

Trabajo Práctico #2 - Memoria Escrita

Programación Orientada a Objetos

Grupo 2- Equipo #2

Por:

Maria Paula Ardila Otero

Daniela Cristina Garcia Torres

Leopold Pierre Lanard

Jose Nicolas Duque Linares

Ronal Yesid Castro Romero

Profesor: Jaime Alberto Guzman Luna



Universidad Nacional de Colombia

**Sede Medellín
2023-2**

Tabla de Contenidos

I. Descripción general de la solución	2
Introducción	2
Estructura	2
1. Capa de persistencia (baseDatos)	3
2. Capa lógica (gestorAplicacion)	3
3. Capa asociada a la interfaz del usuario (uiMain)	4
II. Descripción del diseño estático del sistema en la especificación UML	4
III. Descripción de la implementación de características de programación orientada a objetos del sistema	5
Clase abstracta	5
Método abstracto	5
Interfaz	6
Herencia	6
Ligadura dinámica	7
Se presentan dos casos de ligadura dinámica asociado al modelo lógico:	7
Atributo de clase	8
Método de clase	8
Constante	10
Encapsulamiento	10
Sobrecarga de constructores	11
Sobrecarga de métodos	11
Manejo de referencias this para desambiguar y this()	12
Enumeración	13
IV. Descripción de las 5 funcionalidades implementadas.	14
Funcionalidad 1: Agendar citas	14
Funcionalidad 2: Generar fórmulas médicas	17
Funcionalidad 3: Asignar Habitaciones	21
Funcionalidad 4: Aplicación de vacunas	25
Funcionalidad 5: Facturación	29
V. Manual de usuario	32
Datos precargados en la aplicación	32
¿Cómo funciona la aplicación?	33
1. Funcionalidad de agendar citas:	34
2. Funcionalidad de generar fórmulas médicas:	35
3. Funcionalidad de asignar habitación:	37
4. Funcionalidad de vacunas:	40
5. Facturación	41

I. Descripción general de la solución

Introducción

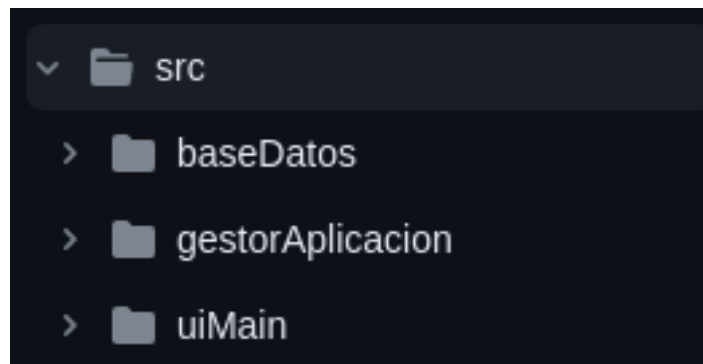
La idea de este proyecto se basa en crear un sistema de información hospitalaria donde generalmente una sola persona será la encargada de manipular las funcionalidades encontradas en este (preferiblemente algún personal de secretaría).

El primer análisis al que nos lleva este proyecto es pensar en todos los elementos involucrados alrededor del funcionamiento de un hospital. Decidimos centrarnos en aspectos muy comunes relacionados al área de la salud y su manejo, tales como, citas médicas, vacunas, medicamentos, reserva de habitaciones, y pagos de servicios.

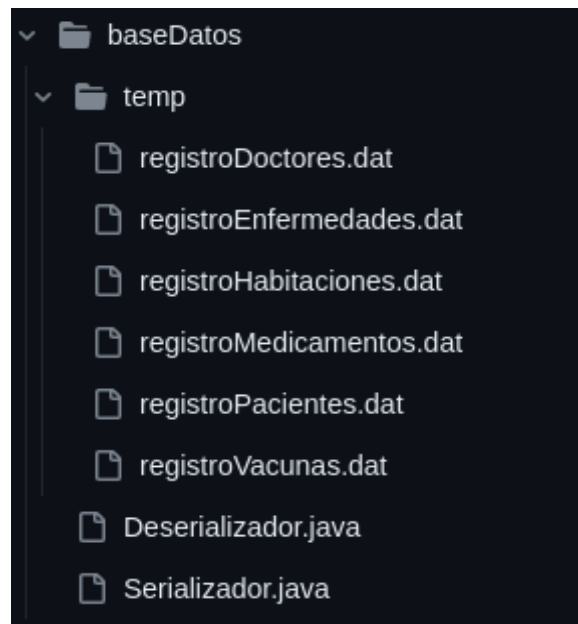
El modelo anteriormente descrito, se ajusta muy bien al modelo de programación imperativa propiamente de la programación orientada a objetos, pues es adecuado abordar la implementación del programa por medio de clases y objetos, elementos, que por la naturaleza del proyecto van a ser muy descriptivos, reduciendo el nivel de abstracción, haciendo mucho más entendible el código y su funcionamiento.

Estructura

Para la construcción de la solución, surgen tres necesidades fundamentales que se deben discutir, organizadas de esta manera:



1. Capa de persistencia (baseDatos)



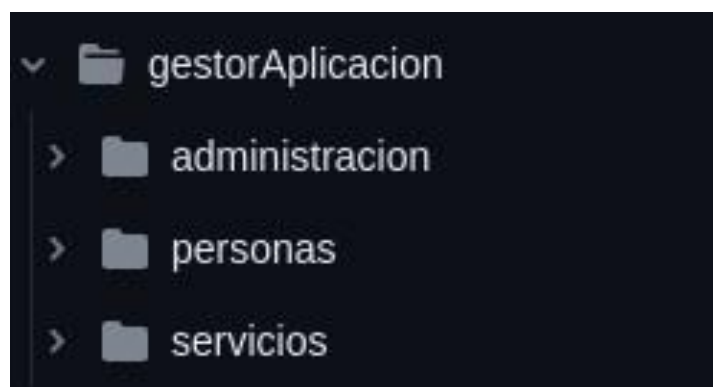
Con esta capa se implementa la persistencia en los datos del programa, los cuales deben ser estables en el tiempo, permitiendo que las funcionalidades aumenten su usabilidad y no estén limitadas al tiempo de ejecución.

Es por esto, que en primer lugar, se hace en la estructura del proyecto, una capa de persistencia, encargada de satisfacer esta primera necesidad asociada con los datos y la creación de objetos dentro de ella.

Con el archivo llamado “Serializador” al momento de salir de la aplicación se van a serializar todos los objetos creados durante la ejecución que están asociados al hospital. Estos valores se van a guardar en archivos de texto plano, para su posterior uso en la deserialización.

Con el archivo “Deserializador” se van a leer estos valores guardados en las respectivas rutas, y se van a montar en la memoria al momento de la ejecución, haciendo posible usar estos objetos creados en ejecuciones anteriores.

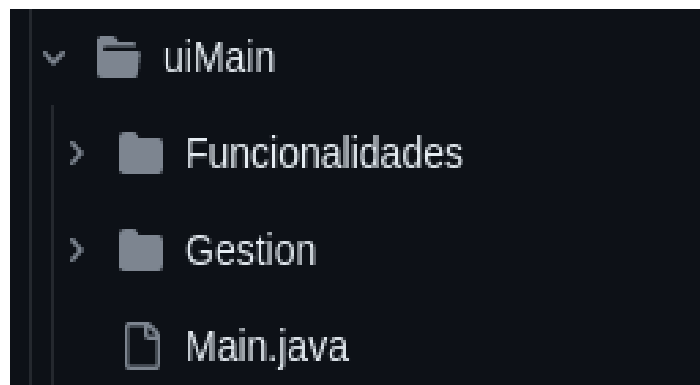
2. Capa lógica (gestorAplicacion)



En el paquete de gestorAplicacion tenemos:

- El paquete administracion donde están las clases:
 - El enumerado CategoriaHabitacion
 - Factura
 - HistoriaClinica
 - Hospital
 - Medicamento
 - Vacuna
 - La interfaz Pago
- El paquete personas donde están las clases:
 - Doctor
 - Enfermedad
 - Paciente
 - Persona
- El paquete servicios donde están las clases:
 - Cita
 - CitaVacuna
 - Formula
 - Habitacion
 - La clase abstracta Servicio

3. Capa asociada a la interfaz del usuario (uiMain)



En esta capa es donde se implementa la ejecución del programa y donde encontramos todos los menús para interactuar con los distintos objetos. Tenemos el paquete *Gestión* que es donde se crea, ven y eliminan las instancias de las clases *Paciente*, *Vacuna*, *Doctores* y *Hospital*; el otro paquete que encontramos acá es *Funcionalidades* donde están las 5 funcionalidades del programa y también está el menú principal que inicia el programa.

II. Descripción del diseño estático del sistema en la especificación UML

https://drive.google.com/file/d/1j46CTJk6Swfv2yVL7YvKxg8BebK4JBU/view?usp=drive_link

III. Descripción de la implementación de características de programación orientada a objetos del sistema

Clase abstracta

```
1 > /.../
3 package gestorAplicacion.servicios;
4
5 import gestorAplicacion.administracion.HistoriaClinica;
6 import gestorAplicacion.personas.Paciente;
7
8 import java.io.Serializable;
9 import java.util.ArrayList;
10
11 //Clase Abstracta destinada a herencia de servicios del hospital
11 usages 4 inheritors 1 Santiago Arboleda +1
12 public abstract class Servicio implements Serializable {
13
14     //Atributos
14     1 usage
15     protected static long generadorID = 1000;
15     6 usages
16     protected final long idServicio;
17     protected Paciente paciente;
17     3 usages
18     private boolean estadoPago;
19
20     //Constructor
20     3 usages 1 Santiago Arboleda
21     public Servicio(Paciente paciente) {
22         this.idServicio = generadorID++;
23         this.paciente = paciente;
24         this.estadoPago = false;
```

En el paquete “*gestorAplicacion.servicios*” se encuentra la clase abstracta “**Servicio**”. Hay 4 clases que heredan de esta clase abstracta, “Cita”, “Formula”, “Habitacion”, “CitaVacuna”.

El objetivo de la clase “Servicio” es poder crear otras clases asociadas a las funcionalidades del hospital que directamente representan un servicio. Pues un servicio es un concepto intangible y muy general, no es adecuado instanciar objetos de esta clase, por esta razón se decide reducir esta clase a abstracta.

Método abstracto

```
/* Método abstracto que debe implementarse en las clases hijas para
validar el pago
*/
1 usage 4 implementations 1 Santiago Arboleda
public abstract void validarPago(Paciente paciente, long idServicio);

/* Metodo abstracto para imprimir la descripcion del servicio.
Se usa este metodo porque el toString se utiliza en otra funcionalidad.
*/
1 usage 4 implementations 1 Santiago Arboleda
public abstract String descripcionServicio();
```

En la clase abstracta “**Servicio**” se usa un método abstracto llamado “validarPago” con dos parámetros, un paciente y el id del servicio. Posteriormente se implementa en las clases “Cita”, “Formula”, “Habitacion”, “CitaVacuna”. El objetivo de definir este método abstracto es obligar a las clases anteriormente mencionadas a que lo implementen, sin embargo, en cada clase el cuerpo del

método es diferente. En líneas generales, con este método se valida el pago de un servicio y se cambia su estado de pago, recalando que en cada clase hija de “Servicio” el método tiene una lógica distinta.

En esta misma clase tenemos otro método abstracto “descripcionServicio” que retorna un string con la descripción del servicio. Este método se implementó porque el método “toString” que usualmente se usa para esto se está usando en otra funcionalidad.

