



Grado en Ingeniería de Computadores

Curso 2019/2020 || Convocatoria extraordinaria

53755294Q - Bustos Miranda, Eduardo

09105135X - Munuera Pérez, César

| | |
|--|-----------|
| Análisis de alto nivel | 3 |
| Parte I | 3 |
| Parte II | 3 |
| Diseño general del sistema y de las herramientas de sincronización utilizadas | 4 |
| Parte I | 4 |
| Parte II | 4 |
| Clases principales que intervienen con su descripción (atributos y métodos) | 5 |
| Parte I | 5 |
| Parte II | 11 |
| Diagrama de clases | 13 |
| Parte I | 13 |
| Parte II | 14 |
| Código fuente | 15 |
| Parte I | 15 |
| Parte II | 63 |

1. Análisis de alto nivel

1.1. Parte I

Después de haber leído y analizado los requerimientos de esta práctica podemos realizar este análisis.

El programa debe tener 3 ascensores que van a estar implementados por hilos. En ellos, las personas de las diferentes plantas del hospital, se van a poder mover a otras plantas. Estas personas también serán modeladas por hilos.

No todos los ascensores van a funcionar a la vez. Hay dos ascensores que estarán funcionando continuamente (siempre y cuando haya gente que los necesite). El tercero será uno de reserva que se utilizará en caso de que uno de los otros dos ascensores se estropee.

Las personas saldrán de forma aleatoria en cada planta, llamarán al ascensor, el ascensor que pueda irá a por ella, se montaran y finalmente saldrán en la planta que hayan marcado (también aleatoriamente).

1.2. Parte II

Para el módulo del vigilante se requerirá un sistema de conteo, para poder saber en cada momento qué gente está en cada ascensor, sus destinos, dónde está cada ascensor, ... Realmente será una consulta de cliente-servidor a través de RMI.

Para el módulo de evacuación se implementará un método que será llamado desde el cliente, y lo ejecutará el servidor. Se detendrá el servicio de los elevadores, y comenzará otra función. Esta consiste en ir bajando a todas las personas a la planta baja, sin importar que quieran ir a otra planta. Realizará los viajes necesarios para no dejar a nadie en el hospital.

2. Diseño general del sistema y de las herramientas de sincronización utilizadas

2.1. Parte I

Como hemos explicado anteriormente, tanto los ascensores como las personas, van a ser modeladas como hilos.

Las personas van a utilizar la clase *Floor* para acceder al botón del ascensor. Este botón pertenece a una clase (clase botón) enlazada con otra (una clase que simulará una centralita).

La centralita será la que registre a todas las personas, y diga a los ascensores a dónde van a ir, ya que registra a través del botón, a dónde hay que ir a buscar a personas o a dejar a personas. También tendrá la capacidad de exportar datos al módulo vigilante. Por tanto, los ascensores son mandados en todo momento a los sitios a donde deben ir.

También existe una clase para guardar configuraciones, además de una clase hospital con las plantas, un generador de personas y una clase, modelada como un hilo, que simula los fallos de los ascensores cada cierto tiempo aleatorio.

2.2. Parte II

El módulo del vigilante se realizará con JFrames. Una ventana donde aparecerá toda la información relativa a cada ascensor. Es el cliente, que solicitará toda la información al servidor. Este, la obtendrá de la centralita, a su vez de los ascensores. También aparecerán las personas que hay en cada planta.

En la misma ventana de JFrame se encontrará el botón de evacuación. El cliente solicitará un método al objeto del servidor. Esto se encontrará en la centralita, que mandará a los elevadores detenerse inmediatamente para bajar a todas las personas a la planta baja.

3. Clases principales que intervienen con su descripción (atributos y métodos)

3.1. Parte I

- **Elevator** Hereda de thread

ATRIBUTOS

- private static final Logger logger;
- private String identificador;
- private int previusFloor;
- private int currentFloor;
- private Map<Integer, Boolean> requestedFloors;

MÉTODOS

- public void initRequestedFloors ();
Setea el map "requestedFloors" en false, ya que en el primer instante, nadie ha llamado al ascensor.
- public void turnOn ();
- public void turnOff ();
- public void turnEnd ();
- public void broke ();
- synchronized void forceOutPeople ();
- synchronized void forceOutPeople (boolean evacuateSystem);
- private void move ();
- void moveToNextFloor ();
- public void stopInFloor ();
Este método hace que el ascensor pare cuando llega a una planta que ha sido solicitada.
- public boolean enter (Person person);
- public synchronized void out (Person person);
- private void repair ();
Se llama cuando se rompe un ascensor, para repararlo.
- private void waitInFloor ();

Cuando el ascensor llega a una planta, este método hace que primero salgan las personas, antes de que suban los que estaban esperándolo. Respetando el dicho de “dejar salir antes de entrar”.

- public void run ();

El ascensor va a parar primero en la planta 0 por si hay gente esperando. Luego, mientras que el programa no se haya acabado y el ascensor siga estando operativo, va a continuar yendo a su siguiente planta asignada. Si hay una evacuación, los echa del ascensor.

- ***ElevatorBackUp*** Hereda de Elevator

Esta clase representa al tercer ascensor, que se activará cuando uno de los dos ascensores principales se haya roto y existan personas que lo necesiten.

- ***ElevatorBreaker*** Hereda de thread

ATRIBUTOS

- private static final Logger logger;
- private List <Elevator> elevator;
- private ElevatorBackUp elevatorBackUp;

MÉTODOS

- private void brokeRandomElevator ();
- private void sleepRandomTime ();
- public void run ();

Este método espera un tiempo aleatorio (con un rango prefijado), y rompe un ascensor.

- ***Hospital***

ATRIBUTOS

- private static final Logger logger;

MÉTODOS

- private void initFloors ();
Inicializa las plantas del hospital, con sus respectivos botones (del ascensor).
- public HospitalFloor getFloor (int currentFloor);
Se introduce el número de una planta, y este método te retorna el objeto de esa planta, te retorna esa planta.

- ***HospitalFloor***

ATRIBUTOS

- private static final Logger logger;
- private JarvisRemoteControl jarvisRemoteControl;

MÉTODOS

- public void callElevator ();
- public ArrayList <Elevator> getElevator ();
Tanto este método, como en anterior, utilizan jarvisRemoteControl para llamar a sus métodos.

- ***JarvisSystem***

ATRIBUTOS

- private static final Logger logger;
- private ArrayList <JarvisRemoteControl> remotes;
- private final AtomicInteger movesCounter;
- private ArrayList <Elevator> elevators;

MÉTODOS

- private void initElevators ();
Crea y almacena los dos tipos de ascensores (los normales y el de backup), y el elevatorBreaker.
- private synchronized void printStatus ();
Imprime en la terminal el desarrollo de los ascensores.
- public void addMovement ();
Método que se utiliza para controlar que el número de movimientos no se pase de los especificados en el enunciado.
- public void turnSystemOff ();
- public void startEvacuation ();
- public void callElevator ();
- public ArrayList <Elevartor> getElevatorsInFloor (int floor);
Dada una planta pasada como entero, retorna un AL con los ascensores que se encuentran en dicha planta.
- public HashMap <Integer, Integer> getPeopleInAllFloors ();
Retorna un hashmap con todas las personas de todas las plantas del hospital.

• *JarvisRemoteControl*

ATRIBUTOS

- private static final Logger logger;
- private JarvisSystem jarvisSystem;
- private final Lock lock
- private final Condition elevatorInFloorCondition;

MÉTODOS

- private void waitForElvator ();
Usa el lock y la condición para que la gente duerma una vez que llaman al ascensor, hasta que este llegue a su planta.
- public synchronized void notifyElevatorIsArriving ();
Desbloquea el waitForElvator (), se realiza mediante la condición.

- public void callElevator (int floor);

Si el botón no está pulsado (nadie lo ha pulsado antes), permite que lo pulsen. Una vez que ha sido pulsado, se bloquea. Esta condición de bloqueo la activa otro método llamado notifyElevatorLeaving().

- **Logging**

ATRIBUTOS

- private static final Logger logger;
- private static final String LOG_NAME
- private static final String LOF_FILE

MÉTODOS

- public static void initLogger ();

Sirve para almacenar en un txt, una vez que se ha acabado la ejecución, toda la información de dicha ejecución.

- **PeopleGenerator** Hereda de thread

ATRIBUTOS

- private static final Logger logger;

MÉTODOS

- private String idGenerator ();
- private Person generate ();
- public void run ();

Usa métodos como idGenerator y varios más para cálculos secundarios, como tiempos aleatorios. Luego, con generate (), genera a una persona (usando los resultados de estos métodos anteriores).

- **Person** Hereda de thread

ATRIBUTOS

- private static final Logger logger;
- private static final String PREFIX = "P";
- private String identifier;
- private int sourceFloor
- private int floor;
- private int targetFloor;

MÉTODOS

- public void chooseDirection ();
Elige la dirección del ascensor en función de la posición de la persona, y su destino.
- public void evacuate ();
- private void waitFlor(Elevator elevator);
La persona va a esperar, mediante una espera activa, a llegar a su planta de destino. Para ello, va a ir actualizando constantemente por la planta en la que se encuentra. Cuando llega, se acaba la espera activa.
- private Elevator chooseElevator (ArrayList <Elevator> elevator);
Comprueba si los ascensores están disponibles, y si tienen la misma dirección que la que la persona desea.
- public void run ();
Primero se llama al ascensor. Una vez que llega, se comprueba si está evacuando. Si no es así, se deja salir a la gente. En caso de que esté lleno, se reiniciaría el ciclo. Si no es así, y la persona cabe, entra y espera a llegar a su destino.

- **ElevatorInfoDTO**

ATRIBUTOS

- private int peopleCounter;
- private int floor;
- private String identification;
- private List <Integer> peopleDestination;

MÉTODOS

- Getters de los métodos anteriores.

3.2. Parte II

- **ClientController** RMI → Cliente

MÉTODOS

- `public List<ElevatorInfoDTO> getElevatorsInfo ();`
Retorna una lista de los elevadores, para poder exportar la información al JFrame.
- `public void evacuateSystem ();`
Solicita al servidor que realice una evacuación.
- `public Map<Integer, Integer> getPeopleInFloor ();`
Retorna un mapa con todas las personas que hay en las diferentes plantas del hospital.

- **ClientInfoRefresher** Hereda de thread

Es una clase que tiene la función de retardar un segundo las ráfagas que muestra el JFrame. Es decir, le da una periodicidad de un segundo.

- **MainClient**

Es una clase que, básicamente, levanta el front (el JFrame) donde el cliente proyecta los datos que solicita al objeto del servidor.

- **MainServer** RMI → Servidor

MÉTODOS

- `private static void startServer (ServerController controller);`
Método que levanta el servidor, pasándole el respectivo objeto para conectarlo.

