Hospital

Un hospital posee un nombre, dirección y cuenta con 100 camas. Cada cama conoce su número, si está ocupada o no y en caso de estar ocupada tiene la información del paciente que la ocupa (DNI, nombre y obra social) y la cantidad de días que lleva ocupada por su último paciente.

1) Modele el problema generando las clases que considere necesarias, cada una con los constructores, estado, getters y setter que considere necesarios. **Tenga en cuenta que el hospital debe ser iniciado con todas sus camas desocupadas.**

```java
public class Hospital {
    private String nombre, direccion;
    private Cama[] camas;

    public Hospital(String n, String d){
        nombre = n;
        direccion = d;
        camas = new Cama[100];
        for(int i = 0; i < 100; i++)
            camas[i] = new Cama();
    }
}
```

```java
public class Cama {
    private Paciente paciente;
    private int dias;

    public Cama(){
        paciente = null;
        dias = 0;
    }
}
```

```java
public class Paciente extends Persona {
    private String obraSocial;

    public Paciente(String nombre, int dni, int edad, String os){
        super(nombre, dni, edad);
        obraSocial = os;
    }
}
```

2a) Ingresar un paciente (se recibe el paciente y el número de cama donde se lo va a internar)

```
public class Hospital {

    public void ingresarPaciente(Paciente p, int cama){
        camas[cama - 1].setPaciente(p);
    }
```

2a) Ingresar un paciente (se recibe el paciente el cual debe ser internado en la primer cama libre que haya)

```
public class Hospital {

    public void ingresarPaciente(Paciente p){
        int i = 0;
        while(camas[i].getOcupada())
            i++;
        camas[i].setPaciente(p);
    }
```

2b) Dar de alta un paciente liberando la cama que ocupa (se recibe el DNI/nombre del paciente que seguro existe).

```
public class Hospital {

    public void altaPaciente(int dni){
        int i = 0; boolean buscar = true;
        while(buscar){
            if(camas[i].getOcupada() && (camas[i].getPaciente().getDNI() == dni))
                buscar = false;
            else
                i++;
        }
        camas[i].setPaciente(null);
        camas[i].setDiasOcupada(0);
    }
```

```
public class Cama {

    public boolean getOcupada(){
        return paciente != null;
    }
```

2b) Dar de alta un paciente liberando la cama que ocupa (se recibe el número de cama que seguro está ocupada)

```
public class Hospital {

    public void darAltaPaciente(int cama){
        camas[cama-1].setPaciente(null);
        camas[cama-1].setDiasOcupada(0);
    }
```

2c) Incrementar en uno la cantidad de días de ocupación de todas las camas que estén ocupadas

```
public class Hospital {

    public void incrementarDias(){
        for(int i = 0; i < 100; i++){
            if(camas[i].getOcupada())
                camas[i].incrementarDias(1);
        }
    }
```

```
public class Cama {

    public void incrementarDias(int d){
        dias = dias + d;
    }
```

2c) Mover un paciente de una cama X (seguro está ocupada) a otra Y (seguro está desocupada).

```
public class Hospital {

    public void moverPaciente(int camaX, int camaY){
        camas[camaY - 1].setPaciente(camas[camaX - 1].getPaciente());
        camas[camaY - 1].setDiasOcupada(camas[camaX - 1].getDiasOcupada());
        camas[camaX - 1].setPaciente(null);
        camas[camaX - 1].setDiasOcupada(0);
    }
```

2d) Devolver la cantidad de pacientes internados

```
public class Hospital {

    public int cantidadPacientesInternados(){
        int c = 0;
        for(int i = 0; i < 100; i++){
            if(camas[i].getOcupada())
                c++;
        }
        return c;
    }
```

2d) Devolver la cantidad de camas libres

```
public class Hospital {

    public int cantidadCamasLibres(){
        int c = 0;
        for(int i = 0; i < 100; i++){
            if( ! camas[i].getOcupada())
                c++;
        }
        return c;
    }
```

2d) Devolver la cantidad de pacientes con más de ***D*** días de internación

```
public class Hospital {

    public int cantidadPacientesInternados(int D){
        int c = 0;
        for(int i = 0; i < 100; i++){
            if(camas[i].getOcupada() && camas[i].getDias() > D)
                c++;
        }
        return c;
    }
```

3) Implemente una función main que instancie un hospital, simule el ingreso de tres pacientes e invoque …

```
public static void main(String[] args) {
    Hospital h = new Hospital("NOMBRE", "DIRECCION");
    Paciente p1 = new Paciente("AA", 11, 20, "OS1");
    Paciente p2 = new Paciente("BB", 22, 40, "OS2");
    Paciente p3 = new Paciente("CC", 33, 60, "OS3");

    h.ingresarPaciente(p1, 10);
    h.ingresarPaciente(p2, 67);
    h.ingresarPaciente(p3, 32);

    h.incrementarDias();

    System.out.println(h.cantidadPacientesInternados());

    h.altaPaciente(Lector.leerInt());
}
```

Movilidad ciudadana

La ciudad de La Plata necesita un sistema para registrar datos estadísticos de movilidad ciudadana. El sistema debe registrar para cada uno de los ocho controles vehiculares y para cada uno de los cinco motivos por los cuales el ciudadano puede circular (1: trabajo; 2: salud; 3: trámite; 4: compra esencial; 5: paseo) la siguiente información estadística: cantidad de vehículos que pasan por cada control y cada motivo, cantidad de minutos que piensa estar en la calle (acumulado entre todos los vehículos que pasan) y cantidad de vehículos donde solo viaja el conductor.

