacuacion() throws RemoteException
    {
        cerrojo2.lock();
        try
        {
            h.setEvacuacion(true);
            cambiarDestinoACero();
        }
        finally
        {
            cerrojo2.unlock();
        }
    }

    /**
     * cambiamos todos los destino a cero
     */
    private void cambiarDestinoACero()
    {
        //cambiamos a los que estan dentro de los ascensores
        for (int i = 0; i < h.getAscensores().length; i++)
        {
            for (int j = 0; j < h.getAscensores()[i].getPersonas_en_ascensor().size(); j++)
            {
                Persona p = h.getAscensores()[i].getPersonas_en_ascensor().get(j);
                p.setDestino(0);
            }
        }

        //cambiamos a los que están en las plantas
        for (int i = h.getPlantas_Hospital().length - 1; i >= 0; i--) //recorremos al revés
        {
            for (int j = 0; j < h.getPlantas_Hospital()[i].getPersonas().size(); j++)
            {
                Persona p = h.getPlantas_Hospital()[i].getPersonas().get(j);
                p.setDestino(0);
            }
        }
    }
}
```

// EstropeaAscensor

```
package ascensor;

import static java.lang.Thread.sleep;

/**
 *
 * @author Laura y Víctor
 */
public class EstropeaAscensor extends Thread
{

    private final Hospital hospital;

    public EstropeaAscensor(Hospital h)
    {
        hospital = h;
    }

    /**
     * se controlan los ascensores
     * @param n
     */
    private void controlarAscensores(int n)
    {
        for (int i = 0; i < hospital.getAscensores().length; i++)
        {
            if (hospital.getAscensores()[i].getId_ascensor() == n)
            {
                hospital.getAscensores()[i].setEsperando(true);
                hospital.getAscensores()[i].setEstado("E");
            }
            else
            {
                hospital.getAscensores()[i].setEsperando(false);
            }
        }
    }

    @Override
    public void run()
    {
        while (hospital.getCont_movimientos() < hospital.getNum_movimientos())
        {
            try
            {
                sleep((int) (10000 + 10000 * Math.random()));
            }
            catch (InterruptedException e)
            {
                System.out.println(e.getMessage());
            }
            int ascensor = (int) (Math.random() * 3) + 1;
            while (hospital.getAscensores()[ascensor - 1].isEsperando())
            {
                ascensor = (int) (Math.random() * 3) + 1;
            }
```

```
            controlarAscensores(ascensor);
        }
    }
}
```

// Hospital

```
package ascensor;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

/**
 *
 * @author Laura y Victor
 */
public final class Hospital
{

    private final int num_movimientos = 5000; // total de movimientos a hacer por el programa
    private int cont_movimientos = 0; // contador de movimientos actuales
    private final int num_ascensores = 3;

    private boolean evacuacion;

    private final Planta plantas_Hospital[] = new Planta[21];
    private final Ascensor ascensores[] = new Ascensor[3];

    //archivo
    private final FileWriter archivo;
    BufferedWriter bw;

    // LOCKS
    private final Lock cerrojo_movimiento = new ReentrantLock();
    private final Lock cerrojo_pulsar_boton = new ReentrantLock();


    public Hospital() throws IOException
    {
        for (int i = 0; i < 21; i++)
        {
            plantas_Hospital[i] = new Planta(i);
        }

        for (int i = 0; i < 3; i++)
        {
            if (i != 2)
            {
                ascensores[i] = new Ascensor(this, i+1, false);
            }
            else
            {
                ascensores[i] = new Ascensor(this, i+1, true);
            }
```

```
            ascensores[i].start();
        }

        EstropeaAscensor ea = new EstropeaAscensor(this);
        ea.start();

        //fichero
        String ruta = "archivo.txt";
        archivo = new FileWriter(ruta);
        bw = new BufferedWriter(archivo);

        evacuacion = false;

        imprimir();
    }

    /**
     * se suman los movimientos
     * @throws IOException
     */
    public void sumarContadorMovimientos() throws IOException
    {
        cerrojo_movimiento.lock();
        try
        {
            cont_movimientos++;
            imprimir();
        }
        finally
        {
            cerrojo_movimiento.unlock();
        }
    }

