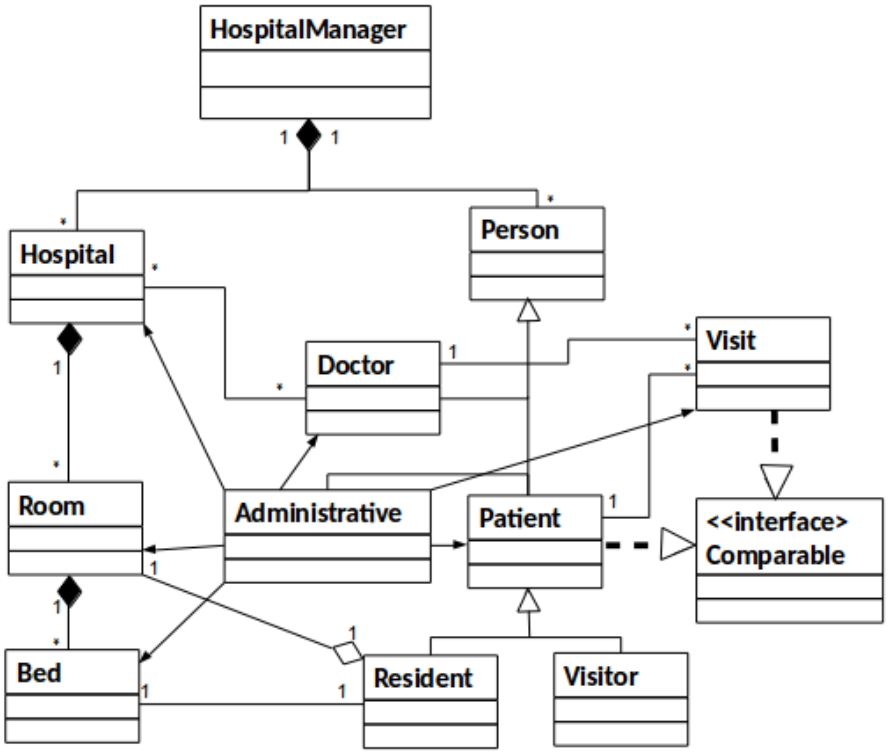
Lab 5
Programación Orientada a Objetos

-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 + ")");
}
```

- Clase Patient

Esta será la clase referente a los pacientes del Hospital, esta clase extenderá la clase Person e implementará la interfaz Comparable para poder implementar el método compareTo y ordenar los pacientes por su edad. La implementación de esta clase será la siguiente:

```
public class Patient extends Person implements Comparable<Patient>{

    protected Date admissionDate;
    protected Integer age;
    protected LinkedList< Visit > visits;

    public Patient( int id, String name, int age ){
        super(id, name);
        this.age = age;
        visits = new LinkedList< Visit >();
    }

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

    public void setAdmissionDate( Date d ){ admissionDate = d;}

    public Date getAdmissionDate(){ return admissionDate;}

    public void setAge( Integer age ){ this.age = age;}

    public Integer getAge(){ return age;}
```

Como podemos ver, tenemos 3 nuevos atributos, la **fecha de admisión** del paciente, su edad y la **lista de visitas** que le han realizado.

Podemos observar el método constructor de la clase y los **getters** y **setters** correspondientes a los atributos de la clase.

También encontramos dos métodos más en esta clase que nos permitirán comparar los pacientes por edad e imprimir por pantalla la información de estos.

