Pau Cobacho, Arnau Adan
POO

Lab session 5: Create a Complete Application

Hospital Manager

INTRODUCCIÓ DEL PROBLEMA

L'objectiu per aquesta sessió de laboratori és implementar el codi del seminari 5. La solució resultarà en una aplicació orientada a objectes que simularà un programa de gestió d'hospitals. Haurem d'omplir els buits en el codi proporcionat per a que el programa compili i executi tal com s'espera.

L'aplicació en questió haurà de permetre afegir hospitals, doctors amb els seus camps d'especialització, administradors, habitacions amb els seus llits corresponents, assignar administradors i doctors als hospitals on treballen, afegir residents i visitants a cada hospital, crear visites, assignar llits als residents i comprovar que tot s'ha assignat correctament . També necessitarà treballar tenint en compte la data actual (que incrementarà a mesura que s'executi el programa) i ordenar els pacients i visites segons edat i data, respectivament.

El disseny de la solució es basarà en el diagrama de classes proposat a l'enunciat (figura 1). Veurem doncs que el disseny es basa en en 6 classes principals: HospitalManager, Hospital, Person, Visit, Room i Bed. També apareix la interfície Comparable que ens permetrà definir les funcions a implementar per odrenar visites i pacients. La classe Person tindrà diverses classes filles que permetràn especialitzar a les persones segons la seva funció dins l'hospital.

Per tant, tenim:

- HospitalManager: encarregada de manegar els diferents hospitals.
- Hospital: encarregada de definir les característiques de l'hospital i tots els comportaments necessaris per poder manegar-lo.
- Person: representant una persona de l'hospital.
    - Doctor: representant un doctor de l'hospital.
    - Adminsitrative: representant un administrador de l'hospital.
    - Patient: representant un pacient de l'hospital.
        - Resident: representant un pacient del tipus resident.
        - Visitor: representant un pacient del tipus visitant.
- Visit: encarregada de definir una visita d'hospital.
- Room: encarregada de representar una habitació de l'hospital
- Bed: encarregada de representar un llit de l'hospital.

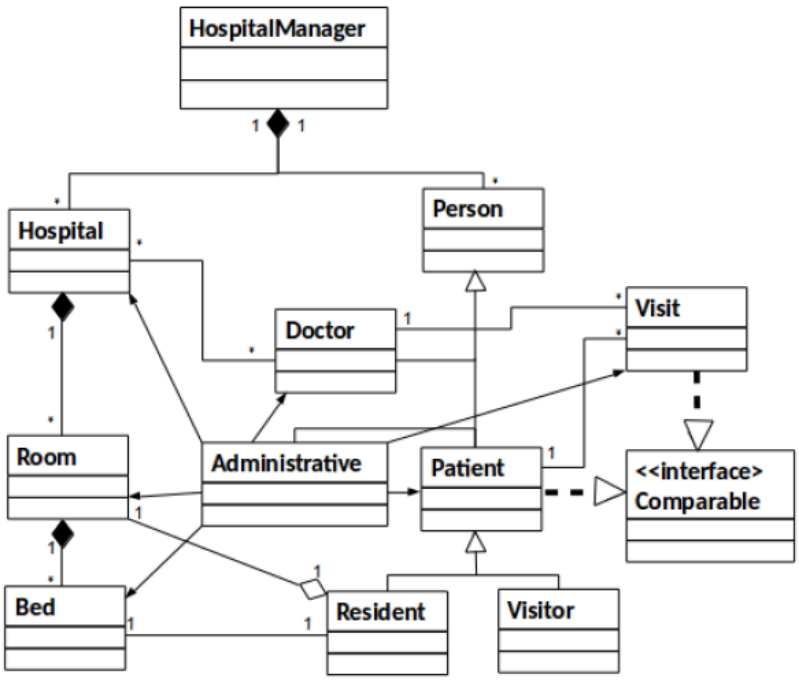

Figura 1: Diagrama de classes per l'aplicació HospitalManager

A continuació expliquem els detalls per cada classe, repassant els mètodes i atributs pertinents.

CLASSE PERSON _____

Cada persona serà representada pel seu nom i nombre identificador. Tindrem tres tipus diferents de persones dins l'hospital, per tant, usarem el concepte d'herència per especialitzar la classe definint les filles Doctor, Administrative i Patient.

Llavors els atributs de Peron seràn 2: **name** (del tipus String representant el nom de la persona) i **id** (del tipus enter representant el nombre identificador de la persona).