    /**
     * Comprueba si el hospital esta abierto o no
     * lo hace mirando si los movimientos no han superado al num max de mov
     * Además, se cerrará si se evacua el hospital
     * @return
     * @throws java.io.IOException
     */
    public boolean abierto() throws IOException
    {
        if (!(cont_movimientos < num_movimientos)) //esto es que va a estar cerrado
        {
            bw.close();
        }
        if (evacuacion && todasLasPlantasVacias() && todosLosAscensoresVacios())
        {
            return false;
        }
        else
        {
            return cont_movimientos < num_movimientos;
        }
    }

    /**
```

```
 * vemos si todas las plantas estan vacias o no
 * @return
 */
private synchronized boolean todasLasPlantasVacias()
{
    int comprobante = 0;

    for (int i = 0; i < plantas_Hospital.length; i++)
    {
        if (!plantas_Hospital[i].getPersonas().isEmpty())
        {
            comprobante++;
        }
    }

    return comprobante == 0;
}

/**
 * comprobamos si estan todos los ascensores vacios o no
 * @return
 */
private synchronized boolean todosLosAscensoresVacios()
{
    int comprobante = 0;

    for (int i = 0; i < ascensores.length; i++)
    {
        if (!ascensores[i].getPersonas_en_ascensor().isEmpty())
        {
            comprobante ++;
        }
    }

    return comprobante == 0;
}

/**
 * La persona cuando llega a la planta pulsa el boton del ascensor
   Nos encontramos una zona critica --> locks
 * @param planta_origen
 */
public synchronized void pulsarBoton(int planta_origen)
{
    cerrojo_pulsar_boton.lock();
    try
    {
        plantas_Hospital[planta_origen].setPulsado(true);

    }
    finally
    {
        cerrojo_pulsar_boton.unlock();
    }
}

//////////////////////////////////////////////////////
//IMPRIMIR
```

```
    ////////////////////////////////////////////////////

    public synchronized void imprimir() throws IOException
    {
            System.out.println("----------\nMovimiento nº " +
getCont_movimientos() + "\n");
                    bw.write("----------\nMovimiento nº " +
getCont_movimientos() + "\n" + "\n");

        System.out.println("Piso    Asc.1    Asc.2    Asc.3    Botón pulsado:    Destinos Asc.1    Destinos Asc.2
Destinos Asc.3    Personas planta");
         bw.write("\nPiso    Asc.1    Asc.2    Asc.3    Botón pulsado:    Destinos Asc.1    Destinos Asc.2
Destinos Asc.3    Personas planta\n");

      for (int i = plantas_Hospital.length - 1; i >= 0; i--)
      {
         System.out.print(" ");
         bw.write(" ");

         if (plantas_Hospital[i].getId_Planta() < 10)
         {
            System.out.print(" ");
            bw.write(" ");
         }

         System.out.print(plantas_Hospital[i].getId_Planta() + "      ");
         bw.write(plantas_Hospital[i].getId_Planta() + "      ");

         for (int j = 0; j < 3; j++)
         {
            if (ascensores[j].getPlanta_actual() == plantas_Hospital[i].getId_Planta()) //misma planta
            {
               if ("E".equals(ascensores[j].getEstado()))
               {
                  System.out.print(" E ");
                  bw.write(" E ");
               }
               else
               {
                  System.out.print(ascensores[j].getEstado() + "#" + ascensores[j].getNum_personas());
                  bw.write(ascensores[j].getEstado() + "#" + ascensores[j].getNum_personas());
               }
            }
            else  //distinta planta
            {
               System.out.print(" | ");
               bw.write(" | ");
            }
            System.out.print("      ");
            bw.write("      ");
         }

         if (plantas_Hospital[i].isPulsado())
         {
            System.out.print("  --Sí");
            bw.write("  --Sí");
         }
         else
```

```
        {
            System.out.print("     No");
            bw.write("     No");
        }