```
public int compareTo( Patient p ){

    if(p.getAge() - getAge() > 0){
        return -1;
    }
    if(p.getAge() - getAge() < 0){
        return 1;
    }
    return 0;
}

public String toString(){
    return ("Patient "+ name + "(ID "+ id +", age "+ age + ").");
}
```

Como vemos esta será la implementación del método **CompareTo** que dado un paciente devolverá un entero diferente (**-1, 0 ,1**) según si el paciente con el que comparamos es menor, igual o mayor que el de la instancia que está aplicando el método.

Por último, podemos ver el método **toString** que nos permite imprimir por pantalla la información del paciente (**nombre**, **identificador** y **edad**).

- Clase Resident

Esta será la clase correspondiente a los pacientes de tipo residente, estos pacientes tendrán una cama asignada en una habitación del hospital (en el caso de que puedan tenerla), la implementación de esta clase será la siguiente:

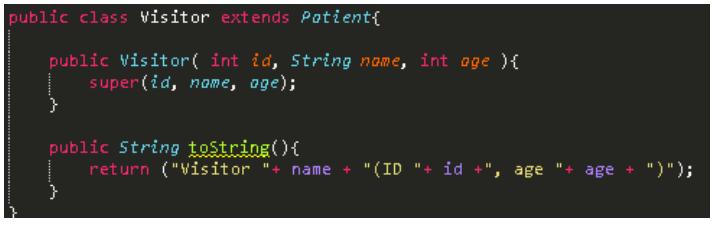
```
public class Resident extends Patient{

    private Room room;
    private Bed bed;

    public Resident ( int id, String name, int age ){
        super(id, name, age);
        room = null;
        bed = null;
    }

    public void assignRoom( Room r ){ room = r;}

    public void assignBed( Bed b ){ bed = b;}

    public Doctor getDoctor(){ return visits.get(0).getDoctor();}

    public String toString(){

        if((visits.size() == 0)){

            if(room == null){
                return ("Resident "+ name + "(ID "+ id +", age "+ age + ") and ha
            }
            return ("Resident "+ name + "(ID "+ id +", age "+ age + ") who is ass
        }
        return ("Resident "+ name + "(ID "+ id +", age "+ age + ") who is assigne
    }
}
```

Como podemos ver, la clase **Resident** extiende la clase **Patient** debido a que se trata de una subclase de **Patient**. Podemos observar el método constructor y los métodos **getters** y **setters** de la clase. Tenemos un método **getter** llamado **getDoctor**, este método nos devuelve el primer doctor que visitó al residente dado que este será su doctor.

Por último, encontramos el método **toString** que, como hemos mencionado anteriormente, imprime por pantalla la información del **Residente** en este caso. Este método tiene una variante y es que el residente puede tener o no **doctor** y puede tener o no una **cama**, por lo tanto, deberemos comprobar la situación del residente. Si el tamaño de su lista de **visitas** es **0** entonces no tiene un doctor asignado y si su habitación tiene valor **null**, no tendrá una habitación asignada.

- Clase Visitor

Esta será la clase correspondiente a los pacientes de tipo visitante, estos no tendrán una habitación asignada y no añadirán nada nuevo a la clase Patient, simplemente nos servirá para distinguir a los pacientes.



```
public class Visitor extends Patient{

    public Visitor( int id, String name, int age ){
        super(id, name, age);
    }

    public String toString(){
        return ("Visitor "+ name + "(ID "+ id +", age "+ age + ")");
    }
}
```

Podemos ver el método **constructor** de la clase y su método **toString**.

- Clase Room

Esta será nuestra clase correspondiente a las habitaciones del hospital, en esta clase tendremos como atributos una lista de camas y un identificador de la habitación. La implementación de esta clase será la siguiente:

```
public class Room{

    private LinkedList< Bed > beds;
    private int roomID;

    public Room( int room ){
        roomID = room;
        beds = new LinkedList< Bed >();
    }

    public void addBed( int bedID ){ beds.add(new Bed(bedID, this));}

    public Bed getBed( int idx ){ return beds.get(idx);}

    public Bed getAvailableBed(){

        for(Bed b : beds){
            if(b.isAvailable()){
                return b;
            }
        }
        return null;
    }

    public boolean isAvailable(){ return getAvailableBed()!=null;}

    public String listBeds(){

        String beds2 = "";
        for(Bed b: beds){
            beds2 = (beds2+b+"\n");
        }
        return beds2;
    }