Interfaz

```
1  /* Autores: Diego Andres Gracia Granados, Daniel Giraldo Vanegas, Elian David Velandia Riveros, Juan Camilo Gutierrez
2  y Santiago Arboleda Acevedo */
3  package gestorAplicacion.administracion;
4
5  > import ...
6
7
8  //Interfaz destinada a pagos
9  1 usage 1 implementation 1 Santiago Arboleda +1
11 public interface Pago {
12     //Constante
13     6 usages
14     double IVA = 0.19;
15
16     //Métodos
17     1 usage 1 implementation 1 Santiago Arboleda
18     double calcularPrecio(Cita cita);
19     1 usage 1 implementation 1 Santiago Arboleda
20     double calcularPrecio(CitaVacuna citaVacuna);
21     1 usage 1 implementation 1 Santiago Arboleda
22     double calcularPrecio(Formula formula);
23     1 usage 1 implementation 1 Santiago Arboleda
24     double calcularPrecio(Habitacion habitacion);
25 }
26
```

En el paquete “*gestorAplicacion.administracion*” se encuentra la interfaz “Pago”, la cual tiene una constante llamada IVA, y define 4 métodos que al compilarse son abstractos y tienen diferentes parámetros, permitiendo también una sobrecarga en el método “calcularPrecio”.

Esta interfaz es implementada en la clase “Paciente” que se encuentra en el paquete “*gestorAplicacion.personas*”, obligando a codificar los 4 métodos llamados “calcularPrecio”, además dentro del cuerpo de cada uno de estos métodos se hace uso de la constante IVA brindada por la interfaz.

Herencia

En este proyecto, se presentan 6 relaciones de herencia, organizadas de la siguiente forma:

- En el paquete “*gestorAplicacion.personas*” las clases “**Paciente**” y “**Doctor**” heredan de la clase “Persona”
- En el paquete “*gestorAplicacion.servicios*” las clases “**Cita**”, “**Formula**” y “**Habitación**” heredan de la clase abstracta “Servicio”, adicionalmente, la clase “**CitaVacuna**” hereda de la clase “Cita”

```

public class Cita extends Servicio {
    //Atributos
    3 usages
    private Doctor doctor;
    3 usages
    private String fecha;
}

```

```

public class CitaVacuna extends Cita{
    //Atributos
    3 usages
    private Vacuna vacuna;
}

```

Ligadura dinámica

Se presentan dos casos de ligadura dinámica asociado al modelo lógico:

El primer caso, se presenta en la clase “Paciente” que se encuentra en el paquete “*gestorAplicacion.personas*”, en el método “despedida” con un parámetro. Lo primero que se debe recalcar es que un objeto CitaVacuna también es tipo Cita (pues la clase CitaVacuna extiende de Cita), por lo tanto como parámetro del método puede ingresar una cita de vacuna como también una cita médica. La clase “Cita” conoce el método “mensaje”, y “CitaVacuna” también tiene este mismo método definido de una manera diferente. Por lo cual, la ambigüedad de estos métodos, se resuelve por ligadura dinámica en tiempo de ejecución. Es decir, si al método ingresa una cita de vacuna va a ejecutar el método “mensaje” asociado a la clase “CitaVacuna”, en cambio, si ingresa una cita común, se irá por el método “mensaje” de la clase “Cita”.

```

2 usages Juan Camilo Gutiérrez Martínez
public String despedida(Cita citaAsignada ){
    /*Hay ligadura dinámica ya que la referencia
    citaAsignada puede apuntar a una cita o a una cita vacuna
    y dependiendo de eso ejecutará el método más específico
    de mensaje().
    */
    return ";Adiós "+getNombre()+" "+citaAsignada.mensaje();
}

```

Esta ligadura dinámica hace posible que en la funcionalidad de vacunas y en la de citas médicas se muestre un mensaje de despedida diferente al usuario.

El segundo caso, también se encuentra en la clase “Paciente” en el método “mensajeDoctor” que tiene un parámetro. Un doctor juega también a ser tipo persona, por la relación de herencia que tienen estas dos clases. Por lo tanto, un doctor puede pasar como parámetro en este método. Consecuentemente, hay una ambigüedad, pues la clase persona tiene el método “bienvenida” y el doctor también tiene el método bienvenida(). Este problema se resuelve en el momento de ejecución por ligadura dinámica.

```

//Método para bienvenida de doctor
1 usage Juan Camilo Gutiérrez Martínez
public String mensajeDoctor(Persona doctor){
    //Hay ligadura dinámica
    return doctor.bienvenida()+ "\nPor favor selecciona los medicamentos que vas a formularle a: "+getNombre();
}

```


Esto permite tener una relación polimórfica en el momento en el que el doctor le formula los medicamentos al paciente en la funcionalidad de fórmula médica y de esta manera tener un mensaje de bienvenida completamente distinto de acuerdo al tipo de usuario.

Adicionalmente se encuentra una ligadura dinámica asociada a la interfaz:

```
// Calcular precio
double precioServicioSeleccionado = 0;
if (servicioSeleccionado instanceof Formula)
    precioServicioSeleccionado = pacienteSeleccionado.calcularPrecio((Formula) servicioSeleccionado);
else if (servicioSeleccionado instanceof CitaVacuna)
    precioServicioSeleccionado = pacienteSeleccionado.calcularPrecio((CitaVacuna) servicioSeleccionado);
else if (servicioSeleccionado instanceof Habitacion)
    precioServicioSeleccionado = pacienteSeleccionado.calcularPrecio((Habitacion) servicioSeleccionado);
else if (servicioSeleccionado instanceof Cita)
    precioServicioSeleccionado = pacienteSeleccionado.calcularPrecio((Cita) servicioSeleccionado);
```

Antes de que se ejecute esta parte del código, la variable “servicioSeleccionado” se inicializa con tipo “Servicio” y posteriormente se le asigna como valor un objeto tipo “Formula”, “Cita”, “CitaVacuna” o “Habitacion”.

La ligadura dinámica se produce al ejecutar el método “calcularPrecio”, ya que el tipo real del objeto determina el método que se va a ejecutar.

Atributo de clase

En la clase “Hospital”, ubicada en el paquete “gestorAplicacion.administracion”, se ha implementado el atributo estático “habitaciones” como un “ArrayList<Habitacion>”. Este atributo se utiliza para almacenar una lista de objetos “Habitacion” y se emplea en la funcionalidad “AsignarHabitacion”. Se utiliza porque se proporciona una forma de acceder y manipular las habitaciones del hospital de manera centralizada y compartida entre todas las instancias de la clase.

```
39 usages
18     public static ArrayList<Habitacion> habitaciones= new ArrayList<>();
19
```

Método de clase

En la clase “Habitacion”, ubicada en el paquete “gestorAplicacion.servicios”, se implementa el método estático “BuscarHabitacionesDisponibles” que retorna un ArrayList de objetos de tipo “Habitacion” (ArrayList<Habitacion>). El método acepta un parámetro de entrada de tipo “CategoriaHabitacion”; este método se utiliza para buscar habitaciones disponibles en función de una categoría específica la cual es la categoría que desea buscar y devuelve una lista de habitaciones que se ajusten a esa categoría y que estén disponibles (no ocupadas).

Es estático porque proporciona una forma conveniente de buscar y obtener una lista de habitaciones disponibles de una categoría específica sin necesidad de crear objetos de la clase “Hospital”.

```

11 usages  Daimon-22 *
public static ArrayList<Habitacion> BuscarHabitacionDisponible(CategoriaHabitacion categoria)
{
    //se instancia una nueva variable tipo ArrayList de clase habitacion
    ArrayList<Habitacion>habitacionesDisponibles=new ArrayList<>();
    //se recorre la lista estatica que se encuentra en hospital
    for (Habitacion habitacion : Hospital.habitaciones)
    {
        //se comprueba si la habitacion esta no ocupada y si la categoria que se ingreso es la misma que la
        //del objeto habitacion de la lista
        if (!habitacion.isOcupada() && habitacion.getCategoria() == categoria)
        {
            // se agrega el objeto habitacion que cumpla y se retorna
            habitacionesDisponibles.add(habitacion);
        }
    }

    //se comprueba si la lista esta vacia y si lo esta retorna null
    if (habitacionesDisponibles.isEmpty()) {
        return null;
    }
    //retorna la nueva lista
    return habitacionesDisponibles;
}

```

En la clase “Servicio”, ubicada en el paquete “gestorAplicacion.servicios”. Aca implementamos el metodo de clase “obtenerServiciosSinPagar”, que recibe como parámetro un paciente. Se utiliza para obtener una lista de servicios pendientes de pago para un paciente específico. Recopila los servicios de la HistoriaClinica del paciente y filtra aquellos que ya han sido pagados.

```

public static ArrayList<Servicio> obtenerServiciosSinPagar(Paciente paciente) {
    HistoriaClinica historiaClinicaPaciente = paciente.getHistoriaClinica();
    ArrayList<Servicio> serviciosSinPagar = new ArrayList<>();

    // Obtiene todos los servicios brindados al paciente
    serviciosSinPagar.addAll(historiaClinicaPaciente.getHistorialCitas());
    serviciosSinPagar.addAll(historiaClinicaPaciente.getListaFormulas());
    if (paciente.getHabitacionAsignada() != null)
        serviciosSinPagar.add(paciente.getHabitacionAsignada());
    serviciosSinPagar.addAll(historiaClinicaPaciente.getHistorialVacunas());

    // Filtra los servicios pagados
    serviciosSinPagar.removeIf(Servicio::isEstadoPago);

    return serviciosSinPagar;
}

```

Constante

Además de la constante usada en la interfaz, la aplicación también hace uso de una constante en la clase “Paciente” al momento de definir el atributo “historiaClinica”, esto se hace con el objetivo que un

paciente no tenga un cambio de historial clínica, y por ejemplo, no se le asigne alguna de otra persona.

Alternativamente, la clase “HistoriaClinica” tiene una constante al momento de definir el atributo “paciente”. Ya que una historia clínica debe tener un paciente fijo, no cambia a través del tiempo.

```
public class Paciente extends Persona implements Pago {  
  
    // Atributos  
    4 usages  
    private final HistoriaClinica historiaClinica;  
}
```

```
public class HistoriaClinica implements Serializable {  
  
    //Atributos  
    2 usages  
    private final Paciente paciente;  
}
```

Otra constante utilizada se encuentra en la clase “Servicio”, ubicada en el paquete “gestorAplicacion.servicios”. Aca definimos la constante “idServicio” que sirve como un identificador único para cada servicio.

```
public abstract class Servicio implements Serializable {  
  
    //Atributos  
    1 usage  
    protected static long generadorID = 1;  
    6 usages  
    protected final long idServicio;  
    protected Paciente paciente;  
    3 usages  
    private boolean estadoPago;  
}
```

Encapsulamiento

Generalmente, los métodos y constructores usan el encapsulamiento public, en cambio, se trató que la mayoría de los atributos tuvieran un encapsulamiento private, a excepción de algunos que tienen protected o public.

```
public class Cita extends Servicio {  
  
    //Atributos  
    4 usages  
    private Doctor doctor;  
    5 usages  
    protected String fecha;  
}
```

```
@Override  
public void validarPago(Paciente paciente, long idServicio) {  
    for (Cita cita :  
        paciente.getHistoriaClinica().getHistorialCitas()) {  
        if (cita.getIdServicio() == idServicio)  
            cita.setEstadoPago(true);  
    }  
}
```