        System.out.print("          ");
        bw.write("          ");

        for (int j = 0; j < num_ascensores; j++)
        {
            ArrayList<Persona> personas_ascensor = new ArrayList<>();
            for (int k = 0; k < ascensores[j].getNum_personas(); k++)
            {
                if (ascensores[j].getPersonas_en_ascensor().get(k).getDestino() == i)
                {
                    personas_ascensor.add(ascensores[j].getPersonas_en_ascensor().get(k));
                }
            }

            int n_espacios = 7;

            if (personas_ascensor.size() > 1)
            {
                                                    n_espacios = Math.max((14 - (personas_ascensor.size() *
String.valueOf(ascensores[j].getPersonas_en_ascensor().get(0).getId_persona()).length()                    +
(personas_ascensor.size() - 1) * 2)) / 2, 0);
            }
            else if (personas_ascensor.size() == 1)
            {
                                                              n_espacios = (14 - personas_ascensor.size() *
String.valueOf(ascensores[j].getPersonas_en_ascensor().get(0).getId_persona()).length()) / 2;
            }

            for (int k = 0; k < n_espacios; k++)
            {
                System.out.print(" ");
                bw.write(" ");
            }

            for (int k = 0; k < personas_ascensor.size(); k++)
            {
                System.out.print(personas_ascensor.get(k).getId_persona());
                bw.write(personas_ascensor.get(k).getId_persona());
                if (personas_ascensor.size() - 1 != k) {
                    System.out.print(", ");
                    bw.write(", ");
                }
            }

            for (int k = 0; k < n_espacios; k++)
            {
                System.out.print(" ");
                bw.write(" ");
            }
            System.out.print("    ");
            bw.write("    ");
        }
```

```
            for (int j = 0; j < plantas_Hospital[i].getPersonas().size(); j++)
            {
                System.out.print(plantas_Hospital[i].getPersonas().get(j).getId_persona());
                bw.write(plantas_Hospital[i].getPersonas().get(j).getId_persona());
                if (plantas_Hospital[i].getPersonas().size() - 1 != j)
                {
                    System.out.print(", ");
                    bw.write(", ");
                }
            }

            System.out.println("");
            bw.write("\n");
        }

        System.out.println("");
        bw.write("\n");
        for (int i = 0; i < ascensores.length; i++)
        {
                        System.out.println("ASCENSOR: " + ascensores[i].getId_ascensor() + " - Estado: " +
ascensores[i].getEstado() + " - Nº personas: " + ascensores[i].getNum_personas() + " - ** Planta Actual ** - "
+ ascensores[i].getPlanta_actual());
            bw.write("ASCENSOR: " + ascensores[i].getId_ascensor() + " - Estado: " + ascensores[i].getEstado()
+ " - Nº personas: " + ascensores[i].getNum_personas() + " - ** Planta Actual ** - " +
ascensores[i].getPlanta_actual() + "\n");

            System.out.println("// PERSONAS DENTRO ");
            bw.write("// PERSONAS DENTRO \n");

            for (int j = 0; j < ascensores[i].getPersonas_en_ascensor().size(); j++)
            {
                System.out.println(ascensores[i].getPersonas_en_ascensor().get(j).getId_persona() + " - Origen: "
+       ascensores[i].getPersonas_en_ascensor().get(j).getOrigen()       +       "       -       Destino:       "       +
ascensores[i].getPersonas_en_ascensor().get(j).getDestino());
                        bw.write(ascensores[i].getPersonas_en_ascensor().get(j).getId_persona() + " - Origen: " +
ascensores[i].getPersonas_en_ascensor().get(j).getOrigen()       +       "       -       Destino:       "       +
ascensores[i].getPersonas_en_ascensor().get(j).getDestino() + "\n");
            }
            System.out.println("----------");
            bw.write("----------\n");
        }
        bw.write("\n");
        bw.write("\n");
    }

    ////////////////////////////////////////////////////////////////
    // GETTERS Y SETTERS
    ////////////////////////////////////////////////////////////////
     public BufferedWriter getBW()
    {
        return bw;
    }

    public int getNum_movimientos()
    {
        return num_movimientos;
    }
```

```
    public int getCont_movimientos()
    {
        int cont;
        cerrojo_movimiento.lock();
        try
        {
            cont = cont_movimientos;
        }
        finally
        {
            cerrojo_movimiento.unlock();
        }
        return cont;
    }

    public synchronized Planta[] getPlantas_Hospital()
    {
        return plantas_Hospital;
    }

    public Ascensor[] getAscensores()
    {
        return ascensores;
    }

    public void setEvacuacion(boolean e)
    {
        evacuacion = e;
    }

    public boolean isEvacuacion()
    {
        return evacuacion;
    }

    public ArrayList<Integer> getRegistro_llamadas()
    {
        ArrayList<Integer> llamadas = new ArrayList<>();

        for (int i = 0; i < 21; i++)
        {
            if (plantas_Hospital[i].isPulsado())
            {
                llamadas.add(i);
            }
        }
        return llamadas;
    }
}
```

// Metodos