1) Modele el problema generando las clases que considere necesarias, cada una con los constructores, estado, getters y setter que considere. **Tenga en cuenta que las estadísticas deben ser inicializadas con todos sus valores en cero.**

```java
public class Movilidad {

    private Estadistica[][] estadisticas;

    public Movilidad(){
        estadisticas = new Estadistica[8][5];
        for(int i = 0; i < 8; i++)
            for(int j = 0; j < 5; j++)
                estadisticas[i][j] = new Estadistica();
    }
```

```java
public class Estadistica {
    private int vehiculos, vehiculosSoloConductor, minutos;

    public Estadistica(){
        vehiculos = 0;
        vehiculosSoloConductor = 0;
        minutos = 0;
    }
}
```

2a) Registrar los datos de un vehículo (viaja solo o acompañado, minutos y motivo) que pasa por el control C

```
public class Movilidad {

    public void registrarAuto(int control, int motivo, int minutos, boolean viajaSolo)
        estadisticas[control - 1][motivo - 1].registrarAuto(minutos, viajaSolo);
    }
```

```
public class Estadistica {

    public void registrarAuto(int mins, boolean viajaSolo){
        vehiculos++;
        if(viajaSolo)
            vehiculosSoloConductor++;
        minutos = minutos + mins;
    }
```

2b) Devolver el promedio de minutos acumulados entre todos los controles y motivos

```java
public class Movilidad {

    public int controlMasConcurrido(){
        int maxControl = 0, maxVehiculos = -1, v;

        for(int i = 0; i < 8; i++){
            v = 0;
            for(int j = 0; j < 5; j++){
                v = v + estadisticas[i][j].getVehiculos();
            }
            if(v > maxVehiculos){
                maxVehiculos = v;
                maxControl = i;
            }
        }

        return maxControl + 1;
    }
```

2b) Devolver la cantidad de conductores que viajan acompañados entre todos los controles y motivos

```
public class Movilidad {

    public int conductoresAcompañados(){
        int c = 0;
        for(int i = 0; i < 8; i++)
            for(int j = 0; j < 5; j++)
                c = c + estadisticas[i][j].getVehiculosAcompañados();
        return c;
    }
```

```
public class Estadistica {

    public int getVehiculosAcompañados(){
        return vehiculos - vehiculosSoloConductor;
    }
```

2b) Devuelva un string con motivo y número de control donde pasaron menos conductores solos.

```
public class Movilidad {

    public String motivoYControlConMenosVehiculos(){
        int motivo = 0, control = 0;

        for(int i = 0; i < 8; i++)
            for(int j = 0; j < 5; j++)
                if(estadisticas[i][j].getVehiculosSolos() <
                        estadisticas[motivo][control].getVehiculosSolos()){
                    motivo = i;
                    control = j;
                }

        return "Motivo " + motivo + " y control " + control;
    }
```

2c) Número de control por el cual pasaron más vehículos (sin importar el motivo)

```
public class Movilidad {

    public int controlMasConcurrido(){
        int maxControl = 0, maxVehiculos = -1, v;

        for(int i = 0; i < 8; i++){
            v = 0;
            for(int j = 0; j < 5; j++){
                v = v + estadisticas[i][j].getVehiculos();
            }
            if(v > maxVehiculos){
                maxVehiculos = v;
                maxControl = i;
            }
        }

        return maxControl + 1;
    }
```

2c) Devolver el motivo por el cual se moviliza más cantidad de vehículos/minutos (entre todos los controles).

```
public class Movilidad {

    public int motivoMasUsado(){
        int maxMotivo = 0, maxVehiculos = -1, v;

        for(int j = 0; j < 5; j++){
            v = 0;
            for(int i = 0; i < 8; i++){
                v = v + estadisticas[i][j].getVehiculos();
            }
            if(v > maxVehiculos){
                maxVehiculos = v;
                maxMotivo = j;
            }
        }

        return maxMotivo + 1;
    }
```

3) Implemente una función main que instancie el sistema de control de movilidad ciudadana y simule el registro del paso de 10 vehículos…

```
public static void main(String[] args) {
    Movilidad m = new Movilidad();

    for(int i = 0; i < 100; i++){
        int con = GeneradorAleatorio.generarInt(8)+1;
        int mot = GeneradorAleatorio.generarInt(5)+1;
        int mins = GeneradorAleatorio.generarInt(120)+1;
        boolean solo = GeneradorAleatorio.generarBoolean();
        m.registrarAuto(con, mot, mins, solo);
    }

    System.out.println(m.promedioMinutos());

    System.out.println(m.controlMasConcurrido());
}
```

Grupos alfa y beta

Un Laboratorio realiza experimentos para evaluar la eficacia de un nuevo fármaco para la Diabetes y para ello conformó dos grupos de pacientes: Grupo Alfa y Grupo Beta.