Sobrecarga de constructores

En la clase "**Paciente**" ubicada en el paquete "*gestorAplicacion.personas*", se utiliza la sobrecarga de constructores para ofrecer dos opciones: una que permite crear un objeto Paciente con todos los atributos requeridos, incluyendo la categoría de habitación, y otra que simplifica la creación omitiendo la categoría de habitación. En la clase "**Formula**" ubicada en el paquete "*gestorAplicacion.servicios*", también se utiliza la sobrecarga para permitir la creación de objetos con diferentes combinaciones de parámetros, proporcionando una forma más sencilla cuando no se especifican ciertos valores. La sobrecarga de constructores ofrece flexibilidad al adaptarse a diferentes necesidades y situaciones.

```
16      no usages
17      public Formula(ArrayList<Medicamento> listaMedicamentos, Doctor doctor, Paciente paciente) {
18          super(paciente);
19          this.listaMedicamentos = listaMedicamentos;
20          this.doctor = doctor;
21      }
22      no usages
23      public Formula(Paciente paciente){
24          this( listaMedicamentos: null, doctor: null, paciente);
25      }
26
27      // Constructores
28      no usages
29      public Paciente(int cedula, String nombre, String tipoEps, CategoriaHabitacion categoriaHabitacion) {
30          super(cedula,nombre,tipoEps);
31          this.categoriaHabitacion=categoriaHabitacion;
32          this.historiaClinica = new HistoriaClinica(this);
33      }
34      // Sobrecarga de constructor
35      no usages
36      public Paciente(int cedula, String nombre, String tipoEps){
37          this(cedula, nombre, tipoEps, categoriaHabitacion: null);
38      }
39  }
```

Sobrecarga de métodos

La sobrecarga de métodos se utilizó en la clase "**Paciente**" que implementa la interfaz "**Pago**" ubicado en el paquete "*gestorAplicacion.personas*", se utiliza para calcular precios en diferentes situaciones específicas relacionadas con el sistema de gestión del hospital. Cada método de cálculo de precio está diseñado para recibir diferentes tipos de objetos ("**Formula**", "**Cita**", "**CitaVacuna**" y "**Habitacion**") como parámetros y realizar cálculos adaptados a esas situaciones, como lo son el tipo de EPS, categoría de habitación, tipo de cita, tipo de vacuna y la cantidad de medicamentos.

```
1 usage  👤 dangv31 +1
@Override
public double calcularPrecio(Formula formula){...}

1 usage  👤 DiegoGG512
@Override
public double calcularPrecio(Cita citaAsignada) {...}

1 usage  👤 Juan Camilo Gutiérrez Martínez +1
@Override
public double calcularPrecio(CitaVacuna citaAsignada){...}

1 usage  👤 Daimon-22 +2
@Override
public double calcularPrecio(Habitacion habitacionAsignada)
{...}
```

Manejo de referencias this para desambiguar y this()

Las referencias this para desambiguar se usan en casi todos los constructores presentes en el proyecto, se muestra como ejemplo los constructores de la clase "Enfermedad" y la clase "Habitación".

```
public Habitacion(int numero, CategoriaHabitacion categoria, boolean ocupada, Paciente paciente, int dias)
{
    super(paciente);
    this.numero = numero;
    this.categoria = categoria;
    this.ocupada = ocupada;
    this.dias=dias;
}
```

```
public Enfermedad(String especialidad, String nombre, String tipologia) {
    this.especialidad = especialidad;
    this.nombre = nombre;
    this.tipologia = tipologia;
    enfermedadesRegistradas.add(this);
}
```

La referencia this() lo podemos observar en la sobrecarga de constructores de "Formula" y en "Paciente"

```
// Constructores
Daimon-22 +2
public Paciente(int cedula, String nombre, String tipoEps, CategoriaHabitacion categoriaHabitacion) {
    super(cedula,nombre,tipoEps);
    this.categoriaHabitacion=categoriaHabitacion;
    this.historiaClinica = new HistoriaClinica( paciente: this);
}

// Sobrecarga de constructor
dangv31
public Paciente(int cedula, String nombre, String tipoEps){
    this(cedula, nombre, tipoEps, categoriaHabitacion: null);
}
```

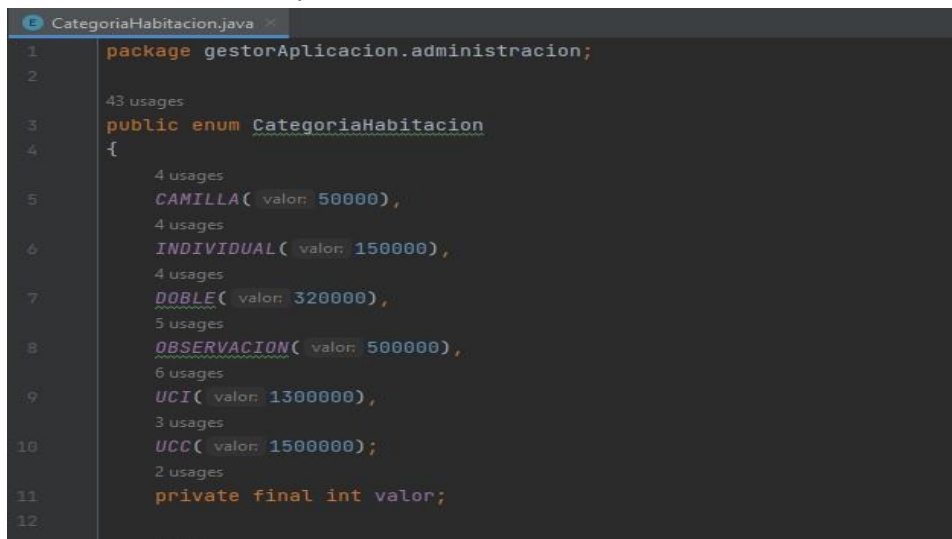
```
// Constructor
1 usage  dangv31
public Formula(ArrayList<Medicamento> listaMedicamentos, Doctor doctor, Paciente paciente) {
    super(paciente);
    this.listaMedicamentos = listaMedicamentos;
    this.doctor = doctor;
}

// Sobrecarga de constructor y uso del this()
1 usage  dangv31
public Formula(Paciente paciente){
    this( listaMedicamentos: null, doctor: null, paciente);
}
```

Enumeración

Se crea una clase enum llamada “**CategoriaHabitacion**” en el paquete llamado “*gestorAplicacion.administracion*” que se utiliza para representar las diferentes categorías de habitaciones en un hospital. Cada categoría tiene un valor asociado utilizando la sintaxis “<NOMBRE>(<VALOR>)”. En esta clase, se define un constructor privado que asigna el valor correspondiente a cada categoría, y este valor no es modificado en la funcionalidad. Además, se incluye un método “**getValor()**” para obtener el valor asociado a cada categoría.

Esta clase enum se utiliza en la funcionalidad de “**AsignarHabitacion**”, donde cada habitación se asigna a una categoría específica junto con su valor correspondiente. En el método “**calcularPrecio**”, se utiliza esta información para determinar el costo a pagar según la EPS del paciente. De esta manera, se establece un precio diferente para cada categoría de habitación y se adapta a las necesidades de cada paciente.



```
1 package gestorAplicacion.administracion;
2
3 public enum CategoriaHabitacion
4 {
5     CAMILLA( valor: 50000),
6     INDIVIDUAL( valor: 150000),
7     DOBLE( valor: 320000),
8     OBSERVACION( valor: 500000),
9     UCI( valor: 1300000),
10    UCC( valor: 1500000);
11    private final int valor;
12 }
```

IV. Descripción de las 5 funcionalidades implementadas.

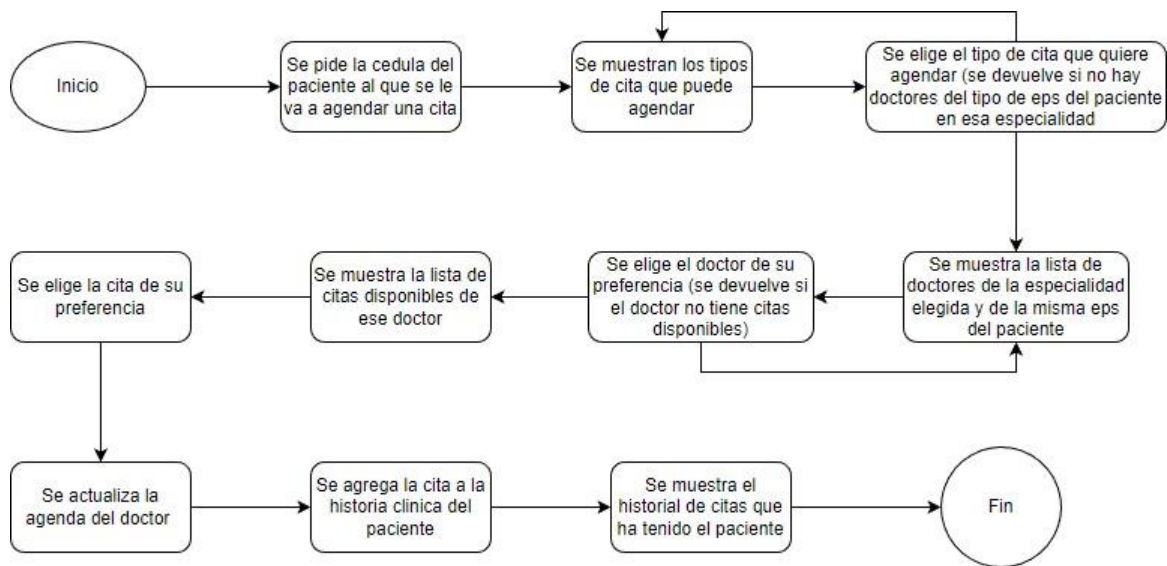
Funcionalidad 1: Agendar citas

El cuerpo de la funcionalidad se encuentra en el paquete “*iuMain.funcionalidades*” en la clase “AgendarCitas”.

Esta funcionalidad se encarga de realizar el agendamiento de una cita médica en la especialidad que se requiera, con la libertad de elegir el doctor y la fecha entre las que estén disponibles. Durante su proceso de ejecución, intervienen varias clases:

- **Hospital:** De esta se sacarán las listas de pacientes y doctores para filtrarlas
- **Paciente:** En esta se filtrarán los doctores de acuerdo con el tipo de eps del paciente
- **Doctor:** De esta se sacarán las citas que tenga disponibles para que el paciente escoja
- **Cita:** Son los objetos que estarán en la agenda del doctor, y a los cuales se les asignará un paciente
- **HistoriaClinica:** en esta se agregaran las citas que haya agendado el paciente, también se podrá ver el historial de citas

Secuencia de la funcionalidad:



Ejecución de la funcionalidad:

```

MENU FUNCIONALIDADES
1. Agendar una cita medica
2. Generar fórmula médica
3. Asignar habitación a un paciente
4. Aplicarse una vacuna
5. Facturacion
6. Regresar al menu inicial
7. Salir
Ingrese una opcion:
1
Ingrese su numero de cedula: 444
Bienvenido Usuario Camilo

Seleccione el tipo de cita que requiere
1. General
2. Odontologia
3. Oftalmologia
4. --Regresar al menu--
Ingrese una opcion: 1

Lista de doctores Generales para el tipo de eps Subsidiado:
1. Diego
2. Andres
3. Felipe

Seleccione el doctor con el que quiere la cita: 1
  
```

```
Citas disponibles:
1. 3 de Abril, 8:00 am
2. 4 de Abril, 3:00 pm
3. 5 de Abril, 10:00 am

Seleccione la cita de su preferencia: 1

Su cita ha sido asignada con exito

Resumen de su cita:
Fecha: 3 de Abril, 8:00 am
Paciente: Camilo
Doctor: Diego

historial citas de historia clinica del paciente:

Fecha: 5 de Abril, 10:00 am
Paciente: Camilo
Doctor: Andres

Fecha: 3 de Abril, 8:00 am
Paciente: Camilo
Doctor: Diego

¡Adiós Camilo del servicio cita médica!
```