```
package ascensor;

import java.rmi.Remote;
import java.rmi.RemoteException;

/**
 * Aquí se encuentran los métodos que necesitábamos de Control
 * Las interfaces "remota" de los modulos instancian una variable de esta clase
 *
 * @author Laura y Victor
 */

public interface Metodos extends Remote
{
    //Modulo vigilante
    void metodoModuloVigilante() throws RemoteException;

    //Modulo evacuacion
    void metodoModuloEvacuacion() throws RemoteException;
}
```

// Persona

```
package ascensor;

import java.util.concurrent.locks.ReentrantLock;

/**
 *
 * @author Laura y Victor
 */
public class Persona extends Thread
{

    private String id_persona;
    private int origen;
    private int destino;
    private String sentido;
    private boolean esta_en_ascensor = false;
    private Hospital hospital;

    private ReentrantLock cerrojo_en_ascensor = new ReentrantLock();

    public Persona(Hospital h, int num_id_persona, int origen_planta)
    {
        this.hospital = h;
        this.id_persona = "P" + num_id_persona;
        this.origen = origen_planta;
        destino = (int) (20 * Math.random());
        while (this.origen == this.destino)  //nos aseguramos que origen != destino
        {
            this.destino = (int) (20 * Math.random());
        }
        //sentido
        if (this.origen < this.destino)
```

```
        {
            this.sentido = "S";
        }
        else
        {
            this.sentido = "B";
        }
    }

    public Persona(Hospital h, String num_id_persona, int origen_planta, int destino_planta, String sentido)
    {
        this.hospital = h;
        this.id_persona = num_id_persona;
        this.origen = origen_planta;
        this.destino = destino_planta;
        this.sentido = sentido;
    }

    @Override
    public void run()
    {
        while (!isEsta_en_ascensor())
        {
            try
            {
                if (!hospital.getPlantas_Hospital()[origen].isPulsado())
                {
                    hospital.pulsarBoton(origen);
                }
                sleep(100);
            }
            catch (InterruptedException e)
            {
                System.out.println(e.getMessage());
            }
        }
    }

    ////////////////////////////////////////////////////////////////////////////
    //GETTERS Y SETTERS
    ////////////////////////////////////////////////////////////////////////////

    /**
     * Get the value of origen
     *
     * @return the value of origen
     */
    public synchronized int getOrigen()
    {
        return origen;
    }

    /**
     * Set the value of origen
     *
     * @param origen new value of origen
     */
    public void setOrigen(int origen)
```

```
{
    this.origen = origen;
}

/**
 * Get the value of esta_en_ascensor
 *
 * @return the value of origen
 */
public boolean isEsta_en_ascensor()
{
    boolean ascensor;
    cerrojo_en_ascensor.lock();
    try
    {
        ascensor = esta_en_ascensor;
    }
    finally
    {
        cerrojo_en_ascensor.unlock();
    }
    return ascensor;
}

/**
 * Set the value of esta_en_ascensor
 *
 * @param esta_en_ascensor
 */
public void setEsta_en_ascensor(boolean esta_en_ascensor)
{
    cerrojo_en_ascensor.lock();
    try
    {
        this.esta_en_ascensor = esta_en_ascensor;
    }
    finally
    {
        cerrojo_en_ascensor.unlock();
    }
}


/**
 * Get the value of destino
 *
 * @return the value of destino
 */
public synchronized int getDestino()
{
    return destino;
}

/**
 * Set the value of destino
 *
 * @param destino new value of destino
 */
```

```
    public void setDestino(int destino)
    {
        this.destino = destino;
    }

    /**
     * Get the value of id_persona
     *
     * @return the value of id_persona
     */
    public synchronized String getId_persona()
    {
        return id_persona;
    }