- ***ServerController***

ATRIBUTOS

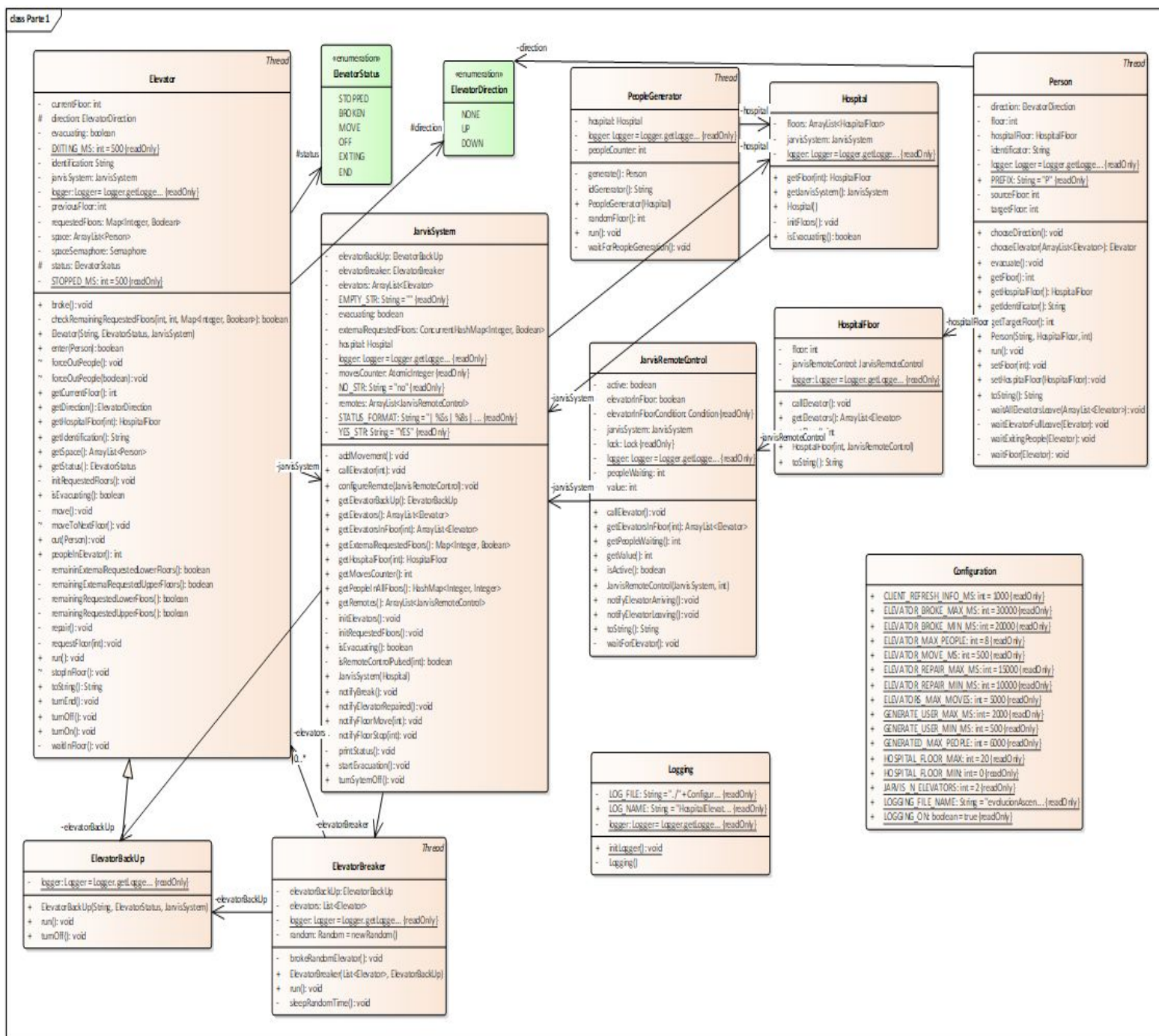
- private JarvisSystem jarvisSystem;

MÉTODOS

- public List<ElevatorInfoDTO> getElevatorsInfo ();
Retorna una lista de tipo ElevatorInfoDTO, donde se recoge la información relativa a los ascensores, para ponerlos en el front.
- public void evacuateSystem ();
Llama a un método del jarvisSystem para comenzar la evacuación.

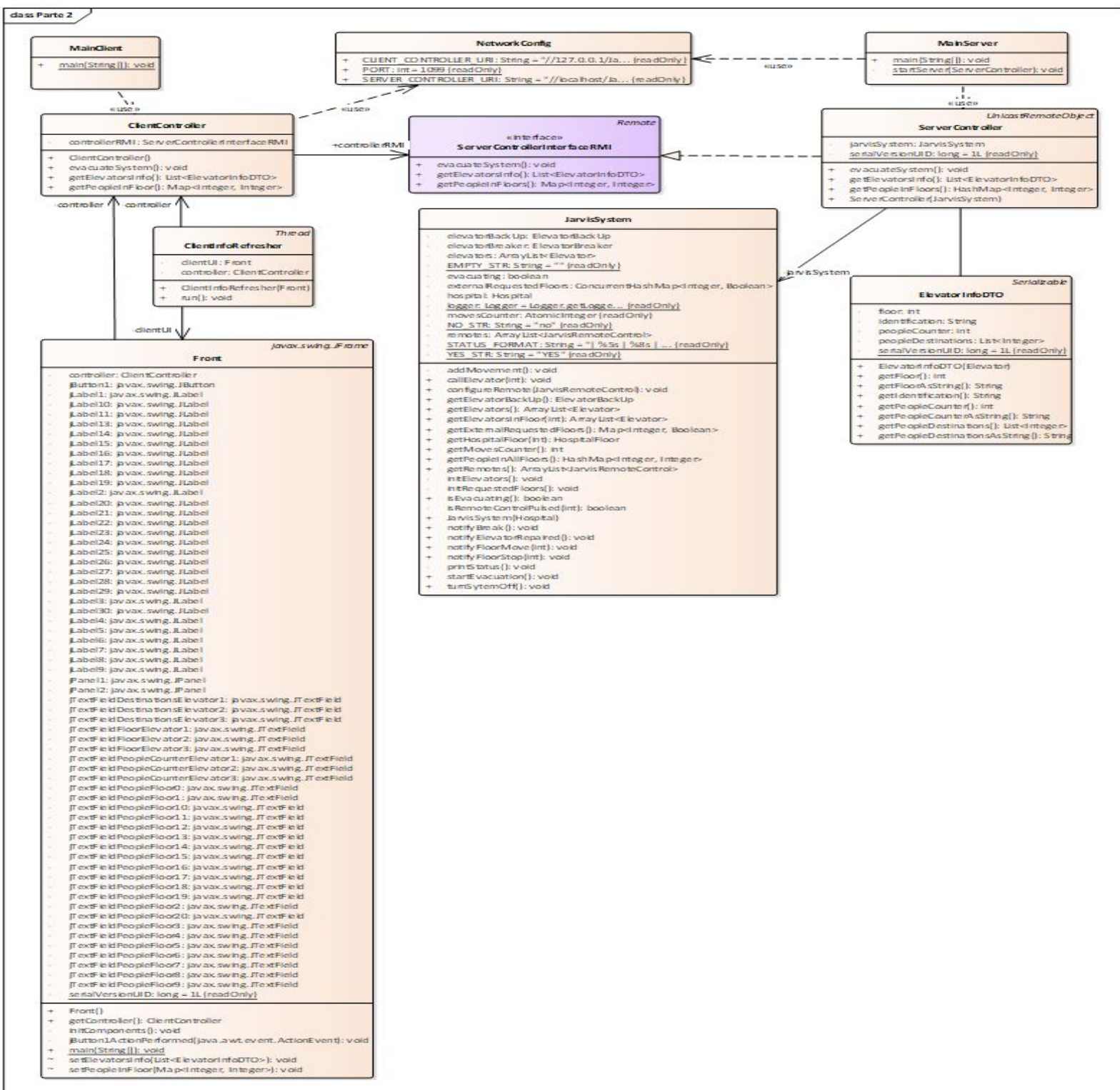
4. Diagrama de clases

4.1. Parte I



Se ha insertado un PDF llamado "Parte 1" donde se puede apreciar mejor.

4.2. Parte II



Se ha insertado un PDF llamado "Parte 2" donde se puede apreciar mejor.