En quant als mètodes, per definir la morgologia de cada persona implementarem un constructor que assignarà els atributs a cada instància. També establirem getters i setters que ens serviràn per obtenir i establir la infromació sobre els atributs: **getID( )**, **getName( )**, **setID( )** i **setName( )**. Implementarem, a més a més, el mètode **toString( )**, que ens servirà per obtenir en format String la el valor dels atributs.

Ara, detallem les classes filles representant cada tipus de persona dins l'hospital:

- **DOCTOR**: l'objecte representa la persona dins l'hospital que exerceix el càrrec de doctor. Vindrà definit per les caracteístiques del seu pare (Person) i per una llista amb els seus camps d'especialització **specialities** i un altra per les visites realitzades als pacients que ha atés fins el moment **visits**. Pel seu comportament definirem el constructor i els mètodes necessaris per afegir-li visites i especialitats (**addVisit(** Visit v **)** i **addSpeciality(** String s **)**) i per llistar aquestes (**listVisits( )** i **listSpecialities( )**). També implementarem un mètode **toString( )** per obtenir en format String el valor dels seus atributs (el nom i identificador del doctor).

- **PATIENT**: l'objecte representa el tipus de persona pacient dins l'hospital. Un pacient vindrà definit per les característiques del seu pare i per la seva data d'admissió a l'hospital **date** (del tipus Calendar), la seva edat **age** (tipus enter) i per una llista **visits** amb les visites fetes (del tipus LinkedList<Visit>).

Pel seu comportament definirem el constructor i els mètodes necessaris per obtenir i establir els atributs: **addVisit(** Visit v **)**, **setAdmissionDate(** Calendar d **)**, **getAdmissionDate( )**, **setAge(** Integer age **)** i **getAge( )**.

Com que més tard ens interessarà ordenar els pacients de l'hospital segons la seva edat sobreescriurem el mètode **compareTo( )** de la interfície comparable (figura 2), per tant, la classe Patient implementarà Comparable<Patient> (figura 3).

```
@Override
public int compareTo( Patient p ){
    return this.age - p.age;
}
```

Figura 2: Implementació de compareTo( ) per la classe Patient

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

Figura 3: Header de la classe Patient

Per la figura 3 s'observa que la classe és definida com a abstracte. Necessitem definir-la d'aquesta manera ja que d'aquesta n'obtindrem dues classes filles (Resident i Visitor) que especialitzaràn el comportament i característiques del pacient. Llavors, per l'últim mètode de Patient, **toString( )**; ens interessarà retornar en format String la infromació del pacient segons el seu tipus (Resident o Visitor) obtenint per tant el mètode abstracte.

- **RESIDENT**: representa el pacient del tipus resident. Vindrà definit per les característiques dels seus pares (Person i Patient) i pel llit i habitació on és situat **bed** (de la classe Bed) i **room** (de la classe Room) respectivament. Pel seu comportament definirem un constructor, mètodes per assignar-li un llit i una habitació **assignBed(**Bed b**)** i **assignRoom(**Room r**)** respectivament. També un per assignar-li un doctor **getDoctor( )** (figura 4), en concret aquell que se li ha assignat a la primera visita.

```
public Doctor getDoctor(){
    // si ha estat visitat per algun doctor
    if(visits.size() > 0){
        return visits.get(0).getDoctor();
    }
    else{
        return null;
    }
}
```

Figura 4: Mètode getDoctor( ) de la classe Patient

I finalment **toString( )** per retornar en format d'String la informació del Resident.

- **VISITOR**: representa el pacient del tipus visitant. Vindrà definit per les característiques dels seus pares (Person i Patient). Pel seu comportament definirem el constructor i el mètode **toString( )** que, de nou, retornarà el valor dels atributs en forma d'String.

- **ADMINISTRATIVE**: l'objecte representa la persona dins l'hospital que exerceix el càrrec d'administrador. Vindrà definit per les característiques del seu pare (Person) i per l'hospital on treballa **hospital**. Pel seu comportament definirem el constructor i, sabent que serà l'encarregat de registrar les visites, definirem **addVisit( )** (figura5); que ens permetrà crear un registre d'una visita i assignar-la tant al doctor com al pacient de la mateixa.

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

Figura 5: Mètode addVisit( ) de la classe Administrative.