    /**
     * Set the value of id_persona
     *
     * @param id_persona new value of id_persona
     */
    public void setId_persona(String id_persona)
    {
        this.id_persona = id_persona;
    }

    /**
     * Get the value of sentido
     *
     * @return the value of sentido
     */
    public synchronized String getSentido()
    {
        return sentido;
    }

    /**
     * Set the value of sentido
     *
     * @param sentido new value of sentido
     */
    public void setSentido(String sentido)
    {
        this.sentido = sentido;
    }

}
```

// Planta

```
package ascensor;

import java.util.ArrayList;
import java.util.concurrent.locks.ReentrantLock;

/**
 *
 * @author Laura y Victor
 */
public class Planta
{
    private int id_planta;
    private boolean pulsado;
    private ArrayList<Persona> personas = new ArrayList<>();
    private final ReentrantLock cerrojo_boton = new ReentrantLock();
    private final ReentrantLock cerrojo_persona = new ReentrantLock();

    public Planta (int n_planta)
    {
        id_planta = n_planta;
        pulsado = false;
        personas = new ArrayList<Persona>();
    }

    ////////////////////////////////////////////////////////////////////////
    // GETTER Y SETTER
    ////////////////////////////////////////////////////////////////////////

    public int getId_Planta()
    {
        return id_planta;
    }

    public void setId_Planta(int numPlanta)
    {
        this.id_planta = numPlanta;
    }

    public boolean isPulsado()
    {
        boolean boton;
        cerrojo_boton.lock();
        try
        {
            boton = pulsado;
        }
        finally
        {
            cerrojo_boton.unlock();
        }
        return boton;
    }

    public void setPulsado(boolean estaPulsado)
    {
        cerrojo_boton.lock();
```

```
        try
        {
            this.pulsado = estaPulsado;
        }
        finally
        {
            cerrojo_boton.unlock();
        }
    }

    public ArrayList<Persona> getPersonas()
    {
        ArrayList<Persona> persona;
        cerrojo_persona.lock();
        try
        {
            persona = personas;
        }
        finally
        {
            cerrojo_persona.unlock();
        }
        return persona;
    }
}
```

// MAIN (clase principal)

```
package ascensor;

import java.io.IOException;
import static java.lang.Thread.sleep;
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import  java.rmi.registry.Registry;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author Laura y Victor
 */
public class main
{
    public static void main(String[] args) throws InterruptedException, IOException
    {

        Hospital h = new Hospital();
        Control c = new Control(h);

        try
        {
            Registry registry = LocateRegistry.createRegistry(1099); //Arranca rmiregistry local en el puerto 1099
            Naming.rebind("//localhost/Metodos", c);  //rebind sólo funciona sobre una url del equipo local
        }
        catch (RemoteException | MalformedURLException ex)
        {
```

```
            Logger.getLogger(main.class.getName()).log(Level.SEVERE, null, ex);
        }

        // GENERADOR DE PERSONAS
        int i = 1;
        while ((i <= 6000) && (h.isEvacuacion() == false))
        {
            int origen_persona = (int) (20 * Math.random());
            Persona p = new Persona(h, i, origen_persona);
            p.start();
            sleep((int)(500 + Math.random() * 1500));
            h.getPlantas_Hospital()[origen_persona].getPersonas().add(p);
            i++;
        }
    }
}
```

// ModuloEvacuacion (INTERFAZ)

```
package interfaces;

import ascensor.Metodos;
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import  java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author laura
 */
public class ModuloEvacuacion extends javax.swing.JFrame {

    /**
     * Creates new form ModuloEvacuacion
     */

    Metodos acceso;

    public ModuloEvacuacion() throws NotBoundException, MalformedURLException, RemoteException {
        initComponents();
        this.setLocationRelativeTo(null); //ajustarlo en el medio de la pantalla
        acceso = (Metodos) Naming.lookup("//localhost/Metodos");
    }

    /**
     * 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() {

        jButton2 = new javax.swing.JButton();
        jLabel1 = new javax.swing.JLabel();
```