Ambos grupos registran la información de a lo sumo 10 pacientes De cada paciente se guarda: un ID (1..10), nombre, último resultado de glucosa (double) y última dosis recibida de fármaco (double). Sin embargo, los grupos difieren en la forma de aplicar el fármaco a los pacientes (esto se detalla más adelante)

1) Modele el problema generando las clases que considere necesarias, cada una con los constructores, estado, getters y setter que considere. **Tenga en cuenta que los grupos inicialmente no tienen pacientes**

```java
*/
public abstract class Grupo {
    private Paciente[] pacientes;
    private int dimL, dimF = 10;

    public Grupo(){
        pacientes = new Paciente[dimF];
        dimL = 0;
    }
}
```

```java
public class GrupoAlfa extends Grupo {

}
```

```java
*/
public class GrupoBeta extends Grupo {

}
```

```java
public class Paciente extends Persona {

    private double glucosa, ultimaDosis;

    public Paciente(String nombre, int dni, int edad, double g, double ud){
        super(nombre, dni, edad);
        glucosa = g;
        ultimaDosis = ud;
    }
}
```

2a) Agregar un paciente ***P*** al grupo, en caso de existir espacio

```
public abstract class Grupo {

    public void agregarPaciente(Paciente p){
        if(dimL < dimF){
            pacientes[dimL] = p;
            dimL++;
        }
    }
```

2b) Obtener un paciente dado un ID

```
public abstract class Grupo {

    public Paciente obtenerPaciente(int id){
        return pacientes[id - 1];
    }
```

```
public abstract class Grupo {

    public Paciente obtenerPacienteBuscando(int id){
        int i = 0;
        while(pacientes[i].getId() != id)
            i++;
        return pacientes[i];
    }
```

2c) Aplicar una dosis a un paciente (se recibe una dosis D (double) y se debe modificar su última dosis recibida a D y disminuir la glucosa en un valor aleatorio entre 0 y 1.

```
public class Paciente extends Persona {

    public void aplicarDosis(double D){
        ultimaDosis = D;
        glucosa = glucosa - GeneradorAleatorio.generarDouble(1);
    }
```

2d) Aplicar una dosis D de fármaco a los pacientes del grupo, teniendo en cuenta que:

i) en el Grupo Alfa se le aplica la dosis D a todos los pacientes

ii) en el Grupo Beta se le aplica la dosis D a los pacientes cuya glucosa supera el valor 2.5

```
public abstract class Grupo {

    public int cantidadPacientes(){
        return dimL;
    }

    public abstract void aplicarDosis(double D);
```

```
public class GrupoAlfa extends Grupo {
    public void aplicarDosis(double D){
        for(int i = 1; i <= this.cantidadPacientes(); i++){
            Paciente p = this.obtenerPaciente(i);
            p.aplicarDosis(D);
        }
    }
}
```

```
public class GrupoBeta extends Grupo {
    public void aplicarDosis(double D){
        for(int i = 1; i <= this.cantidadPacientes(); i++){
            Paciente p = this.obtenerPaciente(i);
            if(p.getGlucosa() > 2.5)
                p.aplicarDosis(D);
        }
    }
```

2e) Obtener la representación string del grupo, la cual se compone por el ID, nombre, última glucosa y última dosis de todos los pacientes del grupo

```
public abstract class Grupo {

    public String toString(){
        String s = "";
        for(int i = 0; i < dimL; i++)
            s = s + "ID " + (i+1) + ": " + pacientes[i].getNombre() +
                    pacientes[i].getGlucosa() + pacientes[i].getUltimaDosis();
        return s;
    }
```

3) Realice un main que instancie un Grupo Alfa y un Grupo Beta. Llene cada grupo con pacientes (el primero con 3 y el segundo con 4). Aplique una dosis D de fármaco (leída por teclado) a los pacientes de cada grupo. Luego imprima la representación string de cada grupo.

```
public static void main(String[] args) {
    Grupo alfa = new GrupoAlfa(), beta = new GrupoBeta();

    alfa.agregarPaciente(new Paciente("AA", 11, 20, 2.0, 3.0));
    alfa.agregarPaciente(new Paciente("BB", 22, 30, 2.1, 3.1));
    alfa.agregarPaciente(new Paciente("CC", 33, 40, 2.2, 3.2));

    beta.agregarPaciente(new Paciente("DD", 44, 25, 4.0, 6.0));
    beta.agregarPaciente(new Paciente("EE", 55, 35, 4.1, 6.1));
    beta.agregarPaciente(new Paciente("FF", 66, 45, 4.2, 6.2));
    beta.agregarPaciente(new Paciente("GG", 77, 55, 5.2, 7.2));

    double dosis = Lector.leerDouble();
    alfa.aplicarDosis(dosis);
    beta.aplicarDosis(dosis);

    System.out.println(alfa.toString());
    System.out.println(beta.toString());
```