Explicación:

En esta funcionalidad podemos observar tres interacciones complejas con clases y métodos:

- Cuando se pide el número de cédula del paciente, se interactúa con la clase “**Hospital**” y su método llamado “**buscarPaciente**”. Aquí se busca el paciente por su número de cédula y lo retorna. Si no hay ninguna coincidencia se retornará un null.

```
public Paciente buscarPaciente(int cedulaPaciente) {
    for (Paciente paciente :
        listaPacientes) {
        if (paciente.getCedula() == cedulaPaciente) {
            return paciente;
        }
    }
    return null;
}
```


- La siguiente es cuando se le pide al paciente que escoja el tipo de cita que requiere, luego con el método “**buscarDoctorEps**” de la clase “**Paciente**” se filtra la lista de doctores según el tipo de eps y la especialidad que se haya escogido, y la retorna para luego imprimir la lista de doctores disponibles.

```
public ArrayList<Doctor> buscarDoctorEps(String especialidad, Hospital hospital){
    ArrayList<Doctor> doctoresPorEspecialidad = hospital.buscarTipoDoctor(especialidad);
    ArrayList<Doctor> doctoresDisponibles = new ArrayList<>();

    for(int i=1; i<=doctoresPorEspecialidad.size(); i++) {
        if (Objects.equals(doctoresPorEspecialidad.get(i-1).getTipoEps(), getTipoEps())) {
            doctoresDisponibles.add(doctoresPorEspecialidad.get(i-1));
        }
    }

    return doctoresDisponibles;
}
```

- Después de escoger el doctor de la lista anterior, se ejecutará el método “**mostrarAgendaDisponible**” de la clase “**Doctor**” el cual filtra la agenda del doctor, la cual es un array de citas, para retornar solamente las citas en las que no se haya asignado ningún paciente, es decir, las citas disponibles, para luego mostrarlas en una lista y que se pueda escoger una de ellas.

```
public ArrayList<Cita> mostrarAgendaDisponible() {
    ArrayList<Cita> agendaDisponible = new ArrayList<>();
    for(int i=1; i<=agenda.size(); i++) {
        if (agenda.get(i-1).getPaciente() == null) {
            agendaDisponible.add(agenda.get(i-1));
        }
    }

    return agendaDisponible;
}
```

Funcionalidad 2: Generar fórmulas médicas

El cuerpo de la funcionalidad se encuentra en el paquete “*iuMain.funcionalidades*” en la clase “*FormulaMedica*”.

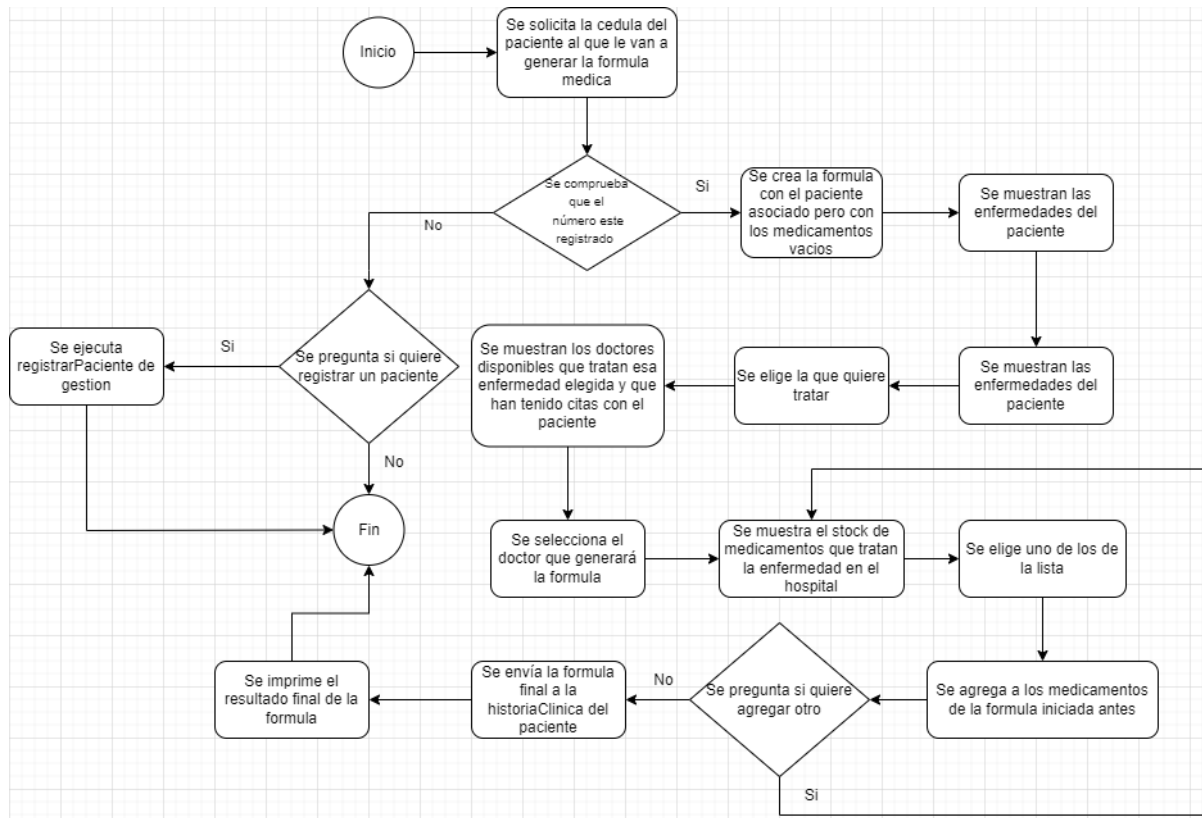
Esta funcionalidad se encarga de la creación de instancias de la clase “*Formula*”, pero para ello ocurren distintos procesos para que se genere con respecto a las necesidades del paciente. Para poder generar la fórmula médica el paciente tiene que tener enfermedades registradas y citas agendadas desde antes, ya que es necesario que un doctor que lo haya atendido sea el que seleccione los medicamentos a formular, posterior a esto se mostrará una lista de medicamentos que tengan stock en el hospital y que a su vez traten la enfermedad deseada.

Durante su ejecución intervienen:

- Hospital:** Donde buscaremos en la *listaPacientes* el paciente asociado por medio del método *buscarPaciente*

- **HistoriaClinica:** Es donde se encuentran las enfermedades del paciente y donde está el método “*buscarCitaDoc*” que encontrará los doctores que tratan la enfermedad seleccionada en el historial de citas de la historia clínica.
- **Paciente:** Donde se ubica el metodo “*medEnfermedad*” que buscará los medicamentos disponibles que traten la enfermedad elegida
- **Medicamento:** Estos son los objetos que estamos agregando a la lista de medicamentos de la fórmula médica asociada al paciente
- **Formula:** Se crea una instancia de este, la cual se guarda en la historia clínica del paciente
- **Cita:** De aquí encontraremos los doctores que atienden la enfermedad del paciente y que alguna vez hayan tenido una cita con él
- **Enfermedad:** Con las instancias de esta clase se filtraran los doctores y medicamentos.

Secuencia de la funcionalidad:



Ejecución de la funcionalidad:

```
¡Adiós Juan del servicio cita médica!

MENU FUNCIONALIDADES
1. Agendar una cita medica
2. Generar fórmula médica
3. Asignar habitación a un paciente
4. Aplicarse una vacuna
5. Facturacion
6. --Regresar al menu inicial--
7. --Salir--
Ingrese una opcion:
2
Ingrese la cédula del paciente:
111
¿Que enfermedad deseas tratar?
1.Caries I Especialidad que la trata: Odontologia
2.Bronquitis II Especialidad que la trata: General
2
Los doctores que lo han atendido y están disponibles para formular Bronquitis II son:
1.David
1
Hola doctor, David
Por favor selecciona los medicamentos que vas a formularle a: Juan
1.Nombre: Oseltamivir para Bronquitis II, Enfermedad: Bronquitis II, Descripcion: Cura la enfermedad, Cantidad: 10, Precio: 5000.0
1
Esta es tu lista actual de medicamentos:[Nombre: Oseltamivir para Bronquitis II, Enfermedad: Bronquitis II, Descripcion: Cura la enfermedad]
1.Nombre: Oseltamivir para Bronquitis II, Enfermedad: Bronquitis II, Descripcion: Cura la enfermedad
¿Desea agregar otro medicamento? (s/n)
s

Hola doctor, David
Por favor selecciona los medicamentos que vas a formularle a: Juan
1.Nombre: Oseltamivir para Bronquitis II, Enfermedad: Bronquitis II, Descripcion: Cura la enfermedad, Cantidad: 9, Precio: 5000.0
1
Esta es tu lista actual de medicamentos:[Nombre: Oseltamivir para Bronquitis II, Enfermedad: Bronquitis II, Descripcion: Cura la enfermedad, Nombre: Oseltamivir para Bronquitis II, Enfermedad: Bronquitis II, Descripcion: Cura la enfermedad]
1.Nombre: Oseltamivir para Bronquitis II, Enfermedad: Bronquitis II, Descripcion: Cura la enfermedad
2.Nombre: Oseltamivir para Bronquitis II, Enfermedad: Bronquitis II, Descripcion: Cura la enfermedad
¿Desea agregar otro medicamento? (s/n)
n
Hola Juan
Estos son tus medicamentos formulados:
[Nombre: Oseltamivir para Bronquitis II, Enfermedad: Bronquitis II, Descripcion: Cura la enfermedad, Nombre: Oseltamivir para Bronquitis II, Enfermedad: Bronquitis II, Descripcion: Cura la enfermedad]
```

Explicación:

- La funcionalidad inicia solicitando al usuario el número de cédula del paciente, este número lo usaremos como parámetro en **“buscarPaciente”** ubicado en la clase **“Hospital”** y en el contexto de nuestra aplicación en la instancia **“hospital”**, este método nos retorna una instancia de tipo **“Paciente”**, al retornar el paciente se inicia una lista de instancias de tipo **“Medicamento”** para ir guardandolos allí y se crea una nueva instancia de la clase **“Formula”** para que la lista de medicamentos se ingrese a la fórmula

```
no usages
public Paciente buscarPaciente(int cedulaPaciente) {
    for (Paciente paciente :
        listaPacientes) {
        if (paciente.getCedula() == cedulaPaciente) {
            return paciente;
        }
    }
    return null;
}
```

- Después de retornar el paciente se imprime las enfermedades de la historia clínica del paciente las cuales son instancias de la clase **“Enfermedad”** al imprimirlas el paciente elige cual desea tratar, de enfermedad elegida se usa el atributo **especialidad** contenida en ella

como parámetro en el método “**buscarCitaDoc**” dentro de “**HistoriaClinica**”, para que la funcionalidad continúe se deben tener citas con algún doctor con la misma **especialidad** con la que se trata la **enfermedad**, ya agendadas en el **historialCitas** y este método nos retornará un **ArrayList** con instancias de la clase “**Doctor**” que sean de la especialidad de la enfermedad y que hayan atendido alguna vez al paciente.

```
/* Método que busca los doctores que estén disponibles y también los doctores con los que el paciente
asociado a esta historia haya tenido alguna cita */
1 usage  Juan Camilo Gutiérrez Martínez +2
public ArrayList<Doctor> buscarCitaDoc(String especialidad, Hospital hospital) {
    ArrayList<Doctor> doctoresDisp = hospital.buscarTipoDoctor(especialidad);
    ArrayList<Doctor> docCita = new ArrayList<>();
    for (Doctor doc : doctoresDisp){
        for (Cita cita : historialCitas){
            if (doc.getCedula() == cita.getDoctor().getCedula()){
                docCita.add(doc);
            }
        }
    }
    return docCita;
}
```