5. Código fuente

5.1. Parte I

```
package hospital;
```

```
public class Configuration {
```

```
    public static final int JARVIS_N_ELEVATORS = 2;
    public static final int HOSPITAL_FLOOR_MIN = 0;
    public static final int HOSPITAL_FLOOR_MAX = 20;
    public static final int GENERATED_MAX_PEOPLE = 6000;
    public static final int GENERATE_USER_MIN_MS = 500;
    public static final int GENERATE_USER_MAX_MS = 2000;
    public static final int ELEVATORS_MAX_MOVES = 5000;
    public static final int ELEVATOR_MAX_PEOPLE = 8;
    public static final int ELEVATOR_BROKE_MIN_MS = 20000;
    public static final int ELEVATOR_BROKE_MAX_MS = 30000;
    public static final int ELEVATOR_REPAIR_MIN_MS = 10000;
    public static final int ELEVATOR_REPAIR_MAX_MS = 15000;
    public static final int ELEVATOR_MOVE_MS = 500;
    public static final boolean LOGGING_ON = true;
    public static final String LOGGING_FILE_NAME = "evolucionAscensor.log";
    public static final int CLIENT_REFRESH_INFO_MS = 1000;
```

```
}
```

```
package hospital;
```



```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.Semaphore;
import java.util.logging.Logger;

public class Elevator extends Thread {

    private static final Logger logger = Logger.getLogger(Logging.LOG_NAME);
    private static final int EXITING_MS = 500;
    private static final int STOPPED_MS = 500;

    private String identification;
    private int currentFloor;
    private int previousFloor;
    private Semaphore spaceSemaphore;
    private JarvisSystem jarvisSystem;
    private ArrayList<Person> space;
    private Map<Integer, Boolean> requestedFloors;
    protected ElevatorStatus status;
    protected ElevatorDirection direction;
    private boolean evacuating;

    private void initRequestedFloors() {
        this.requestedFloors = new HashMap<>();
        for (int i = Configuration.HOSPITAL_FLOOR_MIN; i <=
Configuration.HOSPITAL_FLOOR_MAX; i++) {
            this.requestedFloors.put(i, false);
        }
        if (Configuration.LOGGING_ON) {
```



```
        logger.info("requested floors initialized");
    }
}

public Elevator(String identification, ElevatorStatus status, JarvisSystem jarvisSystem) {
    this.identification = identification;
    this.jarvisSystem = jarvisSystem;
    this.currentFloor = Configuration.HOSPITAL_FLOOR_MIN;
    this.previousFloor = Configuration.HOSPITAL_FLOOR_MIN;
    this.spaceSemaphore = new Semaphore(Configuration.ELEVATOR_MAX_PEOPLE, true);
    this.space = new ArrayList<Person>(Configuration.ELEVATOR_MAX_PEOPLE);
    this.status = status;
    this.direction = ElevatorDirection.NONE;
    this.evacuating = false;
    initRequestedFloors();
}

@Override
public String toString() {
    return "Elevator(" + this.identification + ", floor = " + this.currentFloor + ", people = " +
this.peopleInElevator()
        + ", " + this.status.name() + ", " + this.direction.name() + ")";
}

public int peopleInElevator() {
    return space.size();
}

public void turnOn() {
    if (Configuration.LOGGING_ON) {
```

```
        logger.info(this.toString() + " turning on");
    }
    this.status = ElevatorStatus.STOPPED;
    this.direction = ElevatorDirection.NONE;

}

public void turnOff() {
    if (Configuration.LOGGING_ON) {
        logger.info(this.toString() + " turning off");
    }

    this.status = ElevatorStatus.OFF;
    this.direction = ElevatorDirection.NONE;
    forceOutPeople();

}

public void turnEnd() {
    if (Configuration.LOGGING_ON) {
        logger.info(this.toString() + " turning end");
    }
    this.status = ElevatorStatus.END;
    this.direction = ElevatorDirection.NONE;
    forceOutPeople();

}

public void broke() {
```

```
    if (Configuration.LOGGING_ON) {
        logger.info(this.toString() + " breaking up");
    }
    this.status = ElevatorStatus.BROKEN;
    this.direction = ElevatorDirection.NONE;
    this.jarvisSystem.notifyBreak();

}

synchronized void forceOutPeople() {
    forceOutPeople(false);
}

synchronized void forceOutPeople(boolean evacuateSystem) {
    if (evacuateSystem) {
        this.evacuating = true;
    }
    if (peopleInElevator() > 0) {
        if (Configuration.LOGGING_ON) {
            logger.info(this.toString() + " start evacuating people");
        }
        for (Person person : this.space) {
            person.setFloor(this.currentFloor);
            if (this.evacuating) {
                person.evacuate();
            }
            person.interrupt();
        }
    }
}
```

```
        if (Configuration.LOGGING_ON) {
            logger.info(this.toString() + " evacuated people");
        }
    }
}

private void move() throws InterruptedException {
    this.status = ElevatorStatus.MOVE;
    if (Configuration.LOGGING_ON) {
        logger.info(this.toString() + " continue moving ");
    }

    sleep((long) Configuration.ELEVATOR_MOVE_MS);
    if (this.direction == ElevatorDirection.UP) {
        this.previousFloor = this.currentFloor;
        this.currentFloor++;
    } else if (this.direction == ElevatorDirection.DOWN) {
        this.previousFloor = this.currentFloor;
        this.currentFloor--;
    }

    if (Configuration.LOGGING_ON) {
        logger.info(this.toString() + " move to floor " + this.currentFloor);
    }
    this.jarvisSystem.notifyFloorMove(this.currentFloor);
}

private boolean checkRemainingRequestedFloors(int from, int to, Map<Integer, Boolean>
map) {
    boolean remaining = false;
```

```
int i = from;
int step = 1;
int stop = to + 1;
if (from > to) {
    step = -1;
    stop = to - 1;
}
while (!remaining && i != stop) {
    remaining = map.get(i);
    i += step;
}
return remaining;
}
```

```
private boolean remainingRequestedUpperFloors() {
    boolean remaining = false;
    if (this.currentFloor < Configuration.HOSPITAL_FLOOR_MAX) {
        remaining = checkRemainingRequestedFloors(this.currentFloor + 1,
Configuration.HOSPITAL_FLOOR_MAX, this.requestedFloors);
    }
    return remaining;
}
```

```
private boolean remainingExternalRequestedUpperFloors() {
    boolean remaining = false;
    if (this.currentFloor < Configuration.HOSPITAL_FLOOR_MAX) {
        remaining = checkRemainingRequestedFloors(this.currentFloor + 1,
Configuration.HOSPITAL_FLOOR_MAX, this.jarvisSystem.getExternalRequestedFloors());
    }
}
```

```
        return remaining;
    }

    private boolean remainingRequestedLowerFloors() {
        boolean remaining = false;
        if (this.currentFloor > Configuration.HOSPITAL_FLOOR_MIN) {
            remaining = checkRemainingRequestedFloors(this.currentFloor - 1,
Configuration.HOSPITAL_FLOOR_MIN, this.requestedFloors);
        }
        return remaining;
    }

    private boolean remaininExternalRequestedLowerFloors() {
        boolean remaining = false;
        if (this.currentFloor > Configuration.HOSPITAL_FLOOR_MIN) {
            remaining = checkRemainingRequestedFloors(this.currentFloor - 1,
Configuration.HOSPITAL_FLOOR_MIN, this.jarvisSystem.getExternalRequestedFloors());
        }
        return remaining;
    }

    void moveToNextFloor() throws InterruptedException {
        if (this.status != ElevatorStatus.OFF) {
            boolean internalUpperRequestedFloors = remainingRequestedUpperFloors();
            boolean externalUpperRequestedFloors =
remainingExternalRequestedUpperFloors();
            boolean internalLowerRequestedFloors = remainingRequestedLowerFloors();
            boolean externalLowerRequestedFloors = remaininExternalRequestedLowerFloors();
            boolean move = false;
```

```
if (this.currentFloor == Configuration.HOSPITAL_FLOOR_MIN) {
    this.direction = ElevatorDirection.UP;

} else if (this.currentFloor == Configuration.HOSPITAL_FLOOR_MAX) {
    this.direction = ElevatorDirection.DOWN;

}

if (this.direction == ElevatorDirection.UP) {
    if (internalUpperRequestedFloors || externalUpperRequestedFloors) {
        move = true;
    } else {
        this.direction = ElevatorDirection.NONE;
    }
}

if (this.direction == ElevatorDirection.DOWN) {
    if (internalLowerRequestedFloors || externalLowerRequestedFloors) {
        move = true;
    } else {
        this.direction = ElevatorDirection.NONE;
    }
}

if (this.direction == ElevatorDirection.NONE) {
    if (internalUpperRequestedFloors || externalUpperRequestedFloors) {
        this.direction = ElevatorDirection.UP;
        move = true;
    } else {
        if (internalLowerRequestedFloors || externalLowerRequestedFloors) {
            this.direction = ElevatorDirection.DOWN;
```

```
        move = true;
    }
}

if (move) {
    move();
}

if (this.currentFloor == Configuration.HOSPITAL_FLOOR_MAX ||
    this.currentFloor == Configuration.HOSPITAL_FLOOR_MIN) {
    this.direction = ElevatorDirection.NONE;
}
}
}

private void waitInFloor() throws InterruptedException {
    if (Configuration.LOGGING_ON) {
        logger.info(toString() + " arrived to floor " + this.currentFloor);
    }
    this.status = ElevatorStatus.EXITING;
    sleep((long) EXITING_MS);
    this.status = ElevatorStatus.STOPPED;
    sleep((long) STOPPED_MS);
}

void stopInFloor() throws InterruptedException {
    if (this.status != ElevatorStatus.OFF) {
        boolean floorInternalRequired = this.requestedFloors.get(this.currentFloor);
```



```
        boolean floorExternalRequired =
this.jarvisSystem.getExternalRequestedFloors().get(this.currentFloor);
        if (floorInternalRequired || floorExternalRequired) {
            if (Configuration.LOGGING_ON) {
                logger.info(this.toString() + " stopping in floor " + this.currentFloor);
            }
            this.jarvisSystem.notifyFloorStop(this.currentFloor);
            waitInFloor();
            this.requestedFloors.put(this.currentFloor, false);
            this.jarvisSystem.getExternalRequestedFloors().put(this.currentFloor, false);
        }
    }
}
```

```
private void repair() {
    if (Configuration.LOGGING_ON) {
        logger.info(this.toString() + " starts repairing");
    }
    double randomTime = (Math.random() * (Configuration.ELEVATOR_REPAIR_MAX_MS -
Configuration.ELEVATOR_REPAIR_MIN_MS + 1)
        + Configuration.ELEVATOR_REPAIR_MIN_MS);

    try {
        Thread.sleep((long) randomTime);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    initRequestedFloors();
    this.status = ElevatorStatus.STOPPED;
    this.jarvisSystem.notifyElevatorRepaired();
}
```

```
        if (Configuration.LOGGING_ON) {
            logger.info(this.toString() + " ends repairing");
        }
    }

    public boolean enter(Person person) {
        if (Configuration.LOGGING_ON) {
            logger.info(this.toString() + " trying enter - " + person.toString());
        }
        boolean inside = false;
        inside = this.spaceSemaphore.tryAcquire();
        if (inside) {
            this.space.add(person);
            if (Configuration.LOGGING_ON) {
                logger.info(this.toString() + " inside - " + person.toString());
            }
            requestFloor(person.getTargetFloor());
        }

        return inside;
    }

    private void requestFloor(int floor) {
        this.requestedFloors.put(floor, true);
    }

    public synchronized void out(Person person) {
        person.setHospitalFloor(this.jarvisSystem.getHospitalFloor(this.currentFloor));
        person.setFloor(this.currentFloor);
    }
}
```

```
this.spaceSemaphore.release();
this.space.remove(person);
if (Configuration.LOGGING_ON) {
    logger.info(this.toString() + " went out - " + person.toString());
}
}
```

```
@Override
public void run() {
    if (this.status != ElevatorStatus.OFF) {
        try {
            stopInFloor();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    while (this.status != ElevatorStatus.END) {
        while (this.status != ElevatorStatus.OFF) {
            try {
                moveToNextFloor();
                stopInFloor();
            } catch (InterruptedException e) {
                broke();
                forceOutPeople();
                repair();
            }
        }
    }
    try {
```

```
        sleep(10);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
forceOutPeople();
}

public String getIdentification() {
    return identification;
}

public int getCurrentFloor() {
    return currentFloor;
}

public synchronized ArrayList<Person> getSpace() {
    return space;
}

public ElevatorStatus getStatus() {
    return status;
}

public ElevatorDirection getDirection() {
    return direction;
}

public HospitalFloor getHospitalFloor(int nFloor) {
```

```
        return this.jarvisSystem.getHospitalFloor(nFloor);
    }

    public boolean isEvacuating() {
        return evacuating;
    }
}

package hospital;

import java.util.logging.Logger;

public class ElevatorBackUp extends Elevator {

    private static final Logger logger = Logger.getLogger(Logging.LOG_NAME);

    public ElevatorBackUp(String id, ElevatorStatus status, JarvisSystem jarvisSystem) {
        super(id, status, jarvisSystem);
    }

    public void turnOff() {
        this.status = ElevatorStatus.OFF;
        this.direction = ElevatorDirection.NONE;
        interrupt();
    }

    @Override
```

```
public void run() {
    if (this.status != ElevatorStatus.OFF) {
        try {
            stopInFloor();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    while (this.status != ElevatorStatus.END) {
        while (this.status != ElevatorStatus.OFF) {
            try {
                moveToNextFloor();
                stopInFloor();
                if (Configuration.LOGGING_ON) {
                    logger.info(toString() + ": moved");
                }
            } catch (InterruptedException e) {
                if (Configuration.LOGGING_ON) {
                    logger.info(toString() + ": stoping");
                }
                forceOutPeople();
            }
        }
    }
    try {
        sleep(10);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

```
    }
    forceOutPeople();
}

}

package hospital;

import java.util.List;
import java.util.Random;
import java.util.logging.Logger;

public class ElevatorBreaker extends Thread {

    private static final Logger logger = Logger.getLogger(Logging.LOG_NAME);

    private List<Elevator> elevators;
    private ElevatorBackUp elevatorBackUp;
    private Random random = new Random();

    public ElevatorBreaker(List<Elevator> elevators, ElevatorBackUp elevatorBackUp) {
        this.elevators = elevators;
        this.elevatorBackUp = elevatorBackUp;
    }

    private void sleepRandomTime() {
        double randomTime = (Math.random() * (Configuration.ELEVATOR_BROKE_MAX_MS -
Configuration.ELEVATOR_BROKE_MIN_MS + 1)
        + Configuration.ELEVATOR_BROKE_MIN_MS);
    }
}
```

```
try {
    Thread.sleep((long) randomTime);
} catch (InterruptedException e) {
    e.printStackTrace();
}
}

private void brokeRandomElevator() {
    boolean choosen = false;
    int randomNum;
    Elevator elevator;

    while (!choosen) {
        randomNum = this.random.nextInt(this.elevators.size());
        elevator = this.elevators.get(randomNum);
        if (elevator.getStatus() == ElevatorStatus.STOPPED) {
            elevator.interrupt();
            choosen = true;
        }
    }
}

@Override
public void run() {
    while (true) {
        sleepRandomTime();
        brokeRandomElevator();
    }
}
```




```
}
```

```
package hospital;
```

```
public enum ElevatorDirection {
```

```
    NONE,
```

```
    UP,
```

```
    DOWN
```

```
}
```

```
package hospital;
```

```
public enum ElevatorStatus {
```

```
    STOPPED,
```

```
    BROKEN,
```

```
    MOVE,
```

```
    OFF,
```

```
    EXITING,
```

```
    END
```

```
}
```

```
package hospital;
```

```
import java.util.ArrayList;
```

```
import java.util.logging.Logger;

public class Hospital {

    private static final Logger logger = Logger.getLogger(Logging.LOG_NAME);

    private ArrayList<HospitalFloor> floors;
    private JarvisSystem jarvisSystem;

    private void initFloors() {
        if (Configuration.LOGGING_ON) {
            logger.info("initializing floors");
        }
        this.floors = new ArrayList<>();
        JarvisRemoteControl remote;
        HospitalFloor floor;

        for (int nFloor = Configuration.HOSPITAL_FLOOR_MIN; nFloor <=
Configuration.HOSPITAL_FLOOR_MAX; nFloor++) {
            remote = new JarvisRemoteControl(this.jarvisSystem, nFloor);
            floor = new HospitalFloor(nFloor, remote);
            floors.add(floor);
        }
    }

    public Hospital() {
        this.jarvisSystem = new JarvisSystem(this);
        initFloors();
        if (Configuration.LOGGING_ON) {
            logger.info("initialized with " + this.floors.size() + " floors");
        }
    }
}
```