L'administratiu també s'encarregarà d'assignar un llit pel resident de l'hospital. Llavors implementarem el mètode **assignBed(** Resident r **)** (figura 6) per a que d'entre les habitacions de l'hospital l'administratiu observi si existeix cap llit lliure per llavors assignar-lo al resident desitjat. Obtindrem un return booleà cert si s'ha trobat un llit disponible i fals altrament. El mètode informarà per pantalla sobre els resultats de la cerca (on el resident ha estat assignat). Aquest mètode usarà la funció getAvaliableBed( ) de la classe Room per totes les habitacions de l'hospital, que a la vegada cridarà a isAvaliable( ) de Bed per determinar finalment si existeix cap llit a assignar al resident d'entre les habitacions de l'hospital.

Finalment sobreescrivim també el mètode **toString( )** per brindar informació en format d'String sobre els atributs nom i identificador de l'administrador.

```java
public boolean assignBed( Resident resident ){
    boolean trigger = false;
    LinkedList<Room> rooms = hospital.getRooms();

    for(int i = 0; i < rooms.size(); i++){
        Room room = rooms.get(i);

        // si tenim una habitació disponible
        if(room.isAvailable()){
            Bed avBed = room.getAvailableBed();
            resident.assignBed(avBed);
            System.out.println(this.toString() + "has assigned bed to");
            System.out.println(resident.toString() + " is assigned to");

            // si tenim un doctor assignat pel resident
            if(resident.getDoctor() != null){
                System.out.println(room.toString() + " " + avBed.toString() + " and " + resident.getDoctor().toString());
            }

            // si no tenim un doctor assignat pel resident
            if(resident.getDoctor() == null){
                System.out.println(room.toString() + " " + avBed.toString() + " and has no doctor.");
            }
            trigger = true;
            break;
        }

        // si totes les habitacions estan ocupades
        else if(!rooms.get(rooms.size()-1).isAvailable() & i == rooms.size()-1){
            System.out.println(this.toString() + "has not found bed for");
            System.out.println(resident.toString() + " and has no");
            System.out.println("room neither bed and has no doctor");
            break;
        }
    }
    return trigger;
}
```

Figura 6: Mètode assignBed( ) de la classe Administrative.

CLASSE VISIT

Cada visita serà representada per la seva data, sumari, doctor i pacient. Llavors els atributs de Visit seràn: **date** (del tipus Calendar), **summary** (del tipus String), **doctor** (de la classe Doctor) i **Patient** (de la classe Patient).

En quant als mètodes; definirem el constructor i els getters pels atributs de l'objecte. A més a més, com que més tard ens interessarà ordenar les visites de l'hospital segons la seva data sobreescriurem de nou el mètode **compareTo( )** de la interfície comparable (figura 7), per tant, la classe Visit implementarà Comparable<Patient> (figura 8).

Finalment implementarem també el mètode **toString( )** per retornar la informació dels atributs en format d'String.

```java
@Override
public int compareTo(Visit v){
    return v.date.compareTo(this.date);
}
```

Figura 7: Mètode compareTo( ) de la classe

```java
public class Visit implements Comparable<Visit>{
```

Figura 8: Header de la classe Visit.

CLASSE BED _____

Cada llit serà representat pel seu nombre identificador i el resident que l'ocupa i l'habitació que el conté. Per tant els atributs de Bed seràn: **bedID** (del tipus enter), **room** (de la classe Room) i **resident** (de la classe Resident).

En quant als mètodes; definirem el constructor, **assignRoom(** Room r **)** i **assignResident(** Resident r **)** per assignar un valor especific als atributs, el getter **getBedID( )** per obtenir l'identificador del llit, **release( )** per determinar que el llit és disponible, assignant a l'atribut de resident un valor "null" i **isAvailable( )** per determinar la disponibilitat del llit que observarà si el valor de l'atribut resident és "null" (ocupat) o no(lliure).

```java
public void release(){
    resident = null;
}
public boolean isAvailable(){
    return (this.resident == null);
}
```

Figura 9: Mètodes release( ) i isAvailable de la classe Bed.

Finalment sobreescriurem també **toString( )** per informar sobre l'identificador del llit en format d'String.

CLASSE ROOM _____

Cada habitació estarà definida pel seu nombre identificador i els llits que conté. Llavors tindrà dos atributs: **roomID** (del tipus enter) i **beds** (llista encadenada que contindrà elements de la classe Bed).