    public String toString(){
        return ("Room "+ roomID);
    }
}
```

Como podemos ver, tenemos el método constructor de la clase y diferentes métodos. Tenemos un método que nos permite añadir camas a la instancia de la habitación en la que estamos (**addBed**), un método que nos permite tomar una cama mediante el indice de la lista de camas en el que se encuentra (**getBed**), un método que nos permite tomar la cama disponible de una lista de camas (**getAvailableBed**) usando el método de debajo para saber si una cama está disponible (**isAvailable**). Una cama estará disponible si su valor es **null**. También tendremos un método que nos permitirá listar las camas de una habitación (**listBeds**) y, por último, el método **toString** que nos permite imprimir la habitación y su identificador.

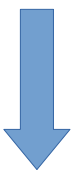
- Clase Bed

Esta será la clase correspondiente a las camas del hospital, cada cama tendrá como atributos un identificador, una habitación a la que corresponde y un residente asignado. La implementación de esta clase será la siguiente:

```
package hospitalmanager;

public class Bed{

    private int bedID;
    private Room room;
    private Resident resident;

    public Bed( int id, Room r ){
        bedID = id;
        room = r;
        resident = null;
    }

    public void assignRoom( Room r ){ room = r;}

    public void assignResident( Resident r ){ resident = r;}

    public void release(){ resident = null;}

    public int getBedID(){ return bedID;}

    public boolean isAvailable(){ return resident == null;}

    public String toString(){ return ("Bed " + bedID);}
}
```

Como podemos observar, tenemos el método constructor de la clase y diferentes métodos más. Podemos ver los **getter** y **setters** correspondientes a los atributos de la clase, seguidamente un método que nos permite liberar una cama (**release**) y un método que nos permite saber si la cama esta disponible (**isAvailable**), una cama está disponible tiene el valor de su residente como **null**.

Por último, encontramos el método **toString** que nos permite imprimir por pantalla la información de la cama.



- Clase Visit

Esta será la clase correspondiente a las visitas. Cada visita tendrá como atributos una fecha, un resumen de la visita, un doctor que realiza la visita y un paciente que recibe la visita. La implementación de esta clase será la siguiente:

```
public class Visit{

    Date date;
    String summary;
    Doctor doctor;
    Patient patient;

    public Visit( Date d, String s, Doctor doc, Patient p ){
        date = d;
        summary = s;
        doctor = doc;
        patient = p;
    }

    public Date getDate(){ return date;}

    public String getSummary(){ return summary;}

    public Doctor getDoctor(){ return doctor;}

    public Patient getPatient(){ return patient;}

    public String toString(){ return("Visit with doc: " + doctor.toString());}
}
```

Como podemos ver, tenemos el método constructor de la clase en el que asignamos los valores a los atributos de la clase con los valores que vienen dados por parámetro del método. También observamos los **getters** correspondientes a cada atributo de la clase y el método **toString** que nos permitirá imprimir por pantalla la información de la visita (el **doctor** que la realiza).

- Clase Date

Esta será la clase correspondiente a la fecha. Esta clase nos permitirá tener en cuenta la fecha en la que nos encontramos para poder realizar las visitas de manera correcta y eficiente. La implementación de esta clase será la siguiente:

```
public class Date{

    private int day;
    private int month;
    private int year;

    Date(int d, int m, int y){
        day = d;
        month = m;
        year = y;
    }

    public int getDay(){ return day;}

    public int getMonth(){ return month;}

    public int getYear(){ return year;}

    public void setDay(int d){ day = d; }

    public void setMonth(int m){ month = m;}

    public void setYear(int y){ year = y;}
}
```

Como vemos, tendremos como atributos el **día**, el **mes** y el **año** de la **fecha**, el método constructor les asignará unos valores en concreto y después encontramos un **getter** y un **setter** para cada atributo de nuestra clase.

En esta práctica, la clase Date no implementa la interfaz Comparable debido a que no tenemos que ordenar ningún elemento por la fecha.

- Clase HospitalManager

Esta será la clase que nos permitirá tener el control de los diferentes hospitales que manejamos, también funcionará como **main** en el que crearemos instancias de nuestras clases principales y comprobaremos que todo funcione correctamente. La implementación de esta clase será la siguiente:

