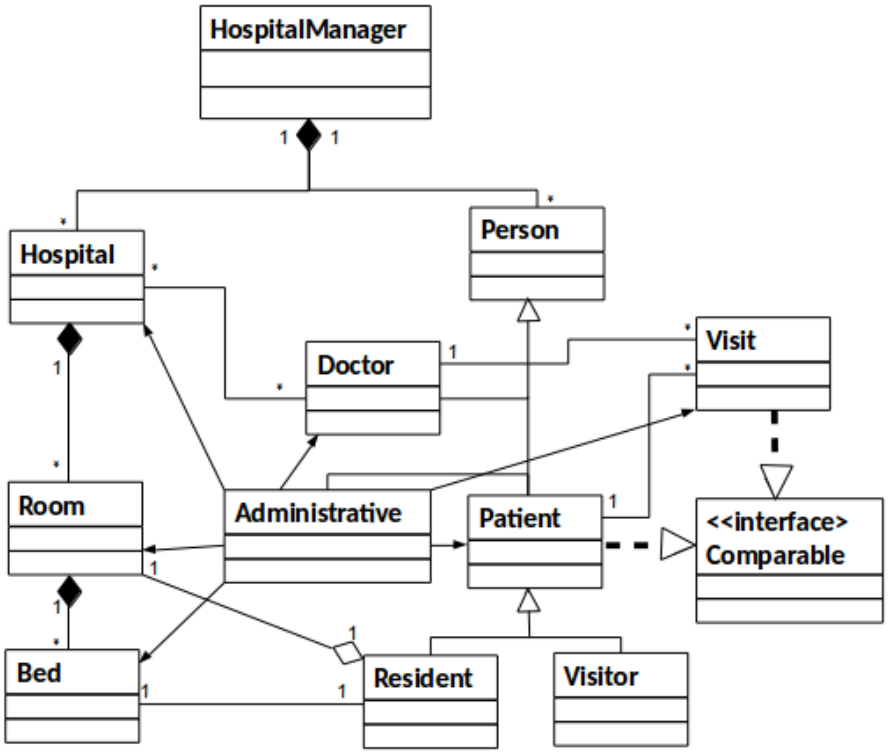
Lab 5
Programación Orientada a Objetos

-Descripción del problema a solucionar

En este laboratorio deberemos implementar el diseño realizado en el seminario 5, este diseño se basa en un Manager de Hospitales (**HospitalManager**) en el que encontramos diferentes clases que mantienen relaciones de herencia con otras y nos permiten implementar correctamente como funcionaría nuestro diseño.

El diseño del que partiremos para implementar esta práctica será el siguiente:



Como podemos ver, tenemos una clase principal llamada **HospitalManager** que gestionará los diferentes hospitales que podemos controlar.

Cada **hospital** tendrá **administrativos, doctores, pacientes, habitaciones** y **visitas**.

Los **administrativos**, los **doctores** y los **pacientes** serán subclases de la clase **Person**, por lo tanto, heredaran de esta.

Como podemos ver, tendremos una clase **Visit** que hará referencia a las visitas de los **doctores** y los **pacientes**, estas serán controladas por los **administrativos** del hospital. Un paciente puede ser un **residente** del hospital o un **visitante**. Una **habitación** estará formada por **camas**.

Lo que debemos hacer es implementar de manera correcta todas estas clases.

- Clase Hospital

Esta será la clase que hará referencia a los hospitales, estos tendrán un **nombre**, **administrativos**, **doctores**, **pacientes**, **habitaciones** y **visitas**.

Por lo tanto, deberemos implementar esta clase de la siguiente manera:

```
public class Hospital{

    private LinkedList< Administrative > admins;
    private LinkedList< Doctor > doctors;
    private LinkedList< Patient > patients;
    private LinkedList< Room > rooms;
    private LinkedList< Visit > visits;
    private String name;

    public Hospital( String name ){

        this.name = name;
        admins = new LinkedList< Administrative >();
        doctors = new LinkedList< Doctor >();
        patients = new LinkedList< Patient >();
        rooms = new LinkedList< Room >();
        visits = new LinkedList< Visit >();
    }
```

Como podemos ver, añadimos las listas de las clases que hemos mencionado anteriormente como atributos de nuestra clase.