En quant als mètodes establirem el constructor per l'objecte, **addBed(** int bedID **)** per afegir un llit concretat a l'habitació, **getBed(** int idx **)** per obtenir la instància del llit desitjat

dins l'habitació, **getAvailableBed( )** (figura 10) que ens retornarà cert si existeix cap llit lliure dins l'habitació, fals altrament, aplicant el mètode isAvaliable( ) de la classe Bed per cadascun d'ells. Implementem també **listBeds( )** per obtenir la informació en format String de cada llit de l'habitació i sobreescrivim **toString( )** per obtenir l'identificador de l'habitació també en forma d'String.

```
public boolean isAvailable(){
    boolean check = true;
    for(Bed bed : beds){
        if(bed.isAvailable()){
            check = true;
            break;
        }
        else{
            check = false;
        }
    }
    return check;
}
```

Figura 10: Mètode isAvailable( ) de la classe Room.

CLASSE HOSPITAL _____

Serà l'objecte encarregat de capacitar l'hospital per manegar totes les classes descrites a dalt. Aqui implementarem els atributs i mètodes necessaris per representar qualsevol hospital.

L'hospital vindrà definit pels administratius, doctors, pacients i habitacions que conté, les visites que ha realitzat i el seu nom. Llavors tindrem els atributs **admins**, **doctors**, **patients**, **rooms, visits** (tots del tipus LinkedList amb elements de la classe adient per cada cas) i **name** (del tipus String).

Establirem pel comportament el constructor que brindarà un nom a l'hospital i inicialitzarà les llistes encadenades, **addAdmin(** Administrative a **)**, **addDoctor(** Doctor d **)**, **addRoom**(int id), **addResident**(int id, String name, int age), **addVisitor**(int id, String name, int age) i **addVisit**(Visit v) per afegir persones i visites a l'hospital (entrant valors a les llistes encadenades). **getAdmin**(int idx), **getDoctor**(int idx), **getRoom**(int idx), **getRooms( )**, **getVisit( )** i **getPatient**(int idx) per obtenir la instància de la persona o

visita indicada dins l'hospital (getRooms( ) retornarà totes les habitacions de l'hospital). També implementem **deletePatient**(int idx) per acomiadar el pacient desitjat un cop el seu tractament hagi acabat, només haurem d'eliminar la seva instància de l'atribut apuntant a la llista de pacients de l'hospital.

assignBeds(int adminIdx) ens servirà per, donada la instància d'administratiu desitjada, assignar mitjançant aquetsa (usant el seu mètode assignBed( ) (figura 6)) assignar un llit per cada pacient del tipus resident dins l'hospital

```
public void assignBeds( int adminIdx ){
    Administrative admin = admins.get(adminIdx);
    for(Patient patient : patients){
        if(patient instanceof Resident){
            admin.assignBed((Resident)patient);
            }
    }
}
```

Figura 11: Mètode assignBeds( ) de la classe

Seguint, voldrem ordenar, com hem dit abans, les visites per data i els pacients per edat. Per tant, implementarem **sortPatients( )** i **sortVisits( )** que, usant la funció sort( ) donada per la classe Collections, ordenaràn les llistes de pacients i visites tal com indiquen els mètodes compareTo( ) abans implementats per cada classe cas. Tindrem doncs:

```
public void sortPatients(){
    Collections.sort(patients);
    System.out.println("Patients sorted by age:");
    for(Patient patient : patients){
        System.out.println(patient.toString());
    }
}
public void sortVisits(){
    Collections.sort(visits);
    System.out.println("Visits sorted by date:");
    for(Visit visit : visits){
        System.out.println(visit.toString());
    }
}
```

Figura 12: Mètodes sortPatients( ) i sortVisits( ) de la classe Hospital.

Finalment sobreescriurem **toString( )** per retornar en format String els components de l'hospital, és a dir, els elements de cada llista encadenada apuntada pels atributs.

CLASSE HOSPITALMANAGER

Aquest objecte serà la classe principal del programa des d'on, mitjançant la classe Hospital, usarem les seves funcionalitats per brindar valors a qualsevol hospital i poder manegar-lo eficientment. El seu objectiu serà afegir personal, pacients, habitacions i visites a l'hospital indicat. També assignar llits, acomiadar pacients, ordenar visites i pacients per data i edat respectivament i, finalment, ser capaç d'informar sobre l'estat de l'hospital indicat.

La classe vindrà definida pels hospitals a manegar i els docotrs i administratius dels quals disposa per fer-ho. És a dir, els atributs **hospitals**, **doctors** i **administratives** (tots del tipus LinkedList amb els elements del tipus de classe adient). S'estableixen els getters i setters corresponents i finalment el main( ), des d'on provarem que tot estigui assignat correctament.