```
    }  
}  
  
public HospitalFloor getFloor(int currentFloor) {  
    HospitalFloor selectedFloor = null;  
    for (HospitalFloor floor : this.floors) {  
        if (floor.getFloor() == currentFloor) {  
            selectedFloor = floor;  
            break;  
        }  
    }  
  
    }  
    return selectedFloor;  
}  
  
public JarvisSystem getJarvisSystem() {  
    return jarvisSystem;  
}  
  
public boolean isEvacuating() {  
    return this.jarvisSystem.isEvacuating();  
}  
}  
  
package hospital;  
  
import java.util.ArrayList;  
import java.util.logging.Logger;
```

```
public class HospitalFloor {

    private static final Logger logger = Logger.getLogger(Logging.LOG_NAME);

    private int floor;
    private JarvisRemoteControl jarvisRemoteControl;

    public HospitalFloor(int floor, JarvisRemoteControl jarvisRemoteControl) {
        this.floor = floor;
        this.jarvisRemoteControl = jarvisRemoteControl;
        if (Configuration.LOGGING_ON) {
            logger.info(this.toString() + " initialized");
        }
    }

    @Override
    public String toString() {
        return "HospitalFloor(" + this.floor + ")";
    }

    public void callElevator() {
        this.jarvisRemoteControl.callElevator();
    }

    public ArrayList<Elevator> getElevators() {
        return this.jarvisRemoteControl.getElevatorsInFloor(this.floor);
    }

    public int getFloor() {
```

```
        return floor;
    }

}

package hospital;

import java.util.ArrayList;
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
import java.util.logging.Logger;

public class JarvisRemoteControl {

    private static final Logger logger = Logger.getLogger(Logging.LOG_NAME);

    private JarvisSystem jarvisSystem;
    private int value;
    private boolean active; // condition
    private boolean elevatorInFloor;
    private final Lock lock;
    private final Condition elevatorInFloorCondition;
    private int peopleWaiting;

    public JarvisRemoteControl(JarvisSystem jarvisSystem, int value) {
        this.jarvisSystem = jarvisSystem;
        this.active = false;
        this.elevatorInFloor = false;
    }
}
```

```
this.lock = new ReentrantLock();
this.elevatorInFloorCondition = lock.newCondition();
this.value = value;
this.jarvisSystem.configureRemote(this);
this.peopleWaiting = 0;
if (Configuration.LOGGING_ON) {
    logger.info(toString() + " initialized and configured");
}
}
```

```
@Override
public String toString() {
    return "JarvisRemote(" + this.value + ")";
}
```

```
private void waitForElevator() {
    try {
        lock.lock();
        this.peopleWaiting++;
        while (!elevatorInFloor) {
            try {
                elevatorInFloorCondition.await();
            } catch (Exception e) {
            }
        }
    }
    // end waiting - person continues with execution
    this.peopleWaiting = 0;
} finally {
    lock.unlock();
}
```

```
    }  
}  
  
public void callElevator() {  
    if (!this.active) {  
        this.jarvisSystem.callElevator(this.value);  
        this.active = true;  
    }  
  
    waitForElevator();  
}  
  
public synchronized void notifyElevatorArriving() {  
    if (Configuration.LOGGING_ON) {  
        logger.info(this.toString() + " elevator arrives - wake up people waiting");  
    }  
    try {  
        lock.lock();  
        this.elevatorInFloor = true;  
        elevatorInFloorCondition.signal();  
    } finally {  
        lock.unlock();  
    }  
}  
  
public void notifyElevatorLeaving() {  
    this.elevatorInFloor = false;  
    this.active = false;  
    this.peopleWaiting = 0;
```

```
}

public ArrayList<Elevator> getElevatorsInFloor(int floor) {
    return this.jarvisSystem.getElevatorsInFloor(floor);
}

public int getValue() {
    return value;
}

public boolean isActive() {
    return active;
}

public int getPeopleWaiting() {
    return peopleWaiting;
}

}

package hospital;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.atomic.AtomicInteger;
import java.util.logging.Logger;
```



```
public class JarvisSystem {

    private static final Logger logger = Logger.getLogger(Logging.LOG_NAME);

    private Hospital hospital;

    private static final String STATUS_FORMAT = "| %5s | %8s | %8s | %8s | %8s | %60s | %60s | %60s | \n";

    private static final String EMPTY_STR = "";
    private static final String YES_STR = "YES";
    private static final String NO_STR = "no";

    private ArrayList<Elevator> elevators;
    private ElevatorBackUp elevatorBackUp;
    private ConcurrentHashMap<Integer, Boolean> externalRequestedFloors;
    private ElevatorBreaker elevatorBreaker;
    private ArrayList<JarvisRemoteControl> remotes;
    private final AtomicInteger movesCounter;
    private boolean evacuating;

    private void initElevators() {
        this.elevators = new ArrayList<>(Configuration.JARVIS_N_ELEVATORS);

        Elevator elevator;
        for (int i = 0; i < Configuration.JARVIS_N_ELEVATORS; i++) {
            elevator = new Elevator("elevator_" + i, ElevatorStatus.MOVE, this);
            this.elevators.add(elevator);
            elevator.start();
        }
        this.elevatorBackUp = new ElevatorBackUp("elevator_bkp", ElevatorStatus.OFF, this);
    }
}
```

```
this.elevatorBackUp.start();
this.elevatorBreaker = new ElevatorBreaker(this.elevators, this.elevatorBackUp);
this.elevatorBreaker.start();
if (Configuration.LOGGING_ON) {
    logger.info("elevators initialized");
}
}

private void initRequestedFloors() {
    this.externalRequestedFloors = new ConcurrentHashMap<>();
    for (int i = Configuration.HOSPITAL_FLOOR_MIN; i <=
Configuration.HOSPITAL_FLOOR_MAX; i++) {
        getExternalRequestedFloors().put(i, false);
    }
}

public JarvisSystem(Hospital hospital) {
    if (Configuration.LOGGING_ON) {
        logger.info("initializing jarvis");
    }
    this.hospital = hospital;
    initRequestedFloors();
    initElevators();
    this.remotes = new ArrayList<>(Configuration.HOSPITAL_FLOOR_MAX + 1);
    this.movesCounter = new AtomicInteger(0);
    this.evacuating = false;
    if (Configuration.LOGGING_ON) {
        logger.info("jarvis initialized!");
    }
}
```

```
private synchronized void printStatus() {

    String statusString = "";
    String floor;
    String elevator0;
    String elevator1;
    String elevator2;
    String buttonPulsed;
    String destinationElevator0;
    String destinationElevator1;
    String destinationElevator2;

    statusString += String.format(STATUS_FORMAT, "Floor", "Evt.0", "Evt.1", "Evt.2",
    "B.Pulsed", "Dest.Evt.0", "Dest.Evt.1", "Dest.Evt.2") + "\n";

    int nFloor = Configuration.HOSPITAL_FLOOR_MAX;
    while (nFloor != Configuration.HOSPITAL_FLOOR_MIN - 1) {
        floor = String.valueOf(nFloor);
        elevator0 = EMPTY_STR;
        elevator1 = EMPTY_STR;
        elevator2 = EMPTY_STR;
        buttonPulsed = NO_STR;
        destinationElevator0 = EMPTY_STR;
        destinationElevator1 = EMPTY_STR;
        destinationElevator2 = EMPTY_STR;

        if (isRemoteControlPulsed(nFloor)) {
            buttonPulsed = YES_STR;
        }
    }
```

```

for (Elevator elevator : this.elevators) {
    if (elevator.getCurrentFloor() == nFloor) {
        ArrayList<String> peopleInElevator = new ArrayList<>();
        for (Person person : elevator.getSpace()) {
            peopleInElevator.add(person.toString());
        }
        String elevatorDirection = elevator.getDirection().name();
        if (elevator.getStatus() == ElevatorStatus.BROKEN) {
            elevatorDirection = "KO";
        }
        String elevatorString = elevatorDirection + "#" + elevator.peopleInElevator();
        if (elevator.getIdentification().equals("elevator_0")) {
            elevator0 = elevatorString;
            destinationElevator0 = peopleInElevator.toString();
        } else if (elevator.getIdentification().equals("elevator_1")) {
            elevator1 = elevatorString;
            destinationElevator1 = peopleInElevator.toString();
        }
    }
}

if (this.elevatorBackUp.getCurrentFloor() == nFloor) {
    String elevatorBackUpDirection = this.elevatorBackUp.getDirection().name();
    String elevatorBackUpString = elevatorBackUpDirection + "#" +
this.elevatorBackUp.peopleInElevator();
    elevator2 = elevatorBackUpString;
    ArrayList<String> peopleInElevatorBackUp = new ArrayList<>();
    for (Person person : elevatorBackUp.getSpace()) {
        peopleInElevatorBackUp.add(person.toString());
    }
}

```

```

        destinationElevator2 = peopleInElevatorBackUp.toString();
    }

    statusString += String.format(STATUS_FORMAT, floor, elevator0, elevator1, elevator2,
        buttonPulsed,
            destinationElevator0, destinationElevator1, destinationElevator2);

    nFloor--;
}

statusString += "\n\n----- " + "MOVEMENT " +
this.getMovesCounter() + " ----- \n\n";

if (Configuration.LOGGING_ON) {
    logger.info(statusString);
} else {
    System.out.println(statusString);
}

}

private boolean isRemoteControlPulsed(int n) {
    boolean pulsed = false;
    for (JarvisRemoteControl remote : this.remotes) {
        if (remote.getValue() == n) {
            pulsed = remote.isActive();
            break;
        }
    }
    return pulsed;
}

```



```
}
```

```
public int getMovesCounter() {  
    return movesCounter.get();  
}
```

```
private void addMovement() {  
    while (true) {  
        int existingValue = getMovesCounter();  
        int newValue = existingValue + 1;  
        if (movesCounter.compareAndSet(existingValue, newValue)) {  
            if (getMovesCounter() == Configuration.ELEVATORS_MAX_MOVES) {  
                turnSytemOff();  
            }  
  
            return;  
        }  
    }  
}
```

```
public void turnSytemOff() {  
    for (Elevator elevator : this.elevators) {  
        elevator.turnEnd();  
    }  
    elevatorBackUp.turnEnd();  
}
```

```
public void startEvacuation() {  
    if (!this.evacuating) {
```

```
this.evacuating = true;
for (Elevator elevator : this.elevators) {
    elevator.forceOutPeople(true);
}
elevatorBackUp.forceOutPeople(true);
elevatorBackUp.turnEnd();
}
}

public void callElevator(int floor) {
    if (Configuration.LOGGING_ON) {
        logger.info("called elevator from floor " + floor);
    }
    getExternalRequestedFloors().put(floor, true);
}

public ArrayList<Elevator> getElevatorsInFloor(int floor) {
    ArrayList<Elevator> elevatorsInFloor = new ArrayList<>();
    for (Elevator elevator : this.elevators) {
        if (elevator.getCurrentFloor() == floor && elevator.getStatus() !=
ElevatorStatus.BROKEN) {
            elevatorsInFloor.add(elevator);
        }
    }
    if (this.elevatorBackUp.getCurrentFloor() == floor && this.elevatorBackUp.getStatus() !=
ElevatorStatus.OFF) {
        elevatorsInFloor.add(this.elevatorBackUp);
    }
}
```