- Al finalizar el método anterior se imprime la lista para elegir qué doctor formulará los medicamentos, este doctor se asocia a la instancia de la clase “**Formula**” creada al inicio. Después de esto se inicia el **do-while** para ir agregando los medicamentos, dentro de este ciclo se inicia el método “**medEnfermedad**” ubicado en “**Paciente**” que recibe como parámetro también la enfermedad elegida anteriormente, para que la funcionalidad siga y funcione se necesita tener en la **listaMedicamentos** de la instancia **hospital** medicamentos que tengan como atributo la **enfermedad** a tratar, este método retorna un **ArrayList** con instancias de la clase “**Medicamento**” y filtra los medicamentos según la enfermedad.

```
/* Método que se encarga de buscar los medicamentos disponibles y posteriormente busca los medicamentos
que traten la enfermedad ingresada como parámetro */
1 usage  Juan Camilo Gutiérrez Martínez +1
public ArrayList<Medicamento> medEnfermedad(Enfermedad enfermedad, Hospital hospital) {
    ArrayList<Medicamento> medicamentos = hospital.medicamentosDisponibles();
    ArrayList<Medicamento> medEnfermedades = new ArrayList<>();
    for (Medicamento med : medicamentos){
        if (enfermedad.getNombre().equals(med.getEnfermedad().getNombre()) && enfermedad.getTipologia().equals(med.getEnfermedad().getTipologia())){
            medEnfermedades.add(med);
        }
    }
    return medEnfermedades;
}
```

- Cuando elija un medicamento para agregar, se añade a la lista de instancias de **Medicamento** iniciada al principio y se actualiza la lista de medicamentos de la instancia de **Formula** creada al inicio, después se pregunta al usuario si desea agregar otro medicamento, si lo desea se ejecuta de nuevo el bucle, sino se finaliza y se agrega a la historia clínica del paciente y se imprime el **toString** de **Formula** asociada a la instancia creada al inicio.

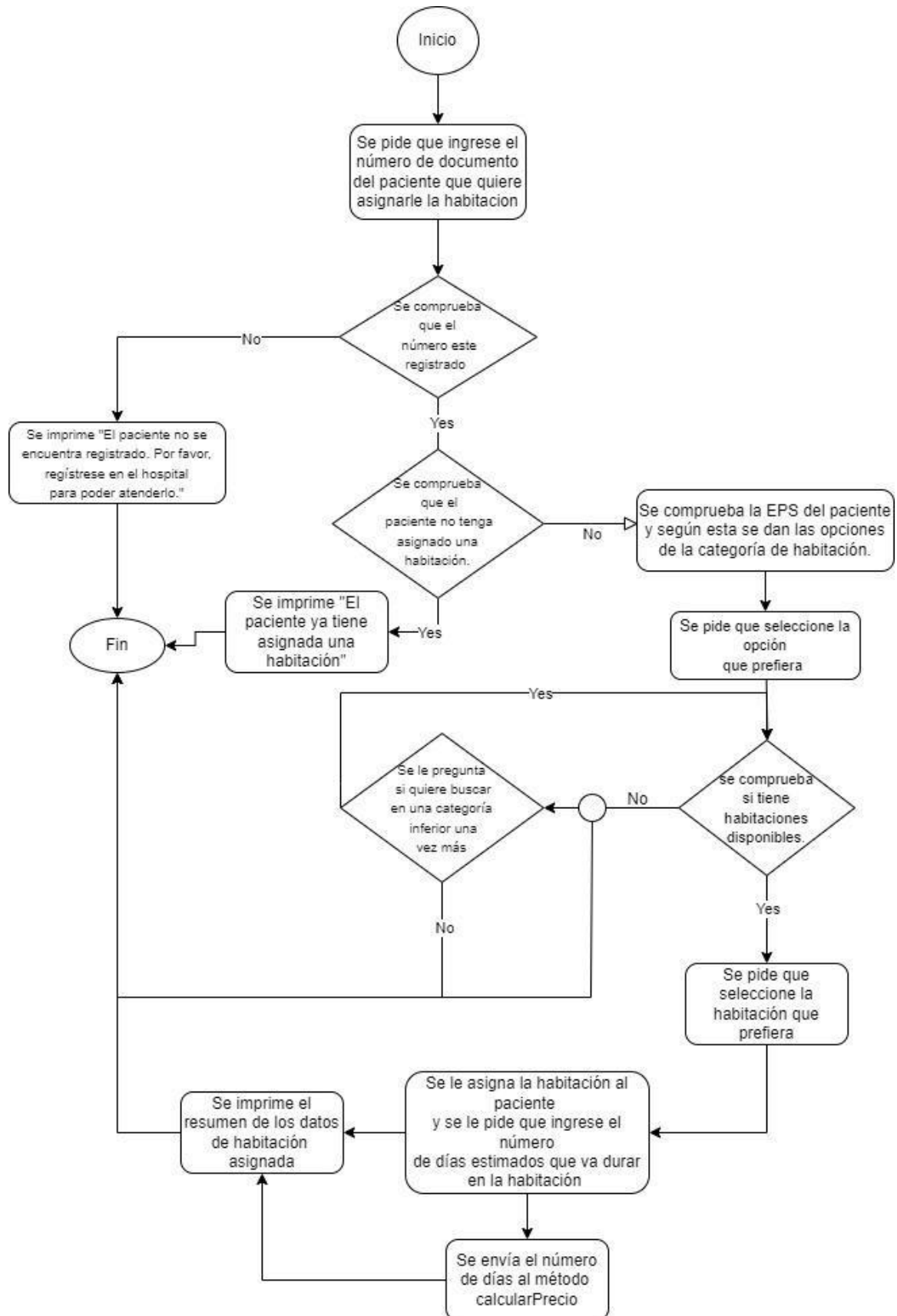
Funcionalidad 3: Asignar Habitaciones

El cuerpo de la funcionalidad se encuentra en el paquete *“iuMain.funcionalidades”* en la clase *“AsignarHabitacion”*.

Esta funcionalidad se encarga de que los pacientes registrados tengan la posibilidad de seleccionar entre categorías específicas de habitaciones, según su afiliación a la EPS. Cada categoría ofrece una variedad y selección única de habitaciones, permitiendo a los pacientes elegir aquella que se ajuste mejor a sus preferencias individuales. Además, cada categoría tiene un valor asociado, asegurando que se aplique una tarifa acorde a la cobertura proporcionada por la EPS, las clases que intervienen en la funcionalidad son las siguientes:

- **Hospital:** Se encarga de almacenar las listas implicadas en la funcionalidad y además contiene el método *buscaPaciente* que es primordial para interactuar con los objetos tipo paciente..
- **Paciente:** Representa a los pacientes registrados en el hospital y está diseñada para permitirles seleccionar categorías específicas de habitaciones según su afiliación a la EPS. Además, la clase tiene métodos para calcular el precio del servicio de habitaciones.
- **Habitación:** Representa una habitación en un hospital y proporciona funcionalidades relacionadas con la gestión de las habitaciones, como buscar habitaciones disponibles, cambiar categorías y validar pagos.
- **CategoriaHabitacion:** Define objetos que representan las diferentes categorías de habitaciones disponibles en un hospital y proporciona un atributo privado (valor) numérico asociado a cada categoría. Esto permite realizar cálculos o tomar decisiones basadas en el costo de cada tipo de habitación.

Secuencia de la funcionalidad:



Ejecución de la funcionalidad:

```
MENU FUNCIONALIDADES
1. Agendar una cita medica
2. Generar fórmula médica
3. Asignar habitación a un paciente
4. Aplicarse una vacuna
5. Facturacion
6. --Regresar al menu inicial--
7. --Salir--
Ingrese una opcion:
3
Ingrese el número de identificación del paciente:
111
Elija el tipo de habitacion que desee, recuerde que segun el tipo va el costo del servicio
1. CAMILLA
2. OBSERVACION
3. UCI
Ingrese una opción: 1
Habitaciones disponibles de la categoría CAMILLA:
0. Número ID: 1
1. Número ID: 2
2. Número ID: 3
Seleccione la habitación deseada (ingrese un número del 0 al 2):
0
Ingrese el número estimado de días para la estadia en la habitacion:
10
```

```
Habitaciones disponibles de la categoría CAMILLA:
0. Número ID: 1
1. Número ID: 2
2. Número ID: 3
Seleccione la habitación deseada (ingrese un número del 0 al 2):
0
Ingrese el número estimado de días para la estadia en la habitacion:
10

Datos de la habitacion del Paciente:
Número de ID de la habitación: 1
Categoría de la habitación: CAMILLA
Cedula del Paciente: 111
Nombre del Paciente: Juan
```

Explicación:

La funcionalidad inicia solicitando al usuario que ingrese el número de cédula del paciente a quien le quiere asignar la habitación, se comunica con la clase "Hospital" más específicamente al método "buscarPaciente" al cual se le pasa como parametro el numero ingresado por el usuario, este método nos retorna un objeto tipo "Paciente".

```

no usages
public Paciente buscarPaciente(int cedulaPaciente) {
    for (Paciente paciente :
        listaPacientes) {
        if (paciente.getCedula() == cedulaPaciente) {
            return paciente;
        }
    }
    return null;
}

```

Después de retornar el paciente se comprueba que tipo de EPS es el paciente dependiendo del tipo se le despliegan opciones de un switch con las categorías disponibles dichas categorías se obtiene de la clase enum llamada “CategoriaHabitacion”, el usuario escoge la categoría que prefiera y necesite, se llama al método estático llamado “BuscarHabitacionDisponible” la cual nos busca las habitaciones que están vacías y son de la misma categoría y nos retorna un ArrayList de tipo habitacion (ArrayList<Habitacion>), se recorre la lista que nos retorno y nuevamente con switch nos muestra las opciones de habitaciones disponibles y se selecciona la que prefiera.

```

30 //Se crea el metodo estatico, el cual se encarga de filtrar y retornar un ArrayList de las habitaciones vacias de la categoria que se selecciono
31 //usages
32 @ public static ArrayList<Habitacion> BuscarHabitacionDisponible(CategoriaHabitacion categoria)
33 {
34     //se instancia una nueva variable tipo ArrayList de clase habitacion
35     ArrayList<Habitacion> habitacionesDisponibles=new ArrayList<>();
36     //se recorre la lista estatica que se encuentra en hospital
37     for (Habitacion habitacion : Hospital.habitaciones)
38     {
39         //se comprueba si la habitacion esta no ocupada y si la categoria que se ingreso es la misma que la del objeto habitacion de la lista
40         if (!habitacion.isOcupada() && habitacion.getCategoria() == categoria)
41         {
42             // se agrega el objeto habitacion que cumple y se retorna
43             habitacionesDisponibles.add(habitacion);
44         }
45     }
46     //se comprueba si la lista esta vacia y si lo esta retorna null
47     if (habitacionesDisponibles.isEmpty()) {
48         return null;
49     }
50     //retorna la nueva lista
51     return habitacionesDisponibles;
52 }
53
54

```

A Continuación se cambia la habitación para que aparezca ocupada y se apunta el paciente a la habitación correspondiente, se busca de la lista inicial la habitación que se modificó y se reemplaza con la habitación modificada, ahora se le pide al usuario que ingrese los días estimados y se reemplaza el atributo de días en la habitación asignada y se comunica con clase “Paciente”, con la habitación asignada al paciente se pasa como parámetro en el método “calcularPrecio, ya después se imprime los datos de la asignación de la habitación.


```

@Override
public double calcularPrecio(Habitacion habitacionAsignada)
{
    double precio=0;
    if (Objects.equals(getTipoEps(), b: "Subsidiado"))
    {
        precio=habitacionAsignada.getCategoria().getValor()*0;
        return precio*(1+IVA);
    }
    else if (Objects.equals(getTipoEps(), b: "Contributivo"))
    {
        precio=(habitacionAsignada.getCategoria().getValor()/2)*habitacionAsignada.getDias();
        return precio*(1+IVA);
    }
    else
    {
        precio=habitacionAsignada.getCategoria().getValor()*habitacionAsignada.getDias();
        return precio*(1+IVA);
    }
}

```

Funcionalidad 4: Aplicación de vacunas

El cuerpo de la funcionalidad se encuentra en el paquete *“iuMain.funcionalidades”* en la clase *“Vacunacion”*.