```
public class HospitalManager{

    private static LinkedList< Hospital > hospitals;
    private static LinkedList< Doctor > doctors;
    private static LinkedList< Administrative > administratives;
    public static Date currentDate;

    public HospitalManager(){
        hospitals = new LinkedList< Hospital >();
        doctors = new LinkedList< Doctor >();
        administratives = new LinkedList< Administrative >();
    }

    public void addHospital( String name ){
        hospitals.add( new Hospital( name ) );
    }

    public void addDoctor( int id, String name ){
        doctors.add( new Doctor( id, name ) );
    }

    public void addAdministrative( int id, String name, Hospital h ){
        administratives.add( new Administrative( id, name, h ) );
    }

    public Hospital getHospital( int idx ){ return hospitals.get( idx ); }

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

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

En esta parte podemos ver los atributos y el método constructor de la clase, además de algunos métodos como **addHospital** que nos permite añadir un nuevo hospital, **addDoctor** que nos permite añadir un doctor, **addAdministrative** que nos permite añadir un nuevo administrativo y los **getters** correspondientes a los hospitales, doctores y administrativos de las listas que vienen dadas por los atributos.

```
public static void dayPass(){

    int currentDay = currentDate.getDay();
    int currentMonth = currentDate.getMonth();
    int currentYear = currentDate.getYear();

    currentDay++;

    if(currentDay==32){

        currentDay = 1;
        currentMonth++;
        if(currentMonth == 13){

            currentMonth = 1;
            currentYear++;
        }
    }
    for(Doctor d: doctors){

        for(Visit v: d.getVisits()){

            if(currentDate.equals(v.date)){

                Patient patient = v.patient;
                patient.addVisit(v);
                d.removeVisit(v);
            }
        }
    }
    currentDate.setDay(currentDay);
    currentDate.setMonth(currentMonth);
    currentDate.setYear(currentYear);
}
```

El método **dayPass**, nos permite actualizar la fecha en la que estamos.

Al actualizar la fecha, comprobamos si hay alguna visita con esa fecha y si la hay la realizamos.

Para realizar la visita quitamos la visita de la lista de visitas pendientes del doctor y añadimos la visita a la lista de visitas realizadas del paciente.

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

    currentDate = new Date(29,10,2020);

    System.out.println("The current date is " + currentDate.getDay()+"/"+c

    HospitalManager hm = new HospitalManager();

    hm.addHospital( "Hospital Sant Joan de Deu" );
    hm.addHospital( "Hospital de Barcelona" );

    hm.addDoctor( 1, "Joline" );
    hm.addDoctor( 2, "Antoine" );

    hm.addAdministrative( 3, "Clarise", hm.getHospital( 0 ) );
    hm.addAdministrative( 4, "Pere", hm.getHospital( 1 ) );

    dayPass();

    for( int id = 0; id < 2; id++ ){
        for( int h = 0; h < 2; h++ ){
            hm.getHospital( h ).addRoom( id );
            for( int b = 0; b < 2; b++ ){
                hm.getHospital( h ).getRoom( id ).addBed( b );
            }
        }
    }

    hm.getHospital( 0 ).addAdmin( hm.getAdmin( 0 ) );
    hm.getHospital( 1 ).addAdmin( hm.getAdmin( 1 ) );