```
        return elevatorsInFloor;
    }

    public Map<Integer, Boolean> getExternalRequestedFloors() {
        return externalRequestedFloors;
    }

    public void configureRemote(JarvisRemoteControl remote) {
        remotes.add(remote);
        if (Configuration.LOGGING_ON) {
            logger.info("added new remote controller " + remote.toString());
        }
    }

    public void notifyFloorStop(int floor) {
        this.externalRequestedFloors.put(floor, false);
        for (JarvisRemoteControl remote : this.remotes) {
            if (remote.getValue() == floor) {
                remote.notifyElevatorArriving();
            }
        }
    }

    public void notifyFloorMove(int floor) {
        for (JarvisRemoteControl remote : this.remotes) {
            if (remote.getValue() == floor) {
                remote.notifyElevatorLeaving();
            }
        }
    }
}
```



```
    }  
    addMovement();  
    printStatus();  
}  
  
public synchronized void notifyElevatorRepaired() {  
    this.elevatorBackUp.turnOff();  
  
}  
  
public void notifyBreak() {  
    this.elevatorBackUp.turnOn();  
  
}  
  
public HashMap<Integer, Integer> getPeopleInAllFloors() {  
    HashMap<Integer, Integer> mapOfPeopleInFloors = new HashMap<Integer, Integer>();  
    for (JarvisRemoteControl remote : getRemotes()) {  
        mapOfPeopleInFloors.put(remote.getValue(), remote.getPeopleWaiting());  
    }  
    return mapOfPeopleInFloors;  
}  
  
public synchronized ArrayList<Elevator> getElevators() {  
    return elevators;  
}  
  
public ElevatorBackUp getElevatorBackUp() {  
    return elevatorBackUp;
```

```
}

    public synchronized ArrayList<JarvisRemoteControl> getRemotes() {
        return remotes;
    }

    public HospitalFloor getHospitalFloor(int nFloor) {
        return this.hospital.getFloor(nFloor);
    }

    public boolean isEvacuating() {
        return evacuating;
    }

}

package hospital;

import java.io.IOException;
import java.util.logging.FileHandler;
import java.util.logging.Handler;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.logging.SimpleFormatter;

public class Logging {

    public static final String LOG_NAME = "HospitalElevatorSystem";
    private static final String LOG_FILE = "." + Configuration.LOGGING_FILE_NAME;
```

```
private static final Logger logger = Logger.getLogger(LOG_NAME);

private Logging() {
}

public static void initLogger() throws SecurityException, IOException {
    Handler fileHandler = new FileHandler(LOG_FILE, true);
    SimpleFormatter simpleFormatter = new SimpleFormatter();
    fileHandler.setFormatter(simpleFormatter);
    fileHandler.setLevel(Level.INFO);
    logger.addHandler(fileHandler);
}

}

package hospital;

import java.util.logging.Logger;

public class PeopleGenerator extends Thread {

    private static final Logger logger = Logger.getLogger(Logging.LOG_NAME);

    private Hospital hospital;
    private int peopleCounter;

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

```
this.peopleCounter = 0;
}

private String idGenerator() {
    this.peopleCounter++;
    return Person.PREFIX + peopleCounter;
}

private int randomFloor() {
    return (int) (Math.random() * (Configuration.HOSPITAL_FLOOR_MAX + 1));
}

private void waitForPeopleGeneration() {
    double randomTime = (Math.random() * (Configuration.GENERATE_USER_MAX_MS -
Configuration.GENERATE_USER_MIN_MS + 1)
        + Configuration.GENERATE_USER_MIN_MS);

    try {
        Thread.sleep((long) randomTime);
    } catch (InterruptedException ex) {
        if (Configuration.LOGGING_ON) {
            logger.warning("error while waiting next generation: " + ex.getMessage());
        }
    }
}

private Person generate() {

    String id = idGenerator();
    int currentFloor = randomFloor();
```

```
int targetFloor = randomFloor();
while (currentFloor == targetFloor) {
    targetFloor = randomFloor();
}

Person person = new Person(id, hospital.getFloor(currentFloor), targetFloor);
if (Configuration.LOGGING_ON) {
    logger.info("generated person " + this.peopleCounter + "/" +
Configuration.GENERATED_MAX_PEOPLE);
}
return person;
}

@Override
public void run() {
    if (Configuration.LOGGING_ON) {
        logger.info("starts generating people");
    }
    while (this.peopleCounter < Configuration.GENERATED_MAX_PEOPLE &&
!this.hospital.isEvacuating()) {
        waitForPeopleGeneration();
        generate().start();
    }
    if (Configuration.LOGGING_ON) {
        logger.info("ends generating people");
    }
}
}
```

```
package hospital;

import java.util.ArrayList;
import java.util.logging.Logger;

public class Person extends Thread {

    private static final Logger logger = Logger.getLogger(Logging.LOG_NAME);
    private static final String PREFIX = "P";

    private HospitalFloor hospitalFloor;
    private String identifier;
    private int sourceFloor;
    private int floor;
    private int targetFloor;
    private ElevatorDirection direction;

    public void chooseDirection() {
        if (this.floor < this.targetFloor) {
            direction = ElevatorDirection.UP;
        } else if (this.floor > this.targetFloor) {
            direction = ElevatorDirection.DOWN;
        } else {
            direction = ElevatorDirection.NONE;
        }
    }

    public Person(String identifier, HospitalFloor hospitalFloor, int targetFloor) {
```

```
this.identificator = identificator;
this.hospitalFloor = hospitalFloor;
this.sourceFloor = hospitalFloor.getFloor();
this.floor = hospitalFloor.getFloor();
this.targetFloor = targetFloor;
chooseDirection();
if (Configuration.LOGGING_ON) {
    logger.info(this.toString() + " initialized");
}
}

@Override
public String toString() {
    return this.identificator + "->" + this.targetFloor;
}

public void evacuate() {
    this.direction = ElevatorDirection.DOWN;
    this.targetFloor = Configuration.HOSPITAL_FLOOR_MIN;
}

private void waitFloor(Elevator elevator) throws InterruptedException {
    if (Configuration.LOGGING_ON) {
        logger.info(this.toString() + " waiting floor - " + elevator.toString());
    }
    boolean isInTargetFloor = false;
    while (!isInTargetFloor) {
        this.floor = elevator.getCurrentFloor();
    }
}
```

```
this.hospitalFloor = elevator.getHospitalFloor(this.floor);
isInTargetFloor = elevator.getCurrentFloor() == this.targetFloor;
sleep(5);
}
}

private Elevator chooseElevator(ArrayList<Elevator> elevators) {
    Elevator choosenElevator = null;
    for (Elevator elevator : elevators) {
        if (elevator != null) {
            if (elevator.getStatus() != ElevatorStatus.BROKEN && elevator.getStatus() !=
ElevatorStatus.OFF) {
                if (elevator.getDirection() == this.direction || elevator.getDirection() ==
ElevatorDirection.NONE) {
                    choosenElevator = elevator;
                    break;
                } else {
                    if (Configuration.LOGGING_ON) {
                        logger.info(this.toString() + " elevator wrong direction, wait next one: " +
elevator.toString());
                    }
                }
            } else {
                if (Configuration.LOGGING_ON) {
                    logger.info(this.toString() + " elevator broken, wait next one: " +
elevator.toString());
                }
            }
        } else {
            if (Configuration.LOGGING_ON) {
```



```
        logger.warning(this.toString() + " not possible to get elevator");
    }
}
return chosenElevator;
}
```

```
private void waitExitingPeople(Elevator elevator) {
    while (elevator.getStatus() != ElevatorStatus.STOPPED) {
        try {
            sleep(5);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

```
private void waitElevatorFullLeave(Elevator elevator) {
    while (elevator.getStatus() == ElevatorStatus.STOPPED) {
        try {
            sleep(5);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

```
private void waitAllElevatorsLeave(ArrayList<Elevator> elevators) {
    for (Elevator elevator : elevators) {
```

```
while (this.floor == elevator.getCurrentFloor()) {  
    try {  
        sleep(5);  
    } catch (InterruptedException e) {  
        e.printStackTrace();  
    }  
}  
}  
}
```

@Override

```
public void run() {  
    ArrayList<Elevator> elevators;  
    Elevator choosenElevator;  
  
    if (Configuration.LOGGING_ON) {  
        logger.info(this.toString() + " called elevator and start waiting");  
    }  
  
    while (this.floor != this.targetFloor) {  
  
        this.hospitalFloor.callElevator(); // sleep until elevator arrives  
        elevators = this.hospitalFloor.getElevators();  
        choosenElevator = chooseElevator(elevators);  
  
        if (choosenElevator != null) {  
            if (choosenElevator.isEvacuating()) {  
                evacuate();  
            }  
        }  
    }  
}
```

```
waitExitingPeople(choosenElevator);
boolean inside = choosenElevator.enter(this);
if (inside) {
    if (Configuration.LOGGING_ON) {
        logger.info("Person " + this.identificator + ": enter to elevator");
    }
    if (Configuration.LOGGING_ON) {
        logger.info(this.toString() + " enter to elevator");
    }
    try {
        waitFloor(choosenElevator);
    } catch (InterruptedException e) {
        logger.info(toString() + " being evacuating");
    } finally {
        choosenElevator.out(this);
    }

    if (Configuration.LOGGING_ON) {
        logger.info(this.toString() + " go out to floor " + this.floor);
    }
    if (this.floor == this.targetFloor) {
        if (Configuration.LOGGING_ON) {
            logger.info(this.toString() + " is in target floor!");
        }
    } else {
        if (Configuration.LOGGING_ON) {
            logger.info(this.toString() + " is not in target floor yet");
        }
    }
}
```

```
    } else {
        waitElevatorFullLeave(choosenElevator);
        if (Configuration.LOGGING_ON) {
            logger.info(this.toString() + " elevator full, wait next one");
        }
    }

    } else {
        waitAllElevatorsLeave(elevators);
    }

}

if (Configuration.LOGGING_ON) {
    logger.info(this.toString() + " ends");
}
}

public synchronized int getFloor() {
    return floor;
}

public synchronized void setFloor(int floor) {
    this.floor = floor;
}

public HospitalFloor getHospitalFloor() {
    return hospitalFloor;
}
```

```
public void setHospitalFloor(HospitalFloor hospitalFloor) {
    this.hospitalFloor = hospitalFloor;
}

public String getIdentificator() {
    return identificator;
}

public int getTargetFloor() {
    return targetFloor;
}

}

package hospital.dto;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

import hospital.Elevator;
import hospital.Person;

public class ElevatorInfoDTO implements Serializable {

    private static final long serialVersionUID = 1L;
    private String identification;
    private int floor;
    private int peopleCounter;
```

```
private List<Integer> peopleDestinations;

public ElevatorInfoDTO(Elevator elevator) {
    this.identification = elevator.getIdentification();
    this.peopleCounter = elevator.peopleInElevator();
    this.floor = elevator.getCurrentFloor();
    this.peopleDestinations = new ArrayList<>();
    for (Person person : elevator.getSpace()) {
        this.peopleDestinations.add(person.getTargetFloor());
    }
}

public String getIdentification() {
    return identification;
}

public int getPeopleCounter() {
    return peopleCounter;
}

public String getPeopleCounterAsString() {
    return String.valueOf(peopleCounter);
}

public List<Integer> getPeopleDestinations() {
    return peopleDestinations;
}

public String getPeopleDestinationsAsString() {
```

```
String peopleDest = "";
for (int i : this.peopleDestinations) {
    peopleDest = peopleDest + i + ", ";
}
if (peopleDest.length() > 0) {
    peopleDest = peopleDest.substring(0, peopleDest.length() - 2);
}
return peopleDest;
}

public int getFloor() {
    return floor;
}

public String getFloorAsString() {
    return String.valueOf(floor);
}

}
```