Esta funcionalidad da la posibilidad de aplicar una vacuna a un paciente, para eso se da la libertad que elija el tipo de vacuna que requiere (Obligatoria, No obligatoria), sin embargo, cada vacuna tiene una disponibilidad de acuerdo al tipo de EPS. Una misma vacuna puede tener existencia en varias EPS (Subsidiado, Contributivo, Particular).

Por esta razón es que se realiza un filtro, como cada paciente tiene un tipo de EPS, al momento de seleccionar (Obligatoria, No obligatoria), el programa buscará esas vacunas que cumplen la condición anterior y que tiene existencia en la eps específica del paciente.

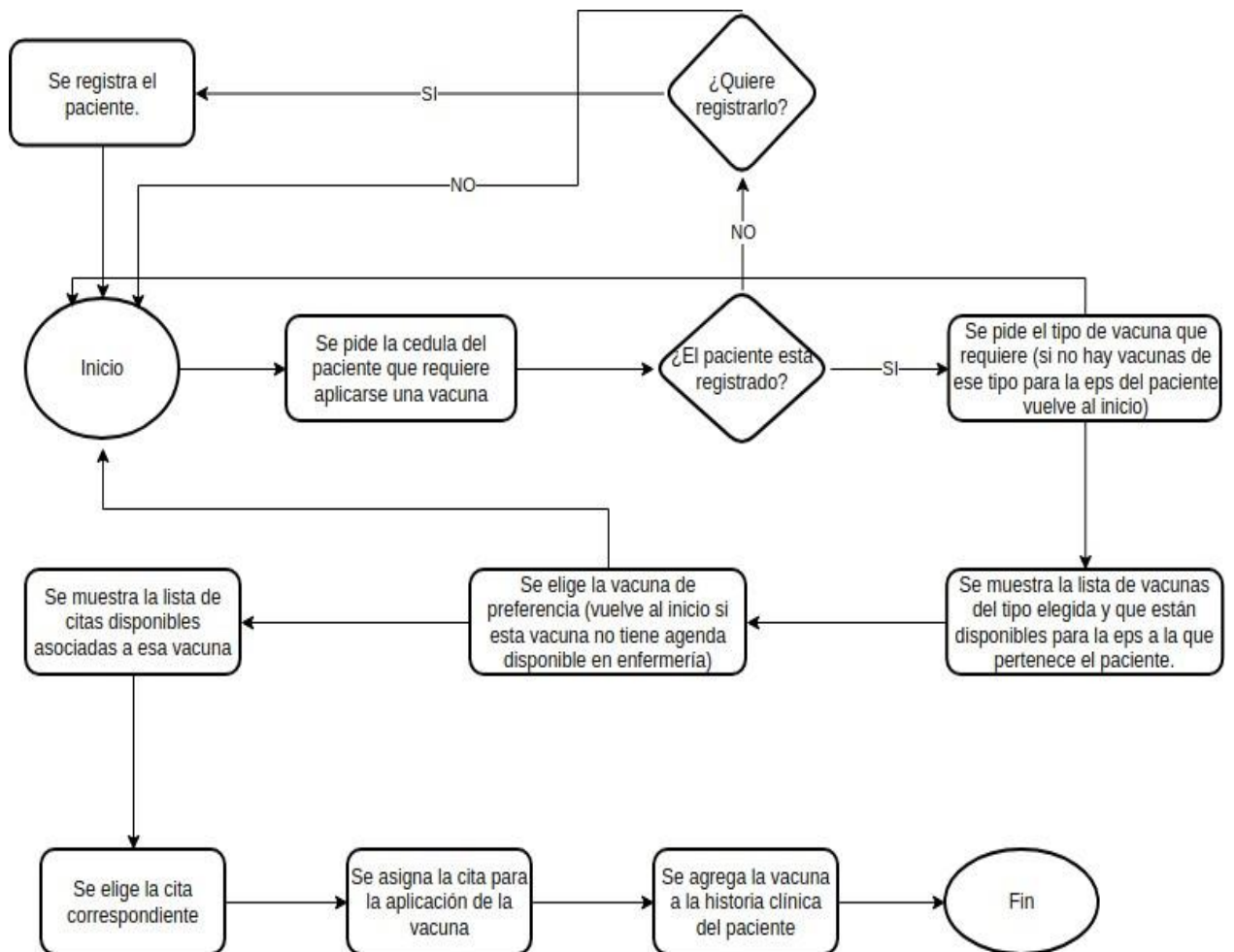
Cuando se elige la vacuna, se muestra la agenda disponible que hay en enfermería para la aplicación de esa vacuna, mostrando solamente las citas que están disponibles, es decir, que no tienen un paciente asignado.

Cuando el usuario selecciona la cita de su preferencia, esta cita ya es de él, y la vacuna elegida se relaciona automáticamente con su historia clínica. Esto con el fin, que en futuros usos de la funcionalidad, no pueda aplicarse una misma vacuna en reiteradas ocasiones.

Durante la ejecución de esta funcionalidad intervienen las siguientes clases:

- **Hospital:** de esta se sacarán las listas de pacientes y vacunas para filtrarlas.
- **Paciente:** en esta se filtrarán las vacunas de acuerdo con el tipo de eps del paciente.
- **Vacuna:** de esta se sacará la agenda disponible para el usuario.
- **CitaVacuna:** hace referencia a cada una de las citas que están asociadas a vacunación.
- **HistoriaClínica:** en esta se agregaran las vacunas que se ha aplicado el paciente.

Secuencia de la funcionalidad:



Ejecución de la funcionalidad:

```
MENU FUNCIONALIDADES
1. Agendar una cita medica
2. Generar fórmula médica
3. Asignar habitación a un paciente
4. Aplicarse una vacuna
5. Facturacion
6. --Regresar al menu inicial--
7. --Salir--
Ingrese una opcion:
4
Ingrese la cédula del paciente:
111
Bienvenido Usuario Juan

Seleccione el tipo de vacuna que requiere
1. Obligatoria
2. No obligatoria
Ingrese una opcion:
1
Vacunas Disponibles
1.Hepatitis
2.Rotavirus
```

```
Seleccione la vacuna que requiere aplicarse:
1

Citas disponibles:
1. 3 de Abril, 8:00 am
2. 3 de Abril, 11:00 am
3. 4 de Abril, 3:00 pm
4. 5 de Abril, 10:00 am

Seleccione la cita de su preferencia:
1

Cita asignada correctamente, puede acudir al centro asistencial con la siguiente informacion:

Resumen de su cita:
Fecha: 3 de Abril, 8:00 am
Paciente: Juan
Vacuna: Hepatitis
Asistente médico: Enfermera

Este es el historial de vacunas aplicadas del paciente seleccionado:
1. Vacuna: Hepatitis

¡Adiós Juan del servicio de vacunas!
```

Explicación:

En esta funcionalidad podemos observar tres interacciones complejas con clases y métodos:

- Cuando se pide el número de cédula del paciente, se interactúa con la clase “**Hospital**” y su método llamado “**buscarPaciente**”. Aquí se busca el paciente por su número de cédula y lo retorna para su manipulación. Si no hay ninguna coincidencia se retornará un null.

```
public Paciente buscarPaciente(int cedulaPaciente) {
    for (Paciente paciente :
        listaPacientes) {
        if (paciente.getCedula() == cedulaPaciente) {
            return paciente;
        }
    }
    return null;
}
```

- Cuando el paciente selecciona el tipo de vacuna que quiere (Obligatoria, No obligatoria) se interactúa con la clase “**Paciente**” con el método “**buscarVacunaPorEps**”, el cuál se encarga de filtrar las vacunas por su tipo, y adicionalmente selecciona aquellas que tienen disponibilidad en la eps específica del paciente.

```
public ArrayList<Vacuna> buscarVacunaPorEps(String tipo, Hospital hospital){
    ArrayList<Vacuna> vacunasPorTipo= hospital.buscarTipoVacuna(tipo);
    ArrayList<Vacuna> vacunasDisponibles= new ArrayList<>();

    for (int i=1; i<=vacunasPorTipo.size();i++){
        ArrayList<String> tipoEpsVacuna = vacunasPorTipo.get(i-1).getTipoEps();
        for (int j=1;j<=tipoEpsVacuna.size();j++){
            if (Objects.equals(tipoEpsVacuna.get(j - 1), getTipoEps())){
                vacunasDisponibles.add(vacunasPorTipo.get(i-1));
            }
        }
    }
    return vacunasDisponibles;
}
```

- En el momento de la selección de la cita, se puede evidenciar la última interacción compleja, que corresponde con la clase “**Vacuna**” y su método “**mostrarAgendaDisponible**”, el cuál busca esas citas de la vacuna que no tienen un paciente asignado y las retorna.

```

public ArrayList<CitaVacuna> mostrarAgendaDisponible(){
    ArrayList<CitaVacuna> agendaDisponible= new ArrayList<>();
    for (int i=1; i<=agenda.size();i++){
        if (agenda.get(i-1).getPaciente()==null){
            agendaDisponible.add(agenda.get(i-1));
        }
    }
    return agendaDisponible;
}

```

Posterior a esto, se le asigna el paciente a esta cita de vacuna, y se agrega la vacuna a su historia clínica.