También vemos el método constructor de la clase en el que asignamos un nombre al hospital e inicializamos las listas de los elementos mencionados.

```
public void addAdmin( Administrative a ){ admins.add(a);}

public Administrative getAdmin( int idx ){ return admins.get(idx);}

public void addDoctor( Doctor d ){ doctors.add(d);}

public Doctor getDoctor( int idx ){ return doctors.get(idx);}

public void addRoom(int id){ rooms.add(new Room(id));}

public Room getRoom( int idx ){ return rooms.get(idx);}

public LinkedList< Room > getRooms(){ return rooms;}

public void addResident(  int id, String name, int age ){
    patients.add(new Resident(id, name, age));
}
public void addVisitor( int id, String name, int age ){
    patients.add(new Visitor(id, name, age));
}
public void addVisit( Visit v ){
    visits.add(v);
}
public Visit getVisit( int idx ){
    return visits.get(idx);
}
public Patient getPatient( int idx ){
    return patients.get(idx);
}
public void deletePatient( int idx ){
    patients.remove(idx);
}
```

Como podemos ver, estos serán algunos de los **getters** y **setters** de nuestra clase **Hospital**.

Observamos como podemos gestionar las listas pudiendo añadir, eliminar y tomar un elemento de la lista con la que trabajamos.

En cuanto a los **pacientes**, añadimos métodos para poder añadir pacientes de tipo residente y de tipo visitante.

Todos estos métodos nos permitirán controlar las listas de los elementos que hemos mencionado anteriormente de manera correcta y eficiente.

```
public void assignBeds( int adminIdx ){

    for(Patient p: patients){

        if(p instanceof Resident){

            Resident r = (Resident)p;
            Administrative a = getAdmin(adminIdx);
            a.assignBed(r);
        }
    }
}

public void sortPatients(){
    Collections.sort(patients);
}

public String toString(){

    String admins2 = "";
    String doctors2 = "";
    String patients2 = "";
    String rooms2 = "";

    for(Administrative a: admins){
        admins2 = (admins2+a+"\n");
    }
    for(Doctor d: doctors){
        doctors2 = (doctors2+d+"\n");
    }
    for(Patient p: patients){
        patients2 = (patients2+p+"\n");
    }
    for(Room r: rooms){
        rooms2 = (rooms2+r+"\n");
        rooms2 = (rooms2+r.listBeds()+"\n");
    }

    return (name +"\n"+"Administratives: \n"+admins2+"\nDoctors:\n"+doctors2
}
```

Por último, encontramos 3 métodos para nuestra clase **Hospital** que nos permitirán asignar camas a los pacientes de tipo residente, ordenar los pacientes e imprimir por pantalla la información del hospital.

En primer lugar, encontramos el método **assignBeds**(), este nos permitirá asignar las camas a los pacientes del hospital según su disponibilidad llamando al método **assignBed** de la clase **Administrative** que veremos más adelante.

Seguidamente, podemos ver el método **sortPatients()**, este método llama al método sort de la clase **Collections** de Java y como los pacientes implementarán el método compareTo de la interfaz **Comparable**, los ordenará según la implementación de este método. En nuestro caso, como veremos más adelante, los ordenará de menor a mayor edad.

Por último, encontramos el método **toString()**, este método nos permite imprimir por pantalla la información del Hospital, este método lo veremos mucho a lo largo de la práctica debido a que lo usaremos para realizar la misma función en las demás clases de nuestra implementación.

- Clase Person

Esta será la clase de la que heredarán las clases **Administrative**, **Doctor** y **Patient**. Estas clases están agrupadas en esta súper-clase debido que tienen atributos compartidos, los atributos de una persona. La implementación de esta clase sería la siguiente:

```
public class Person{

    public int id;
    public String name;

    public Person( int id, String name ){
        this.id = id;
        this.name = name;
    }

    public int getID(){
        return id;
    }

    public String getName(){
        return name;
    }

    public void setID( int id ){
        this.id = id;
    }

    public void setName( String name ){
        this.name = name;
    }

    public String toString(){
        return ("Person "+ name);
    }
}
```

Como podemos ver, esta clase tendrá dos atributos, un nombre y un identificador.

Podemos ver como los métodos que encontramos serían, el constructor, los **getters** y **setters** correspondientes a los atributos de la clase y el método **toString** que hemos mencionado anteriormente.