5.2. Parte II

```
package hospital.clients;

import java.net.MalformedURLException;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;

public class MainClient {
```

```
public static void main(String args[]) {

    Front clientUI;
    ClientInfoRefresher infoRefresher;
    try {
        clientUI = new Front();
        infoRefresher = new ClientInfoRefresher(clientUI);
        infoRefresher.start();
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                clientUI.setVisible(true);
            }
        });
    } catch (RemoteException | MalformedURLException | NotBoundException e) {
        e.printStackTrace();
    }

}

package hospital.clients;

import java.net.MalformedURLException;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.util.List;
import java.util.Map;
```



```
import hospital.dto.ElevatorInfoDTO;

public class Front extends javax.swing.JFrame {

    private static final long serialVersionUID = 1L;
    private ClientController controller;

    public Front() throws RemoteException, MalformedURLException, NotBoundException {
        initComponents();
        this.controller = new ClientController();
    }

    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jPanel1 = new javax.swing.JPanel();
        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jLabel3 = new javax.swing.JLabel();
        jLabel4 = new javax.swing.JLabel();
        jLabel5 = new javax.swing.JLabel();
        jLabel6 = new javax.swing.JLabel();
        jLabel7 = new javax.swing.JLabel();
        jTextFieldFloorElevator1 = new javax.swing.JTextField();
        jTextFieldFloorElevator2 = new javax.swing.JTextField();
        jTextFieldFloorElevator3 = new javax.swing.JTextField();
        jTextFieldPeopleCounterElevator2 = new javax.swing.JTextField();
        jTextFieldDestinationsElevator1 = new javax.swing.JTextField();
        jTextFieldPeopleCounterElevator3 = new javax.swing.JTextField();
```


```
jButton1 = new javax.swing.JButton();
jTextFieldPeopleCounterElevator1 = new javax.swing.JTextField();
jTextFieldDestinationsElevator2 = new javax.swing.JTextField();
jTextFieldDestinationsElevator3 = new javax.swing.JTextField();
jPanel2 = new javax.swing.JPanel();
jLabel8 = new javax.swing.JLabel();
jLabel9 = new javax.swing.JLabel();
jLabel13 = new javax.swing.JLabel();
jLabel14 = new javax.swing.JLabel();
jLabel15 = new javax.swing.JLabel();
jLabel16 = new javax.swing.JLabel();
jLabel17 = new javax.swing.JLabel();
jLabel18 = new javax.swing.JLabel();
jLabel19 = new javax.swing.JLabel();
jLabel20 = new javax.swing.JLabel();
jLabel21 = new javax.swing.JLabel();
jLabel22 = new javax.swing.JLabel();
jLabel23 = new javax.swing.JLabel();
jLabel24 = new javax.swing.JLabel();
jLabel25 = new javax.swing.JLabel();
jLabel26 = new javax.swing.JLabel();
jLabel27 = new javax.swing.JLabel();
jLabel28 = new javax.swing.JLabel();
jLabel29 = new javax.swing.JLabel();
jLabel30 = new javax.swing.JLabel();
jTextFieldPeopleFloor2 = new javax.swing.JTextField();
jTextFieldPeopleFloor1 = new javax.swing.JTextField();
jTextFieldPeopleFloor3 = new javax.swing.JTextField();
jTextFieldPeopleFloor4 = new javax.swing.JTextField();
```

```
jTextFieldPeopleFloor5 = new javax.swing.JTextField();
jTextFieldPeopleFloor6 = new javax.swing.JTextField();
jTextFieldPeopleFloor7 = new javax.swing.JTextField();
jTextFieldPeopleFloor8 = new javax.swing.JTextField();
jTextFieldPeopleFloor9 = new javax.swing.JTextField();
jTextFieldPeopleFloor10 = new javax.swing.JTextField();
jTextFieldPeopleFloor11 = new javax.swing.JTextField();
jTextFieldPeopleFloor12 = new javax.swing.JTextField();
jTextFieldPeopleFloor13 = new javax.swing.JTextField();
jTextFieldPeopleFloor14 = new javax.swing.JTextField();
jTextFieldPeopleFloor15 = new javax.swing.JTextField();
jTextFieldPeopleFloor16 = new javax.swing.JTextField();
jTextFieldPeopleFloor17 = new javax.swing.JTextField();
jTextFieldPeopleFloor18 = new javax.swing.JTextField();
jTextFieldPeopleFloor19 = new javax.swing.JTextField();
jTextFieldPeopleFloor20 = new javax.swing.JTextField();
jLabel10 = new javax.swing.JLabel();
jTextFieldPeopleFloor0 = new javax.swing.JTextField();
jLabel11 = new javax.swing.JLabel();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

jPanel1.setBackground(new java.awt.Color(255, 255, 255));
jPanel1.setForeground(new java.awt.Color(255, 255, 255));

jLabel1.setFont(new java.awt.Font("Tahoma", 1, 36)); // NOI18N
jLabel1.setForeground(new java.awt.Color(0, 51, 204));
jLabel1.setText("HOSPITAL ELEVATORS");
```



```
jLabel2.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N  
jLabel2.setText("Elevator 1");
```

```
jLabel3.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N  
jLabel3.setText("Elevator 2");
```

```
jLabel4.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N  
jLabel4.setText("Elevator 3");
```

```
jLabel5.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N  
jLabel5.setText("Floor");
```

```
jLabel6.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N  
jLabel6.setText("People counter");
```

```
jLabel7.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N  
jLabel7.setText("Destinations");
```

```
jTextFieldFloorElevator1.setEditable(false);
```


```
jTextFieldFloorElevator2.setEditable(false);
```

```
jTextFieldFloorElevator3.setEditable(false);
```

```
jTextFieldPeopleCounterElevator2.setEditable(false);
```

```
jTextFieldDestinationsElevator1.setEditable(false);
```

```
jTextFieldPeopleCounterElevator3.setEditable(false);
```

```
.addGroup(jPanel1Layout.createSequentialGroup())
    .addGap(50, 50, 50)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

    .addComponent(jLabel6)
    .addComponent(jLabel5)
    .addComponent(jLabel7))

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

    .addGroup(jPanel1Layout.createSequentialGroup())
        .addGap(177, 177, 177)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)

    .addComponent(jTextFieldFloorElevator1)
    .addComponent(jLabel2, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
    .addComponent(jTextFieldPeopleCounterElevator1))
    .addGap(211, 211, 211)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)

    .addComponent(jLabel3, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
    .addComponent(jTextFieldFloorElevator2)
    .addComponent(jTextFieldPeopleCounterElevator2,
javax.swing.GroupLayout.Alignment.TRAILING))
    .addGap(215, 215, 215)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
```


```

        .addComponent(jLabel4, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
        .addComponent(jTextFieldFloorElevator3)
        .addComponent(jTextFieldPeopleCounterElevator3)))
    .addGroup(jPanel1Layout.createSequentialGroup())
    .addGap(100, 100, 100)
    .addComponent(jTextFieldDestinationsElevator1,
javax.swing.GroupLayout.PREFERRED_SIZE, 270, javax.swing.GroupLayout.PREFERRED_SIZE)
    .addGap(27, 27, 27)
    .addComponent(jTextFieldDestinationsElevator2,
javax.swing.GroupLayout.PREFERRED_SIZE, 270, javax.swing.GroupLayout.PREFERRED_SIZE)
    .addGap(36, 36, 36)
    .addComponent(jTextFieldDestinationsElevator3,
javax.swing.GroupLayout.PREFERRED_SIZE, 270,
javax.swing.GroupLayout.PREFERRED_SIZE))))
    .addContainerGap(108, Short.MAX_VALUE))
);
jPanel1Layout.setVerticalGroup(
    jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(jPanel1Layout.createSequentialGroup())

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
G)

        .addGroup(jPanel1Layout.createSequentialGroup())
        .addGap(74, 74, 74)
        .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED_SIZE, 44,
javax.swing.GroupLayout.PREFERRED_SIZE))
        .addGroup(jPanel1Layout.createSequentialGroup())
        .addGap(33, 33, 33)
        .addComponent(jLabel1)))
    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

```



```
.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
```

```
    .addGroup(jPanel1Layout.createSequentialGroup()
```

```
        .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
```

```
            .addComponent(jLabel2)
```

```
            .addComponent(jLabel4)
```

```
            .addComponent(jLabel3))
```

```
        .addGap(33, 33, 33)
```

```
        .addComponent(jLabel5))
```

```
    .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
```

```
        .addComponent(jTextFieldFloorElevator1,
            javax.swing.GroupLayout.PREFERRED_SIZE, 31, javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
        .addComponent(jTextFieldFloorElevator2,
            javax.swing.GroupLayout.PREFERRED_SIZE, 31, javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
        .addComponent(jTextFieldFloorElevator3,
            javax.swing.GroupLayout.PREFERRED_SIZE, 31, javax.swing.GroupLayout.PREFERRED_SIZE)))
```

```
    .addGap(45, 45, 45)
```

```
    .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
```

```
        .addComponent(jLabel6)
```

```
        .addComponent(jTextFieldPeopleCounterElevator1,
            javax.swing.GroupLayout.PREFERRED_SIZE, 31, javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
        .addComponent(jTextFieldPeopleCounterElevator2,
            javax.swing.GroupLayout.PREFERRED_SIZE, 31, javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
        .addComponent(jTextFieldPeopleCounterElevator3,
            javax.swing.GroupLayout.PREFERRED_SIZE, 31, javax.swing.GroupLayout.PREFERRED_SIZE))
```

```
    .addGap(43, 43, 43)
```



```
.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

    .addComponent(jLabel7)

    .addComponent(jTextFieldDestinationsElevator1,
javax.swing.GroupLayout.PREFERRED_SIZE, 31, javax.swing.GroupLayout.PREFERRED_SIZE)

    .addComponent(jTextFieldDestinationsElevator2,
javax.swing.GroupLayout.PREFERRED_SIZE, 31, javax.swing.GroupLayout.PREFERRED_SIZE)

    .addComponent(jTextFieldDestinationsElevator3,
javax.swing.GroupLayout.PREFERRED_SIZE, 31, javax.swing.GroupLayout.PREFERRED_SIZE))

    .addContainerGap(44, Short.MAX_VALUE))

);

jPanel2.setBackground(new java.awt.Color(255, 255, 255));

jLabel8.setText("Floor 1");

jLabel9.setText("Floor 2");

jLabel13.setText("Floor 3");

jLabel14.setText("Floor 4");

jLabel15.setText("Floor 5");

jLabel16.setText("Floor 6");

jLabel17.setText("Floor 7");

jLabel18.setText("Floor 8");
```



```
jLabel19.setText("Floor 9");
```

```
jLabel20.setText("Floor 10");
```

```
jLabel21.setText("Floor 11");
```

```
jLabel22.setText("Floor 12");
```

```
jLabel23.setText("Floor 13");
```

```
jLabel24.setText("Floor 14");
```

```
jLabel25.setText("Floor 15");
```

```
jLabel26.setText("Floor 16");
```

```
jLabel27.setText("Floor 17");
```

```
jLabel28.setText("Floor 18");
```

```
jLabel29.setText("Floor 19");
```

```
jLabel30.setText("Floor 20");
```

```
jTextFieldPeopleFloor2.setEditable(false);
```

```
jTextFieldPeopleFloor1.setEditable(false);
```

```
jTextFieldPeopleFloor3.setEditable(false);
```



```
textFieldPeopleFloor4.setEditable(false);
```

```
textFieldPeopleFloor5.setEditable(false);
```

```
textFieldPeopleFloor6.setEditable(false);
```

```
textFieldPeopleFloor7.setEditable(false);
```

```
textFieldPeopleFloor8.setEditable(false);
```

```
textFieldPeopleFloor9.setEditable(false);
```

```
textFieldPeopleFloor10.setEditable(false);
```

```
textFieldPeopleFloor11.setEditable(false);
```

```
textFieldPeopleFloor12.setEditable(false);
```

```
textFieldPeopleFloor13.setEditable(false);
```

```
textFieldPeopleFloor14.setEditable(false);
```

```
textFieldPeopleFloor15.setEditable(false);
```

```
textFieldPeopleFloor16.setEditable(false);
```

```
textFieldPeopleFloor17.setEditable(false);
```

```
jTextFieldPeopleFloor19.setEditable(false);
```

```
jLabel10.setText("Floor 0");
```

```
jLabel11.setFont(new java.awt.Font("Dialog", 1, 24)); // NOI18N
```

```
jLabel11.setForeground(new java.awt.Color(0, 51, 204));
```

```
jLabel11.setText("People on floor");
```

```
javax.swing.GroupLayout jPanel2Layout = new javax.swing.GroupLayout(jPanel2);
```

```
jPanel2.setLayout(jPanel2Layout);
```


```
jPanel2Layout.setHorizontalGroup(  
    jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
        .addGroup(jPanel2Layout.createSequentialGroup()  
            .addContainerGap(28, Short.MAX_VALUE)
```

```
.addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
```

```
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,  
jPanel2Layout.createSequentialGroup())
```

```
.addGroup(jPanel2Layout.createParallelGroup(GroupLayout.Alignment.TRAILING)
```

```
.addGroup(jPanel2Layout.createSequentialGroup())
    .addComponent(jLabel14)
```



```
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
    .addComponent(jTextFieldPeopleFloor4,
javax.swing.GroupLayout.PREFERRED_SIZE, 170,
javax.swing.GroupLayout.PREFERRED_SIZE))
    .addGroup(jPanel2Layout.createSequentialGroup())
    .addComponent(jLabel11)
    .addGap(45, 45, 45)

.addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
G)
    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel2Layout.createSequentialGroup())
    .addComponent(jLabel13)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
    .addComponent(jTextFieldPeopleFloor3,
javax.swing.GroupLayout.PREFERRED_SIZE, 170,
javax.swing.GroupLayout.PREFERRED_SIZE))
    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel2Layout.createSequentialGroup())
    .addComponent(jLabel8, javax.swing.GroupLayout.PREFERRED_SIZE,
41, javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
    .addComponent(jTextFieldPeopleFloor1,
javax.swing.GroupLayout.PREFERRED_SIZE, 168,
javax.swing.GroupLayout.PREFERRED_SIZE))
    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel2Layout.createSequentialGroup())
    .addComponent(jLabel9, javax.swing.GroupLayout.PREFERRED_SIZE,
41, javax.swing.GroupLayout.PREFERRED_SIZE)

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