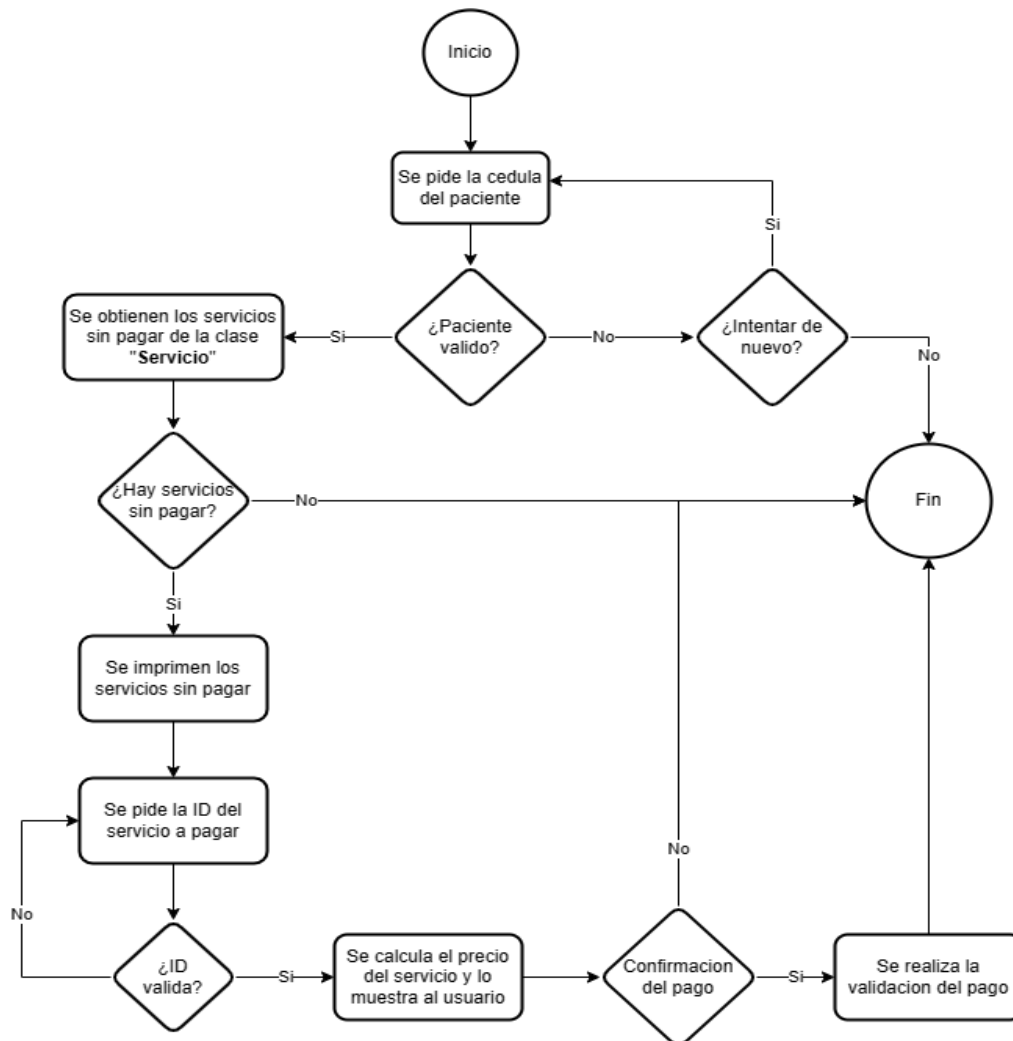
Funcionalidad 5: Facturación

Esta funcionalidad realiza la facturación de servicios médicos para un paciente, buscando al paciente, mostrando los servicios pendientes de pago, permitiendo al usuario seleccionar y pagar un servicio específico, y registrando el pago en el sistema.

Las clases que intervienen en esta funcionalidad:

- **Hospital:** En esta clase se encuentran la lista de pacientes y el método para buscar pacientes.
- **Paciente:** En esta clase se encuentra la historia clínica y la implementación de los métodos para calcular el precio del servicio.
- **Pagos:** Interfaz donde se dan los métodos para calcular el precio del servicio.
- **HistoriaClinica:** En esta clase se encuentran las listas de los servicios brindados al paciente
- **Servicio:** En esta clase se encuentra el método para obtener los servicios sin pagar.
- **Cita, Formula, CitaVacuna, Habitacion:** Clases que heredan de servicio y tienen el método para imprimir la descripción del servicio.

Secuencia de la funcionalidad:



Explicación:

1. La funcionalidad comienza solicitando al usuario que ingrese la cédula del paciente.
2. Utiliza el método buscarPaciente del objeto hospital para buscar al paciente correspondiente a esa cédula. Si no se encuentra un paciente con esa cédula, se le pregunta al usuario si desea intentar nuevamente. Si el usuario elige no intentar nuevamente, la función retorna y finaliza.
3. Si se encuentra un paciente válido, se procede a buscar los servicios pendientes de pago para ese paciente utilizando el método estático obtenerServiciosSinPagar de la clase Servicio. Los servicios pendientes de pago se almacenan en una lista llamada serviciosSinPagar.
4. Si la lista de serviciosSinPagar está vacía, se muestra un mensaje indicando que el paciente no tiene servicios pendientes de pago y la función retorna.
5. Si la lista serviciosSinPagar contiene servicios, se muestra una lista de descripciones de esos servicios.

6. A continuación, se solicita al usuario que ingrese la ID del servicio que desea pagar.
7. Se realiza una búsqueda en la lista `serviciosSinPagar` para encontrar el servicio correspondiente a la ID ingresada por el usuario. Si no se encuentra el servicio, se le pide al usuario que intente nuevamente.
8. Una vez que se ha seleccionado un servicio válido, se calcula el precio del servicio utilizando el método `calcularPrecio` del objeto `pacienteSeleccionado`. El cálculo del precio depende del tipo de servicio seleccionado.
9. Se muestra el total a pagar y se le pregunta al usuario si desea realizar el pago.
10. Si el usuario elige realizar el pago, se llama al método `validarPago` del objeto `servicioSeleccionado` para registrar el pago y se muestra un mensaje de "Pago realizado".
11. Si el usuario elige no realizar el pago, se muestra un mensaje de "Pago cancelado".

Ejecución de la funcionalidad:

En las imágenes siguientes se ve el proceso que realiza el programa para pagar el "Ejemplo Servicio 2". También se observa que este servicio no aparece después de validarse el pago.

```
MENU FUNCIONALIDADES
1. Agendar una cita medica
2. Generar fórmula médica
3. Asignar habitación a un paciente
4. Aplicarse una vacuna
5. Facturacion
6. Regresar al menu inicial
7. Salir
Ingrese una opcion:
5
Ingrese la cedula del paciente:
111
Servicios pendientes de pago:
1 - Ejemplo servicio
2 - Ejemplo servicio
3 - Ejemplo servicio
Ingrese la ID del servicio que va a pagar:
2
Total a pagar: $11900.0
Realizar pago? (S/N)
s
Pago realizado
```

```
MENU FUNCIONALIDADES
1. Agendar una cita medica
2. Generar fórmula médica
3. Asignar habitación a un paciente
4. Aplicarse una vacuna
5. Facturacion
6. Regresar al menu inicial
7. Salir
Ingrese una opcion:
5
Ingrese la cedula del paciente:
111
Servicios pendientes de pago:
1 - Ejemplo servicio
3 - Ejemplo servicio
Ingrese la ID del servicio que va a pagar:
```

V. Manual de usuario

Datos precargados en la aplicación

Recomendación: También se pueden ver los datos sobrecargados haciendo uso de gestionar registros - gestionar hospital - ver lista de Habitaciones, ver inventario de medicamentos, ver personas registradas en el hospital o ver vacunas registradas en el hospital.

Pacientes:

- **Nombre:** Juan **Cedula:** 111 **TipoEps:** Subsidiado **Enfermedades:** Caries I, Bronquitis II
historialCitas: **Fecha:** 3 de Abril, 8:00 am **Doctor:** David
Fecha: 3 de Abril, 8:00 am **Doctor:** Lina
- **Nombre:** Jose **Cedula:** 222 **TipoEps:** Contributivo **Enfermedades:** Uveitis I
historialCitas: **Fecha:** 3 de Abril, 8:00 am **Doctor:** Jaime
- **Nombre:** Luz **Cedula:** 333 **TipoEps:** Particular **Enfermedades:** Gingivitis III, Conjuntivitis II, Influenza I
historialCitas: **Fecha:** 3 de Abril, 8:00 am **Doctor:** Oswaldo
Fecha: 3 de Abril, 8:00 am **Doctor:** Cristian
Fecha: 3 de Abril, 8:00 am **Doctor:** Laura

Doctores:

- **Nombre:** David **Cedula:** 444 **TipoEps:** Subsidiado **Especialidad:** General
- **Nombre:** Manuel **Cedula:** 555 **TipoEps:** Subsidiado **Especialidad:** Oftalmologia
- **Nombre:** Lina **Cedula:** 666 **TipoEps:** Subsidiado **Especialidad:** Odontologia
- **Nombre:** Daniela **Cedula:** 777 **TipoEps:** Contributivo **Especialidad:** General
- **Nombre:** Jaime **Cedula:** 888 **TipoEps:** Contributivo **Especialidad:** Oftalmologia
- **Nombre:** Juana **Cedula:** 999 **TipoEps:** Contributivo **Especialidad:** Odontologia
- **Nombre:** Oswaldo **Cedula:** 1111 **TipoEps:** Particular **Especialidad:** General
- **Nombre:** Laura **Cedula:** 2222 **TipoEps:** Particular **Especialidad:** Oftalmologia
- **Nombre:** Cristian **Cedula:** 3333 **TipoEps:** Particular **Especialidad:** Odontologia

Enfermedades:

- **Nombre:** Conjuntivitis **Tipología:** II **Especialidad que la trata:** Oftalmologia
- **Nombre:** Influenza **Tipología:** I **Especialidad que la trata:** General
- **Nombre:** Caries **Tipología:** I **Especialidad que la trata:** Odontologia
- **Nombre:** Uveítis **Tipología:** II **Especialidad que la trata:** Oftalmologia
- **Nombre:** Bronquitis **Tipología:** II **Especialidad que la trata:** General
- **Nombre:** Gingivitis **Tipología:** III **Especialidad que la trata:** Odontologia

Medicamentos:

- **Nombre:** Antibiotico para Conjuntivitis III **Enfermedad:** Conjuntivitis III **Cantidad:** 10 **Precio:** 250000
- **Nombre:** Crema para Uveítis II **Enfermedad:** Uveitis II **Cantidad:** 10 **Precio:** 150000
- **Nombre:** Fluor para caries I **Enfermedad:** Caries I **Cantidad:** 10 **Precio:** 200000
- **Nombre:** doxiciclina para Gingivitis III **Enfermedad:** Gingivitis III **Cantidad:** 10 **Precio:** 20000

- **Nombre:** Mucolíticos para Bronquitis II **Enfermedad:** Bronquitis II **Cantidad:** 10 **Precio:** 10000
- **Nombre:** Oseltamivir para Influenza I **Enfermedad:** Influenza I **Cantidad:** 10 **Precio:** 5000

Vacunas:

- **Nombre:** Covid **Tipo:** No obligatoria **TipoEps:** Subsidiado, Contributivo **Precio:** 10000
- **Nombre:** Hepatitis **Tipo:** Obligatoria **TipoEps:** Subsidiado, Particular, Contributivo **Precio:** 5000
- **Nombre:** Rotavirus **Tipo:** Obligatoria **TipoEps:** Subsidiado, Particular **Precio:** 3000
- **Nombre:** Varicela **Tipo:** No obligatoria **TipoEps:** Particular **Precio:** 2000
- **Nombre:** Neumococo **Tipo:** Obligatoria **TipoEps:** Particular, Contributivo **Precio:** 4000

Habitaciones:

- **Numero:** 1 **CategoriaHabitacion:** CAMILLA **Ocupada:** False **Paciente:** Null **Dias:** 0
- **Numero:** 2 **CategoriaHabitacion:** CAMILLA **Ocupada:** False **Paciente:** Null **Dias:** 0
- **Numero:** 3 **CategoriaHabitacion:** CAMILLA **Ocupada:** False **Paciente:** Null **Dias:** 0
- **Numero:** 1 **CategoriaHabitacion:** INDIVIDUAL **Ocupada:** False **Paciente:** Null **Dias:** 0
- **Numero:** 2 **CategoriaHabitacion:** INDIVIDUAL **Ocupada:** False **Paciente:** Null **Dias:** 0
- **Numero:** 3 **CategoriaHabitacion:** INDIVIDUAL **Ocupada:** False **Paciente:** Null **Dias:** 0
- **Numero:** 1 **CategoriaHabitacion:** DOBLE **Ocupada:** False **Paciente:** Null **Dias:** 0
- **Numero:** 2 **CategoriaHabitacion:** DOBLE **Ocupada:** False **Paciente:** Null **Dias:** 0
- **Numero:** 3 **CategoriaHabitacion:** DOBLE **Ocupada:** False **Paciente:** Null **Dias:** 0
- **Numero:** 1 **CategoriaHabitacion:** OBSERVACION **Ocupada:** False **Paciente:** Null **Dias:** 0
- **Numero:** 2 **CategoriaHabitacion:** OBSERVACION **Ocupada:** False **Paciente:** Null **Dias:** 0
- **Numero:** 3 **CategoriaHabitacion:** OBSERVACION **Ocupada:** False **Paciente:** Null **Dias:** 0
- **Numero:** 1 **CategoriaHabitacion:** UCI **Ocupada:** False **Paciente:** Null **Dias:** 0
- **Numero:** 2 **CategoriaHabitacion:** UCI **Ocupada:** False **Paciente:** Null **Dias:** 0
- **Numero:** 3 **CategoriaHabitacion:** UCI **Ocupada:** False **Paciente:** Null **Dias:** 0
- **Numero:** 1 **CategoriaHabitacion:** UCC **Ocupada:** False **Paciente:** Null **Dias:** 0
- **Numero:** 2 **CategoriaHabitacion:** UCC **Ocupada:** False **Paciente:** Null **Dias:** 0
- **Numero:** 3 **CategoriaHabitacion:** UCC **Ocupada:** False **Paciente:** Null **Dias:** 0

¿Cómo funciona la aplicación?