    hm.getHospital( 0 ).addDoctor( hm.getDoctor( 0 ) );
    hm.getHospital( 0 ).addDoctor( hm.getDoctor( 1 ) );
    hm.getHospital( 0 ).getDoctor( 0 ).addSpeciality( "General" );
    hm.getHospital( 0 ).getDoctor( 1 ).addSpeciality( "General" );
    hm.getHospital( 0 ).getDoctor( 1 ).addSpeciality( "Cardiologist" );
    hm.getHospital( 1 ).addDoctor( hm.getDoctor( 0 ) );
```

En esta parte realizamos la implementación del **main** de la clase, para ello inicializamos la fecha actual **currentDate**, creamos una instancia de **hospital manager** y empezamos a añadir hospitales, doctores, administrativos, habitaciones y camas a las listas de nuestros atributos.

```
dayPass();

for( int i = 0; i < 2; i++ ){
    Doctor d = hm.getDoctor( i );
    d.listSpecialities();
    d.listVisits();
    System.out.println();
}
System.out.println();

dayPass();

hm.getHospital( 0 ).addResident( 87634, "Jaume", 19 );
hm.getHospital( 0 ).addResident( 34532, "Monica", 25);
hm.getHospital( 0 ).addResident( 62452, "German", 50);
hm.getHospital( 0 ).addResident( 21411, "Maria", 37 );
hm.getHospital( 0 ).addResident( 12999, "Francesc", 88);
hm.getHospital( 0 ).addVisitor( 78678, "Carme", 63);

hm.getHospital( 1 ).addVisitor( 12841, "Xavi", 43 );
hm.getHospital( 1 ).addVisitor( 26256, "Fatima",65 );
hm.getHospital( 1 ).addVisitor( 62213, "Johan",22 );
hm.getHospital( 1 ).addVisitor( 26268, "Johanna",10 );
hm.getHospital( 1 ).addVisitor( 99887, "Jan",90 );

hm.getHospital( 0 ).getAdmin( 0 ).addVisit( new Date(1,12,2020), "Is a cold",
hm.getHospital( 0 ).getAdmin( 0 ).addVisit( new Date(2,12,2020), "Undefined,
hm.getHospital( 1 ).getAdmin( 0 ).addVisit( new Date(3,12,2020), "Is a cold",

hm.getHospital( 0 ).assignBeds( 0 );
hm.getHospital( 0 ).sortPatients();
hm.getHospital( 1 ).sortPatients();

dayPass();
```

Seguidamente, listamos por pantalla los doctores con sus visitas y especialidades, después añadimos pacientes de los dos tipos, tanto Residentes como visitantes. Añadimos visitas y asignamos camas a los pacientes y los **ordenamos** por edad.

```
System.out.println();

for( int i = 0; i < 2; i++ ){
    System.out.println( hm.getHospital( i ) );
}

System.out.println("The current date is " + currentDate.getDay()
```

Por último, imprimimos por pantalla los hospitales y la fecha actual.

- Conclusiones, soluciones alternativas y aportaciones

Las conclusiones que hemos sacado al realizar la práctica han sido que nos ha parecido una práctica más sencilla que las anteriores. Nos ha facilitado mucho la tarea el hecho de que se nos facilitara código para su realización debido a que se pierde mucho tiempo escribiendo código que no tiene complejidad sino que es para ordenar el programa.

En lo que se refiere a la **solución** que hemos realizado, creemos que se podría haber planteado de manera diferente el tema de la **administración**. La administración podría realizar las **cuentas** de los pacientes que entran y salen del hospital, del tiempo que lleva cada paciente y de su historial médico y los doctores tener en cuenta las **visitas** que tienen y las que hacen por su cuenta sin necesidad de pasar por la clase de la Administración.

En cuanto a la participación de esta práctica, **Oriol Estabanell** ha realizado la parte de las clases **Doctor, Bed, Room, Visitor y Visit**, y **Albert Gubau** ha realizado la parte de **Person, Administrative, Patient, Hospital y Date** . Tanto la ordenación de los pacientes como la creación del main y el informe de la práctica han sido realizadas de manera conjunta.