```

        .addComponent(jTextFieldPeopleFloor2,
javax.swing.GroupLayout.PREFERRED_SIZE, 168,
javax.swing.GroupLayout.PREFERRED_SIZE))))))

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

.addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

        .addComponent(jLabel19)
        .addComponent(jLabel18)
        .addComponent(jLabel17)
        .addComponent(jLabel16)))

.addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel2Layout.createSequentialGroup()

        .addComponent(jLabel10, javax.swing.GroupLayout.PREFERRED_SIZE, 41,
javax.swing.GroupLayout.PREFERRED_SIZE)

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

        .addComponent(jTextFieldPeopleFloor0,
javax.swing.GroupLayout.PREFERRED_SIZE, 168, javax.swing.GroupLayout.PREFERRED_SIZE)

        .addGap(18, 18, 18)

        .addComponent(jLabel15)

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

        .addComponent(jTextFieldPeopleFloor5,
javax.swing.GroupLayout.PREFERRED_SIZE, 170, javax.swing.GroupLayout.PREFERRED_SIZE)

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


        .addComponent(jLabel20)))

.addGap(18, 18, 18)

.addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

        .addComponent(jTextFieldPeopleFloor10,
javax.swing.GroupLayout.PREFERRED_SIZE, 170, javax.swing.GroupLayout.PREFERRED_SIZE)

```



```
.addComponent(jTextFieldPeopleFloor9,
javax.swing.GroupLayout.PREFERRED_SIZE, 170, javax.swing.GroupLayout.PREFERRED_SIZE)
.addComponent(jTextFieldPeopleFloor8,
javax.swing.GroupLayout.PREFERRED_SIZE, 170, javax.swing.GroupLayout.PREFERRED_SIZE)
.addComponent(jTextFieldPeopleFloor7,
javax.swing.GroupLayout.PREFERRED_SIZE, 170, javax.swing.GroupLayout.PREFERRED_SIZE)
.addComponent(jTextFieldPeopleFloor6,
javax.swing.GroupLayout.PREFERRED_SIZE, 170,
javax.swing.GroupLayout.PREFERRED_SIZE))
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)

.addComponent(jLabel21)
.addComponent(jLabel22)
.addComponent(jLabel23)
.addComponent(jLabel24)
.addComponent(jLabel25))
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING, false)

.addComponent(jTextFieldPeopleFloor12,
javax.swing.GroupLayout.DEFAULT_SIZE, 170, Short.MAX_VALUE)
.addComponent(jTextFieldPeopleFloor11)
.addComponent(jTextFieldPeopleFloor13,
javax.swing.GroupLayout.Alignment.LEADING)
.addComponent(jTextFieldPeopleFloor14,
javax.swing.GroupLayout.Alignment.LEADING)
.addComponent(jTextFieldPeopleFloor15))
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
```

```
.addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
```

```
    .addComponent(jLabel26, javax.swing.GroupLayout.Alignment.TRAILING)
    .addComponent(jLabel27, javax.swing.GroupLayout.Alignment.TRAILING)
    .addComponent(jLabel28, javax.swing.GroupLayout.Alignment.TRAILING)
    .addComponent(jLabel29, javax.swing.GroupLayout.Alignment.TRAILING)
    .addComponent(jLabel30, javax.swing.GroupLayout.Alignment.TRAILING))
    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
```

```
.addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
```

```
    .addComponent(jTextFieldPeopleFloor19,
javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT_SIZE,
170, Short.MAX_VALUE)
```

```
    .addComponent(jTextFieldPeopleFloor18,
javax.swing.GroupLayout.Alignment.TRAILING)
```

```
    .addComponent(jTextFieldPeopleFloor17,
javax.swing.GroupLayout.Alignment.TRAILING)
```

```
    .addComponent(jTextFieldPeopleFloor16,
javax.swing.GroupLayout.Alignment.TRAILING)
```

```
    .addComponent(jTextFieldPeopleFloor20))
    .addGap(27, 27, 27))
```

```
);
```

```
jPanel2Layout.setVerticalGroup(
```

```
    jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
```

```
    .addGroup(jPanel2Layout.createSequentialGroup())
```

```
    .addContainerGap()
```

```
.addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
```

```
    .addComponent(jTextFieldPeopleFloor1,
javax.swing.GroupLayout.PREFERRED_SIZE, 30, javax.swing.GroupLayout.PREFERRED_SIZE)
```



```
.addComponent(jLabel8)
.addComponent(jLabel16)
.addComponent(jLabel21)
.addComponent(jLabel26)
.addComponent(jTextFieldPeopleFloor6,
javax.swing.GroupLayout.PREFERRED_SIZE, 30, javax.swing.GroupLayout.PREFERRED_SIZE)
.addComponent(jTextFieldPeopleFloor11,
javax.swing.GroupLayout.PREFERRED_SIZE, 30, javax.swing.GroupLayout.PREFERRED_SIZE)
.addComponent(jTextFieldPeopleFloor16,
javax.swing.GroupLayout.PREFERRED_SIZE, 30, javax.swing.GroupLayout.PREFERRED_SIZE))

.addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(jPanel2Layout.createSequentialGroup())

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

.addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(jTextFieldPeopleFloor2,
javax.swing.GroupLayout.PREFERRED_SIZE, 30, javax.swing.GroupLayout.PREFERRED_SIZE)
.addComponent(jLabel9)
.addComponent(jLabel17)
.addComponent(jLabel22)
.addComponent(jLabel27)
.addComponent(jTextFieldPeopleFloor7,
javax.swing.GroupLayout.PREFERRED_SIZE, 30, javax.swing.GroupLayout.PREFERRED_SIZE)
.addComponent(jTextFieldPeopleFloor12,
javax.swing.GroupLayout.PREFERRED_SIZE, 30, javax.swing.GroupLayout.PREFERRED_SIZE)
.addComponent(jTextFieldPeopleFloor17,
javax.swing.GroupLayout.PREFERRED_SIZE, 30, javax.swing.GroupLayout.PREFERRED_SIZE))

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



```
.addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
```

```
    .addComponent(jLabel13)
    .addComponent(jTextFieldPeopleFloor3,
javax.swing.GroupLayout.PREFERRED_SIZE, 30, javax.swing.GroupLayout.PREFERRED_SIZE)
    .addComponent(jLabel18)
    .addComponent(jLabel23)
    .addComponent(jLabel28)
    .addComponent(jTextFieldPeopleFloor8,
javax.swing.GroupLayout.PREFERRED_SIZE, 30, javax.swing.GroupLayout.PREFERRED_SIZE)
    .addComponent(jTextFieldPeopleFloor13,
javax.swing.GroupLayout.PREFERRED_SIZE, 30, javax.swing.GroupLayout.PREFERRED_SIZE)
    .addComponent(jTextFieldPeopleFloor18,
javax.swing.GroupLayout.PREFERRED_SIZE, 30, javax.swing.GroupLayout.PREFERRED_SIZE)))
    .addGroup(jPanel2Layout.createSequentialGroup()
        .addGap(32, 32, 32)
        .addComponent(jLabel11, javax.swing.GroupLayout.PREFERRED_SIZE, 30,
javax.swing.GroupLayout.PREFERRED_SIZE)))
    .addGap(15, 15, 15)
```

```
.addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
```

```
    .addComponent(jTextFieldPeopleFloor4,
javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.PREFERRED_SIZE,
30, javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
.addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
```

```
    .addComponent(jLabel19)
    .addComponent(jLabel24)
    .addComponent(jLabel29)
```

```

        .addComponent(jTextFieldPeopleFloor9,
javax.swing.GroupLayout.PREFERRED_SIZE, 30, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(jTextFieldPeopleFloor14,
javax.swing.GroupLayout.PREFERRED_SIZE, 30, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(jTextFieldPeopleFloor19,
javax.swing.GroupLayout.PREFERRED_SIZE, 30, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(jLabel14)))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

.addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

        .addComponent(jLabel15)
        .addComponent(jTextFieldPeopleFloor5,
javax.swing.GroupLayout.PREFERRED_SIZE, 30, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(jLabel20)
        .addComponent(jLabel25)
        .addComponent(jLabel30)
        .addComponent(jTextFieldPeopleFloor10,
javax.swing.GroupLayout.PREFERRED_SIZE, 30, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(jTextFieldPeopleFloor15,
javax.swing.GroupLayout.PREFERRED_SIZE, 30, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(jTextFieldPeopleFloor20,
javax.swing.GroupLayout.PREFERRED_SIZE, 30, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(jLabel10)
        .addComponent(jTextFieldPeopleFloor0,
javax.swing.GroupLayout.PREFERRED_SIZE, 30, javax.swing.GroupLayout.PREFERRED_SIZE))
        .addGap(15, 15, 15))
);

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

```

```

        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(layout.createSequentialGroup()
                    .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addComponent(jPanel2, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                .addContainerGap())
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(layout.createSequentialGroup()
                    .addContainerGap()
                    .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGap(18, 18, 18)
                    .addComponent(jPanel2, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        );

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

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        try {
            this.controller.evacuateSystem();