- Clase Doctor

Esta será la clase que hará referencia a los doctores del **hospital**, será una subclase de la clase **Person**, por lo tanto, la extenderá y su implementación será la siguiente:

```
public class Doctor extends Person{

    private LinkedList< String > specialities;
    private LinkedList< Visit > visits;

    public Doctor( int id, String name ){
        super(id, name);
        specialities = new LinkedList<String>();
        visits = new LinkedList<Visit>();
    }

    public void addSpeciality( String s ){ specialities.add(s);}

    public void addVisit( Visit v ){ visits.add(v);}

    public void removeVisit( Visit v ){ visits.remove(v);}

    public LinkedList<Visit> getVisits(){ return visits;}
```

Como podemos ver, tenemos dos atributos nuevos, una lista de especialidades y una lista de visitas del doctor.

Podemos observar los métodos necesarios, un constructor de la clase, un método para añadir especialidades (**addSpeciality**), métodos para gestionar las visitas (**addVisit** y **removeVisit**) y un método para tomar la lista de visitas (**getVisits**).

```java
public void listSpecialities(){

    System.out.println("Doctor "+ name + "(ID " + id + ") has specialities:");

    for(String s: specialities){
        System.out.println(s);
    }
}
public void listVisits(){

    System.out.println("Doctor "+ name + "(ID " + id + ") has the following visits:\n");

    for(Visit v: visits){
        System.out.println(v);
    }
}
public String toString(){
    return ("Doctor "+ name + "(ID " + id + ")");
}
```

Podemos ver como también tenemos 3 métodos adicionales que nos permitirán imprimir las especialidades del doctor, las visitas e imprimir por pantalla la información del doctor.

El primer método, llamado **listSpecialities()**, nos permite imprimir por pantalla las especialidades del doctor de manera que se imprime el nombre del doctor, su identificador y sus especialidades por pantalla.

El segundo método, llamado **listVisits()**, funciona exactamente igual que el método anterior pero en este caso imprimimos por pantalla las visitas que tiene el **Doctor** por hacer.

Por último, tenemos el método que hemos mencionado anteriormente, llamado **toString()**, este método nos permitirá imprimir por pantalla la información del **Doctor**.

- Clase Administrative

Esta será la clase que hará referencia al **personal administrativo** del **Hospital**, esta subclase de **Person** simplemente añadirá como atributo el hospital en el que trabaja el administrativo y algunos métodos para trabajar con esta clase correctamente. La implementación de esta clase será la siguiente:

```
public class Administrative extends Person{

    private Hospital hospital;

    public Administrative( int id, String name, Hospital hospital ){
        super(id, name);
        this.hospital = hospital;
    }

    public void addVisit( Date d, String s, Doctor doc, Patient p ){
       hospital.addVisit(new Visit(d, s, doc, p));
       p.addVisit(new Visit(d, s, doc, p));
    }
```

En primer lugar, debido a que es una subclase de la clase **Person**, la clase **Administrative** extenderá la clase **Person**. Seguidamente, podemos ver el método constructor de la clase en el que se llama al método constructor de **Person** usando **super()**, y un método para añadir visitas debido a que el administrativo será el encargado de realizar esta tarea.

```
public boolean assignBed( Resident resident  ){

    LinkedList< Room > rooms = hospital.getRooms();

    Boolean solved = false;

    for(Room r: rooms){

        if(r.isAvailable()){

            Bed b = r.getAvailableBed();

            resident.assignRoom(r);
            resident.assignBed(b);

            b.assignResident(resident);

            System.out.println(toString() + " has assigned bed to\n" + resident.toString());

            solved = true;
            break;
        }
    }

    if(!solved){

        System.out.println(toString() + "has not found bed for\n"+ resident.toString());
    }
    return solved;
}
```

El método **assignBed** nos permitirá asignar una cama a un paciente de tipo residente de manera correcta. Lo que hacemos es iterar por las habitaciones del **Hospital** y comprobamos si están disponibles, es decir, que hay camas libres, si es así, asignamos la habitación y la cama al residente e imprimimos por pantalla el resultado. En caso contrario el residente no tendrá cama y lo imprimiremos por pantalla también.

Por último, encontramos el método **toString** que se implementaría de la siguiente manera:

```
public String toString(){
    return ("Administrative " + name + "(ID " + id + ")");
}
```