```
        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        getContentPane().setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

        jButton2.setFont(new java.awt.Font("Dialog", 1, 36)); // NOI18N
        jButton2.setText("MÓDULO EVACUACIÓN");
        jButton2.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                jButton2MouseClicked(evt);
            }
        });
        jButton2.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton2ActionPerformed(evt);
            }
        });
        getContentPane().add(jButton2, new org.netbeans.lib.awtextra.AbsoluteConstraints(120, 170, 480, 80));

          jLabel1.setIcon(new javax.swing.ImageIcon(getClass().getResource("/imagenes/ascensores.jpg"))); //
NOI18N
        jLabel1.setText("jLabel1");
        getContentPane().add(jLabel1, new org.netbeans.lib.awtextra.AbsoluteConstraints(0, 0, 720, -1));

        pack();
    }// </editor-fold>

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void jButton2MouseClicked(java.awt.event.MouseEvent evt) {
        try {
            // TODO add your handling code here:
            acceso.metodoModuloEvacuacion();
        } catch (RemoteException ex) {
            Logger.getLogger(ModuloEvacuacion.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    /**
     * @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 | InstantiationException | IllegalAccessException |
javax.swing.UnsupportedLookAndFeelException ex) {
```

```
java.util.logging.Logger.getLogger(ModuloEvacuacion.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        }
        //</editor-fold>

        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            @Override
            public void run() {
                try {
                    new ModuloEvacuacion().setVisible(true);
                } catch (NotBoundException | MalformedURLException | RemoteException ex) {
                    Logger.getLogger(ModuloEvacuacion.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JButton jButton2;
    private javax.swing.JLabel jLabel1;
    // End of variables declaration
}
```

// ModuloVigilante (INTERFAZ)

```
package interfaces;

import ascensor.Metodos;
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import  java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author laura y vitor
 */
public class ModuloVigilante extends javax.swing.JFrame {

    /**
     * Creates new form ModuloVigilante
     */

    Metodos acceso;

    public ModuloVigilante() throws NotBoundException, MalformedURLException, RemoteException {
        initComponents();
        this.setLocationRelativeTo(null); //ajustarlo en el medio de la pantalla
        acceso = (Metodos) Naming.lookup("//localhost/Metodos");
    }

    /**
     * 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() {

        jFrame1 = new javax.swing.JFrame();
        jButton1 = new javax.swing.JButton();
        jLabel1 = new javax.swing.JLabel();

        javax.swing.GroupLayout jFrame1Layout = new javax.swing.GroupLayout(jFrame1.getContentPane());
        jFrame1.getContentPane().setLayout(jFrame1Layout);
        jFrame1Layout.setHorizontalGroup(
            jFrame1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 400, Short.MAX_VALUE)
        );
        jFrame1Layout.setVerticalGroup(
            jFrame1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 300, Short.MAX_VALUE)
        );

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setResizable(false);
        getContentPane().setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

        jButton1.setFont(new java.awt.Font("Dialog", 1, 24)); // NOI18N
        jButton1.setText("MÓDULO VIGILANTE");
        jButton1.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                jButton1MouseClicked(evt);
            }
        });
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });
        getContentPane().add(jButton1, new org.netbeans.lib.awtextra.AbsoluteConstraints(210, 150, 310, 80));

          jLabel1.setIcon(new javax.swing.ImageIcon(getClass().getResource("/imagenes/ascensores.jpg"))); //
NOI18N
        jLabel1.setText("jLabel1");
        getContentPane().add(jLabel1, new org.netbeans.lib.awtextra.AbsoluteConstraints(0, 0, 720, -1));

        pack();
    }// </editor-fold>

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void jButton1MouseClicked(java.awt.event.MouseEvent evt) {
        try {
            // TODO add your handling code here:
            acceso.metodoModuloVigilante();
        } catch (RemoteException ex) {
            Logger.getLogger(ModuloVigilante.class.getName()).log(Level.SEVERE, null, ex);
```

```
        }
    }

    /**
     * @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(ModuloVigilante.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(ModuloVigilante.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(ModuloVigilante.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(ModuloVigilante.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        }
        //</editor-fold>
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            @Override
            public void run() {
                try {
                    new ModuloVigilante().setVisible(true);
                } catch (NotBoundException | MalformedURLException | RemoteException ex) {
                    Logger.getLogger(ModuloVigilante.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JFrame jFrame1;
    private javax.swing.JLabel jLabel1;
    // End of variables declaration
```

```
}
```