```

```
} catch (RemoteException e) {  
    // TODO Auto-generated catch block  
    e.printStackTrace();  
}  
}  
  
void setElevatorsInfo(List<ElevatorInfoDTO> elevatorsInfos) {  
    for (ElevatorInfoDTO elevatorInfoDto : elevatorsInfos) {  
        if (elevatorInfoDto.getIdentification().equals("elevator_0")) {  
            this.jTextFieldFloorElevator1.setText(elevatorInfoDto.getFloorAsString());  
  
this.jTextFieldPeopleCounterElevator1.setText(elevatorInfoDto.getPeopleCounterAsString())  
;  
  
this.jTextFieldDestinationsElevator1.setText(elevatorInfoDto.getPeopleDestinationsAsString()  
());  
  
        } else if (elevatorInfoDto.getIdentification().equals("elevator_1")) {  
            this.jTextFieldFloorElevator2.setText(elevatorInfoDto.getFloorAsString());  
  
this.jTextFieldPeopleCounterElevator2.setText(elevatorInfoDto.getPeopleCounterAsString())  
;  
  
this.jTextFieldDestinationsElevator2.setText(elevatorInfoDto.getPeopleDestinationsAsString()  
());  
  
        } else if (elevatorInfoDto.getIdentification().equals("elevator_bkp")) {  
            this.jTextFieldFloorElevator3.setText(elevatorInfoDto.getFloorAsString());  
  
this.jTextFieldPeopleCounterElevator3.setText(elevatorInfoDto.getPeopleCounterAsString())  
;  
}
```

```
this.jTextFieldDestinationsElevator3.setText(elevatorInfoDto.getPeopleDestinationsAsString());
```

```
    }  
  }  
}
```

```
void setPeopleInFloor(Map<Integer, Integer> mapOfPeople) {  
    this.jTextFieldPeopleFloor0.setText(String.valueOf(mapOfPeople.get(0)));  
    this.jTextFieldPeopleFloor1.setText(String.valueOf(mapOfPeople.get(1)));  
    this.jTextFieldPeopleFloor2.setText(String.valueOf(mapOfPeople.get(2)));  
    this.jTextFieldPeopleFloor3.setText(String.valueOf(mapOfPeople.get(3)));  
    this.jTextFieldPeopleFloor4.setText(String.valueOf(mapOfPeople.get(4)));  
    this.jTextFieldPeopleFloor5.setText(String.valueOf(mapOfPeople.get(5)));  
    this.jTextFieldPeopleFloor6.setText(String.valueOf(mapOfPeople.get(6)));  
    this.jTextFieldPeopleFloor7.setText(String.valueOf(mapOfPeople.get(7)));  
    this.jTextFieldPeopleFloor8.setText(String.valueOf(mapOfPeople.get(8)));  
    this.jTextFieldPeopleFloor9.setText(String.valueOf(mapOfPeople.get(9)));  
    this.jTextFieldPeopleFloor10.setText(String.valueOf(mapOfPeople.get(10)));  
    this.jTextFieldPeopleFloor11.setText(String.valueOf(mapOfPeople.get(11)));  
    this.jTextFieldPeopleFloor12.setText(String.valueOf(mapOfPeople.get(12)));  
    this.jTextFieldPeopleFloor13.setText(String.valueOf(mapOfPeople.get(13)));  
    this.jTextFieldPeopleFloor14.setText(String.valueOf(mapOfPeople.get(14)));  
    this.jTextFieldPeopleFloor15.setText(String.valueOf(mapOfPeople.get(15)));  
    this.jTextFieldPeopleFloor16.setText(String.valueOf(mapOfPeople.get(16)));  
    this.jTextFieldPeopleFloor17.setText(String.valueOf(mapOfPeople.get(17)));  
    this.jTextFieldPeopleFloor18.setText(String.valueOf(mapOfPeople.get(18)));  
    this.jTextFieldPeopleFloor19.setText(String.valueOf(mapOfPeople.get(19)));  
    this.jTextFieldPeopleFloor20.setText(String.valueOf(mapOfPeople.get(20)));  
}
```

```

    }

    public ClientController getController() {
        return controller;
    }

    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(Front.class.getName()).log(java.util.logging.Level.SEVERE,
            null, ex);

        } catch (InstantiationException ex) {

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

        } catch (IllegalAccessException ex) {

```

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

```
    } catch (javax.swing.UnsupportedLookAndFeelException ex) {
```

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

```
    }
```

```
//</editor-fold>
```

```
/* Create and display the form */
```

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

```
    public void run() {
```

```
        try {
```

```
            new Front().setVisible(true);
```

```
        } catch (RemoteException | MalformedURLException | NotBoundException e) {
```

```
            // TODO Auto-generated catch block
```

```
            e.printStackTrace();
```

```
        }
```

```
    }
```

```
});
```

```
}
```

```
// Variables declaration - do not modify
```

```
private javax.swing.JButton jButton1;
```

```
private javax.swing.JLabel jLabel1;
```

```
private javax.swing.JLabel jLabel10;
```

```
private javax.swing.JLabel jLabel11;
```

```
private javax.swing.JLabel jLabel13;
```

```
private javax.swing.JLabel jLabel14;
```




```
private javax.swing.JLabel jLabel15;  
private javax.swing.JLabel jLabel16;  
private javax.swing.JLabel jLabel17;  
private javax.swing.JLabel jLabel18;  
private javax.swing.JLabel jLabel19;  
private javax.swing.JLabel jLabel2;  
private javax.swing.JLabel jLabel20;  
private javax.swing.JLabel jLabel21;  
private javax.swing.JLabel jLabel22;  
private javax.swing.JLabel jLabel23;  
private javax.swing.JLabel jLabel24;  
private javax.swing.JLabel jLabel25;  
private javax.swing.JLabel jLabel26;  
private javax.swing.JLabel jLabel27;  
private javax.swing.JLabel jLabel28;  
private javax.swing.JLabel jLabel29;  
private javax.swing.JLabel jLabel3;  
private javax.swing.JLabel jLabel30;  
private javax.swing.JLabel jLabel4;  
private javax.swing.JLabel jLabel5;  
private javax.swing.JLabel jLabel6;  
private javax.swing.JLabel jLabel7;  
private javax.swing.JLabel jLabel8;  
private javax.swing.JLabel jLabel9;  
private javax.swing.JPanel jPanel1;  
private javax.swing.JPanel jPanel2;  
private javax.swing.JTextField jTextFieldDestinationsElevator1;  
private javax.swing.JTextField jTextFieldDestinationsElevator2;  
private javax.swing.JTextField jTextFieldDestinationsElevator3;
```



```
private javax.swing.JTextField jTextFieldFloorElevator1;
private javax.swing.JTextField jTextFieldFloorElevator2;
private javax.swing.JTextField jTextFieldFloorElevator3;
private javax.swing.JTextField jTextFieldPeopleCounterElevator1;
private javax.swing.JTextField jTextFieldPeopleCounterElevator2;
private javax.swing.JTextField jTextFieldPeopleCounterElevator3;
private javax.swing.JTextField jTextFieldPeopleFloor0;
private javax.swing.JTextField jTextFieldPeopleFloor1;
private javax.swing.JTextField jTextFieldPeopleFloor10;
private javax.swing.JTextField jTextFieldPeopleFloor11;
private javax.swing.JTextField jTextFieldPeopleFloor12;
private javax.swing.JTextField jTextFieldPeopleFloor13;
private javax.swing.JTextField jTextFieldPeopleFloor14;
private javax.swing.JTextField jTextFieldPeopleFloor15;
private javax.swing.JTextField jTextFieldPeopleFloor16;
private javax.swing.JTextField jTextFieldPeopleFloor17;
private javax.swing.JTextField jTextFieldPeopleFloor18;
private javax.swing.JTextField jTextFieldPeopleFloor19;
private javax.swing.JTextField jTextFieldPeopleFloor2;
private javax.swing.JTextField jTextFieldPeopleFloor20;
private javax.swing.JTextField jTextFieldPeopleFloor3;
private javax.swing.JTextField jTextFieldPeopleFloor4;
private javax.swing.JTextField jTextFieldPeopleFloor5;
private javax.swing.JTextField jTextFieldPeopleFloor6;
private javax.swing.JTextField jTextFieldPeopleFloor7;
private javax.swing.JTextField jTextFieldPeopleFloor8;
private javax.swing.JTextField jTextFieldPeopleFloor9;
// End of variables declaration
}
```

```
package hospital.clients;

import java.rmi.RemoteException;

import hospital.Configuration;

public class ClientInfoRefresher extends Thread {

    private Front clientUI;
    private ClientController controller;

    public ClientInfoRefresher(Front clientUI) {
        this.clientUI = clientUI;
        this.controller = this.clientUI.getController();
    }

    @Override
    public void run() {
        while (true) {
            try {
                sleep(Configuration.CLIENT_REFRESH_INFO_MS);
                this.clientUI.setPeopleInFloor(this.controller.getPeopleInFloor());
                this.clientUI.setElevatorsInfo(this.controller.getElevatorsInfo());
            } catch (InterruptedException e) {
                e.printStackTrace();
            } catch (RemoteException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```
    }

}

}

package hospital.clients;

import hospital.server.ServerControllerInterfaceRMI;
import hospital.dto.ElevatorInfoDTO;
import hospital.server.NetworkConfig;
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.util.List;
import java.util.Map;

public class ClientController {

    private ServerControllerInterfaceRMI controllerRMI;

    public ClientController() throws RemoteException, NotBoundException,
MalformedURLException {

        this.controllerRMI = (ServerControllerInterfaceRMI)
Naming.lookup(NetworkConfig.CLIENT_CONTROLLER_URI);
    }

    public List<ElevatorInfoDTO> getElevatorsInfo() throws RemoteException {
```

```
        return this.controllerRMI.getElevatorsInfo();
    }

    public void evacuateSystem() throws RemoteException {
        this.controllerRMI.evacuateSystem();
    }

    public Map<Integer, Integer> getPeopleInFloor() throws RemoteException {
        return this.controllerRMI.getPeopleInFloors();
    }
}

package hospital.server;

import hospital.Hospital;
import hospital.Logging;
import hospital.PeopleGenerator;

import java.rmi.Naming;
import java.rmi.registry.LocateRegistry;

public class MainServer {

    public static void main(String[] args) throws Exception {

        Logging.initLogger();
        Hospital hospital = new Hospital();
        PeopleGenerator peopleGenerator = new PeopleGenerator(hospital);
        ServerController serverController = new ServerController(hospital.getJarvisSystem());
```

```
        peopleGenerator.start();
        startServer(serverController);
    }

    private static void startServer(ServerController controller) {
        try {
            ServerController controllerObj = controller;
            LocateRegistry.createRegistry(NetworkConfig.PORT);
            Naming.rebind(NetworkConfig.SERVER_CONTROLLER_URI, controllerObj);
            System.out.println("Started server");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

package hospital.server;

public class NetworkConfig {

    public static final int PORT = 1099;
    public static final String SERVER_CONTROLLER_URI = "///localhost/JarvisSystem";
    public static final String CLIENT_CONTROLLER_URI = "///127.0.0.1/JarvisSystem";
}

package hospital.server;

import hospital.Elevator;
```

```
import hospital.JarvisSystem;
import hospital.dto.ElevatorInfoDTO;

import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

public class ServerController extends UnicastRemoteObject implements
ServerControllerInterfaceRMI {

    private static final long serialVersionUID = 1L;
    private JarvisSystem jarvisSystem;

    public ServerController(JarvisSystem jarvisSystem) throws RemoteException {
        this.jarvisSystem = jarvisSystem;
    }

    @Override
    public List<ElevatorInfoDTO> getElevatorsInfo() {
        List<ElevatorInfoDTO> elevatorsInfosDto = new ArrayList<>();
        for (Elevator elevator : this.jarvisSystem.getElevators()) {
            elevatorsInfosDto.add(new ElevatorInfoDTO(elevator));
        }
        elevatorsInfosDto.add(new ElevatorInfoDTO(this.jarvisSystem.getElevatorBackUp()));
        return elevatorsInfosDto;
    }

    @Override
```

```
public HashMap<Integer, Integer> getPeopleInFloors() {
    return this.jarvisSystem.getPeopleInAllFloors();
}

@Override
public void evacuateSystem() {
    this.jarvisSystem.startEvacuation();
}

}

package hospital.server;

import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.List;
import java.util.Map;

import hospital.dto.ElevatorInfoDTO;

public interface ServerControllerInterfaceRMI extends Remote {

    public List<ElevatorInfoDTO> getElevatorsInfo() throws RemoteException;

    public void evacuateSystem() throws RemoteException;

    public Map<Integer, Integer> getPeopleInFloors() throws RemoteException;

}
```