POSSIBLES SOLUCIONS ALTERNATIVES

Discutirem aqui breument algunes discusions obtingudes durant el procés de gestació del programa:

Per la funció **compareTo( )** hem acabat implementant una versió més compacte que la proposada primerament:

```
    @Override
    public int compareTo( Patient p ){
        if(this.age < p.getAge()){
            return -1;
        }
        if(this.age > p.getAge()){
            return 1;
        }
        else{
            return 0;
        }
    }

@Override
public int compareTo( Patient p ){
    return this.age - p.age;
}
```

Figura 13: A sobre, la primera versió del mètode compareTo( ) de la classe Patient, a sota, la segona versió; més elegant.

Per tal de simular la data en la classe HospitalManager tal com se'ns demana a l'enunciat hem decidit usar la classe **Calendar** ja que Date té la majoria de mètodes "deprecated".

Per alguns mètodes hem començat a implementar la versió més compacte del bucle **for** de Java:

```
for(Patient patient : patients){
    System.out.println(patient.toString());
}
```

Figura 14: Bucle for implementat seguint una sintaxi diferent a la dels anteriors programes realitzats.

CONCEPTES TEÒRICS

Esmentarem alguns aspectes de la teoria que hem observat estan relacionats amb els conceptes de programació orientada a objectes que hem aplicat com a part de la solució:

Herència: Hem palpat clarament les implicacions i beneficis d'aquest recurs. Observem com els membres d'instància de les classes pare són aplicables perfectament a les seves filles i com promouen una solució elegant pel problema. La classe Person és heretada per tots els tipus de persones dins l'hospital, especialitzant per tant la classe pare i promovent la reutilització.

Relacions entre classes: Observem les relacions entre els objectes de l'aplicació gràcies a les múltiples relacions que apareixen en el programa; per exemple, observem que un hospital conté habitacions, que a la vegada contenen llits, obtenint així una relació de composició entre aquestes. Un altre exemple és la relació d'ús per part de l'administratiu al usar una instància d'habitació i visita en el seu comportament (és així ja que l'administratiu és l'encarregat d'assignar visites i habitacions a les persones adients) .

Classes i mètodes abstractes, override: Tenim la classe Patient com a classe abstracta, això es deu a que els seus objectes fills empleen toString( ) de diferents maneres, per tant, no cal definir el mètode a la classe pare. Tenim llavors un mètode sense implementar, per tant la classe Patient es converteix en abstracte per més tard implementar el mètode abstracte a les classes filles (fent un "override" d'aquesta manera).

Interfícies: Usem la interfície Comparable per tal d'ordenar mitjançant la classe Collections les visites i els pacients com desitjem. Establirem la configuració d'ordenació sobreescrivint el mètode sort( ) definit a la interfície per cadascuna de les classes que ens interessen (Visit i Patient).

EL NOSTRE PROCEDIMENT I IMPLICACIÓ DE CADA MEMBRE

Per tal de concretar la implicació de cada membre pel laboratori, a continuació expliquem el procés de gestació del mateix:

Començem el dia de la sessió online del laboratori, on discutim conjuntament els dubtes sobre el disseny de la solució. Després decidim començar a implementar els mètodes que ens semblen més simples dels fitxers proporcionats. Un cop arribats aquí decidim deixar el projecte per reprendre'l més tard ja que en pocs dies ens trobàvem amb els examens finals del trimestre.

Un cop acabades les proves finals del trimestre ens tornem a posar, ara fixant-nos en la resta de mètodes sense implementar. Decidim progressar individualment per més tard comparar els resultats obtinguts per cadascú. Després decidim anar "testejant" si les implementacions resultaven correctes un cop posades en comú les solucions trobades individualment.

Un cop tenim el projecte avançat i més complert amb el codi ja funcional decidim observar els aspectes d'optimització i comentar-lo. Tot seguit hem procedit, finalment, a la redacció d'aquest informe. Redactant un la introducció i l'altre la explicació per cada classe.

CONCLUSSIÓ

El resultat final del programa és per nosaltres satisfactori, observem que el programa compleix satisfactòriament amb les tasques proposades i es mostra robust. Mitjançant els resultats dels tests de la classe HospitalManajer podem observar que les classes estàn ben implementades. Compleix perfectament l'objectiu de manegar hospitals.

Podem afegir hospitals, doctors amb les seves especialitats, administratius, habitacions amb llits per cadascuna; assignar administradors, doctors, residents i visitants a hospitals, crear visites i assignar llits als residents tot de manera correcta. També hem sigut capaços de simular una data i ordenar les visites i pacients de l'hospital com es demana.

El projecte ens ha permès comprovar de manera pràctica les virtuds de la programació orientada a objectes veient com els nostres objectes promouen els seus principis bàsics de la herència (promovent "reuse" al reusar la definició de la superclasse, evitant la ducplicació de mètodes). Pensem que la solució permet construir actualitzacions majors a partir de la versió actual i pot ser usada per complir tasques diferents si és desitjat.

Mostrem l'output de la solució:

```
compile:
run:
Doctor Joline ( ID 1 ) has specialities:
General
Doctor Joline ( ID 1 ) has the following visits:

Doctor Antoine ( ID 2 ) has specialities:
General
Cardiologist
Doctor Antoine ( ID 2 ) has the following visits:


Administrative Clarise ( ID 3 )has assigned bed to
Resident Jaume ( ID 87634, age 19 ) is assigned to
Room 0 Bed 0 and Doctor Joline ( ID 1 )
Administrative Clarise ( ID 3 )has assigned bed to
Resident Monica ( ID 34532, age 25 ) is assigned to
```

Room 0 Bed 1 and Doctor Antoine (ID 2)
Administrative Clarise (ID 3)has assigned bed to
Resident German (ID 62452, age 50) is assigned to
Room 1 Bed 0 and has no doctor.
Administrative Clarise (ID 3)has assigned bed to
Resident Maria (ID 21411, age 37) is assigned to
Room 1 Bed 1 and Doctor Joline (ID 1)
Administrative Clarise (ID 3)has not found bed for
Resident Francesc (ID 12999, age 88) and has no
room neither bed and has no doctor

Hospital Hospital Sant Joan de Deu
Administratives:
Administrative Clarise (ID 3)

Doctors:
Doctor Joline (ID 1)
Doctor Antoine (ID 2)

Patients:
Resident Jaume (ID 87634, age 19)
Resident Monica (ID 34532, age 25)
Resident German (ID 62452, age 50)
Resident Maria (ID 21411, age 37)
Resident Francesc (ID 12999, age 88)
Visitor Carme (ID 78678, age 63)

Rooms:
Room 0
Bed 0
Bed 1
Room 1
Bed 0
Bed 1

Visits:
29 December 2020
Doctor Joline (ID 1) Resident Jaume (ID 87634, age 19) Summary: Is a cold
30 December 2020
Doctor Antoine (ID 2) Resident Monica (ID 34532, age 25) Summary: Undefined, visit with cardiologist

Patients sorted by age:

Resident Jaume (ID 87634, age 19)

Resident Monica (ID 34532, age 25)

Resident Maria (ID 21411, age 37)

Resident German (ID 62452, age 50)

Visitor Carme (ID 78678, age 63)

Resident Francesc (ID 12999, age 88)

Visits sorted by date:

30 December 2020

Doctor Antoine (ID 2) Resident Monica (ID 34532, age 25) Summary: Undefined, visit with cardiologist

29 December 2020

Doctor Joline (ID 1) Resident Jaume (ID 87634, age 19) Summary: Is a cold

Hospital Hospital de Barcelona

Administratives:

Administrative Pere (ID 4)

Doctors:

Doctor Joline (ID 1)

Patients:

Visitor Xavi (ID 12841, age 43)

Visitor Fatima (ID 26256, age 65)

Visitor Johan (ID 62213, age 22)

Visitor Johanna (ID 26268, age 10)

Visitor Jan (ID 99887, age 90)

Rooms:

Room 0

Bed 0

Bed 1

Room 1

Bed 0

Bed 1

Visits:

31 December 2020

Doctor Joline (ID 1) Resident Maria (ID 21411, age 37) Summary: Is a cold

Patients sorted by age:
Visitor Johanna (ID 26268, age 10)
Visitor Johan (ID 62213, age 22)
Visitor Xavi (ID 12841, age 43)
Visitor Fatima (ID 26256, age 65)
Visitor Jan (ID 99887, age 90)

Visits sorted by date:
31 December 2020
Doctor Joline (ID 1) Resident Maria (ID 21411, age 37) Summary: Is a cold

BUILD SUCCESSFUL (total time: 1 second)

Figura 15: Output del programa.