En esta aplicación a pesar de que hay unos datos precargados, se tiene la libertad de crear objetos solamente ingresando datos por teclado, gracias al apartado de gestión que podemos escoger nada más iniciar la aplicación.

```
MENU INICIAL
1. Servicios para pacientes
2. Gestionar registros
3. --Salir--
Ingrese una opcion:
2
```

En la sección gestionar registros vas a encontrar muchas opciones para crear elementos asociados al hospital (pacientes, doctores, hospital, y vacunas). En la sección servicios para pacientes se encuentran las funcionalidades de nuestra aplicación.

Prueba de funcionalidades

1. Funcionalidad de agendar citas:

Para esta funcionalidad se deben ingresar datos numéricos, tanto en las selecciones como en la cédula del paciente.

```
MENU INICIAL
1. Servicios para pacientes
2. Gestionar registros
3. --Salir--
Ingrese una opcion:
1

MENU FUNCIONALIDADES
1. Agendar una cita medica
2. Generar fórmula médica
3. Asignar habitación a un paciente
4. Aplicarse una vacuna
5. Facturacion
6. --Regresar al menu inicial--
7. --Salir--
Ingrese una opcion:
1
Ingrese su numero de cedula: 222
Bienvenido Usuario Jose

Seleccione el tipo de cita que requiere
1. General
2. Odontologia
3. Oftalmologia
4. --Regresar al menu--
Ingrese una opcion: 3

Lista de oftalmologos para el tipo de eps Contributivo:
1. Jaime

Seleccione el doctor con el que quiere la cita: 1

Citas disponibles:
1. 4 de Abril, 3:00 pm
2. 5 de Abril, 10:00 am

Seleccione la cita de su preferencia: 2

Su cita ha sido asignada con exito

Resumen de su cita:
Fecha: 5 de Abril, 10:00 am
Paciente: Jose
Doctor: Jaime
```

Hay que tener en cuenta que según el tipo de eps del paciente, sólo se mostrarán los doctores que sean del mismo tipo de eps del paciente en cada categoría, en este caso, Jose al tener tipo de eps contributivo, y seleccionar oftalmología, solo se muestra el doctor Jaime, que es el único oftalmólogo con tipo de eps contributivo que hay en los datos precargados.

```
historial citas de historia clinica del paciente:

Fecha: 3 de Abril, 8:00 am
Paciente: Jose
Doctor: Jaime

Fecha: 5 de Abril, 10:00 am
Paciente: Jose
Doctor: Jaime

¡Adiós Jose del servicio cita médica!
```

Una vez se haya elegido la cita, se actualizará tanto la agenda de ese doctor como la historia clínica del paciente, por lo que al volver a agendar otra cita, ya no aparecerá la que se había agendado anteriormente. En este caso, Jose ya ha agendado 2 citas con Jaime, por lo que estas ya no volverán a aparecer en el momento de agendar otra cita.

2. Funcionalidad de generar fórmulas médicas:

Observación: Para que se ejecute la funcionalidad a la perfección el paciente debe tener mínimo una cita médica con un doctor que el atributo **especialidad** sea el mismo para el atributo de la instancia de **Enfermedad**, que es la enfermedad al paciente asociado, también las instancias de **Medicamento** deben tener de atributo la instancia de **Enfermedad** que se desea tratar y el atributo **cantidad** del medicamento debe ser mayor que cero.

Para la funcionalidad se tiene en cuenta en la primera interacción la entrada tipo numérica para seleccionar el menú de servicios para pacientes.

```
MENU INICIAL
1. Servicios para pacientes
2. Gestionar registros
3. --Salir--
Ingrese una opcion:
1

MENU FUNCIONALIDADES
1. Agendar una cita medica
2. Generar fórmula médica
3. Asignar habitación a un paciente
4. Aplicarse una vacuna
5. Facturacion
6. --Regresar al menu inicial--
7. --Salir--
Ingrese una opcion:
2
Ingrese la cédula del paciente:
```

- Este menú también solo recibe entradas numéricas, entonces seleccionamos la opción 2 para generar la fórmula médica

```
MENU FUNCIONALIDADES
1. Agendar una cita medica
2. Generar fórmula médica
3. Asignar habitación a un paciente
4. Aplicarse una vacuna
5. Facturacion
6. --Regresar al menu inicial--
7. --Salir--
Ingrese una opcion:
2
```

- Después de iniciar la funcionalidad, nos pedirá una entrada de tipo entero que será la cédula de algún paciente registrado, después de ingresar el número de cédula se imprime la lista de las enfermedades del paciente y se debe ingresar un número entero para seleccionar la enfermedad a tratar.

```
¿Que enfermedad deseas tratar?
1.Conjuntivitis II Especialidad que la trata: Oftalmologia
2.Gingivitis III Especialidad que la trata: Odontologia
3.Influenza I Especialidad que la trata: General
1
```

- Al seleccionar la enfermedad nos retornará con los doctores que han atendido al paciente y que tenga la misma especialidad que trata la enfermedad (para que la funcionalidad sirva este paciente deberá tener citas en su historial de citas con algún doctor que tenga la misma especialidad que trata las enfermedades del paciente), ahora ingresamos un número entero de nuevo para seleccionar que doctor se desea que genere la fórmula médica.

```
Los doctores que lo han atendido y están disponibles para formular Conjuntivitis II son:
1.Laura
1
```

- Después se pasa a ingresar los medicamentos de la fórmula médica, entonces se imprimirán los medicamentos disponibles que tratan la enfermedad del paciente y seleccionamos cuál agregar.

```
Hola doctor, Laura
Por favor selecciona los medicamentos que vas a formularle a: Luz
1.Nombre: Antibiotico Conjuntivitis III, Enfermedad: Conjuntivitis II, Descripcion: Cura la enfermedad, Cantidad: 10, Precio: 250000.0
1
```

- Ahora se nos imprime la lista actual de medicamentos y se nos pregunta si deseamos agregar otro, aca solo recibe dos entradas “s” para continuar el bucle o “b” para detenerlo, si ingresamos “s” de nuevo nos mostrará medicamentos disponibles y volvemos a elegir cuál agregar

```
sta es tu lista actual de medicamentos:[Nombre: Antibiotico Conjuntivitis III, Enfermedad: Conjuntivitis II, Descripcion: Cura la enfermedad]
.Nombre: Antibiotico Conjuntivitis III, Enfermedad: Conjuntivitis II, Descripcion: Cura la enfermedad
Desea agregar otro medicamento? (s/n)

ola doctor, Laura
or favor selecciona los medicamentos que vas a formularle a: Luz
.Nombre: Antibiotico Conjuntivitis III, Enfermedad: Conjuntivitis II, Descripcion: Cura la enfermedad, Cantidad: 9, Precio: 250000.0

sta es tu lista actual de medicamentos:[Nombre: Antibiotico Conjuntivitis III, Enfermedad: Conjuntivitis II, Descripcion: Cura la enfermedad, Nombre: Antibiotico Conjuntivitis III, Enfermedad: Conjuntivitis II, Descripcion: Cura la enfermedad]
.Descripcion: Cura la enfermedad]
.Nombre: Antibiotico Conjuntivitis III, Enfermedad: Conjuntivitis II, Descripcion: Cura la enfermedad
.Nombre: Antibiotico Conjuntivitis III, Enfermedad: Conjuntivitis II, Descripcion: Cura la enfermedad
Desea agregar otro medicamento? (s/n)
```

- Y si seleccionamos “n” se para el bucle y solo nos imprime el resumen de la fórmula

```
{Desea agregar otro medicamento? (s/n)
n
Hola Luz
Estos son tus medicamentos formulados:
[Nombre: Antibiotico Conjuntivitis III, Enfermedad: Conjuntivitis II, Descripcion: Cura la enfermedad, Nombre: Antibiotico Conjuntivitis III, Enfermedad: Conjuntivitis II
```

Ahí termina la funcionalidad

3. Funcionalidad de asignar habitación:

Para la funcionalidad se tiene en cuenta las entradas tipo numéricas tanto al momento de las elecciones como en el ingreso de datos.

```
"C:\Program Files\Java\jdk-17\bin\java.exe"

MENU INICIAL
1. Servicios para pacientes
2. Gestionar registros
3. --Salir--
Ingrese una opcion:
1
```

Al comienzo de ejecutar nuestro proyecto al usuario le aparecerá un menú inicial que consta de 3 opciones como se ve en la imagen, para utilizar las funcionalidades escogemos la opción número 1.

```
MENU FUNCIONALIDADES
1. Agendar una cita medica
2. Generar fórmula médica
3. Asignar habitación a un paciente
4. Aplicarse una vacuna
5. Facturacion
6. --Regresar al menu inicial--
7. --Salir--
Ingrese una opcion:
3
```

Ya después de haber seleccionado la opción nos aparecerá un nuevo menú que consta de 7 elecciones, de las cuales las 5 primeras son nuestras funcionalidades, pero ahora la que nos interesa es la de “Asignar habitación a un paciente” dicho esto escogemos la opción número 3.

```
Ingrese el número de identificación del paciente:
111
Elija el tipo de habitacion que desee, recuerde que segun el tipo va el costo del servicio
1. CAMILLA
2. OBSERVACION
3. UCI
Ingrese una opción: 3
```

Cuando seleccionamos dicha opción nos desplegará un mensaje pidiéndonos que ingresemos el número de identificación del paciente al cual queremos asignarle una habitación, al momento de ingresar el número de documento, se comprobará que el paciente esté registrado en el hospital y si no esta nos retorna al menú de las funcionalidades; en este caso el paciente si se encuentra registrado y al buscarlo en la lista que contiene a todos lo pacientes se comprueba que tipo de EPS es el paciente y desplegando las categorías que tiene disponibles según su EPS, se seleccione la que prefiera o necesite en este caso decidí escoger la categoría UCI.

```
Elija el tipo de habitacion que desee, recuerde que segun el tipo va el costo del servicio
1. CAMILLA
2. OBSERVACION
3. UCI
Ingrese una opción: 3
Habitaciones disponibles de la categoría UCI:
0. Número ID: 1
Seleccione la habitación deseada (ingrese un número del 0 al 0):
0
```

Luego de haber escogido la categoría nos despliega nuevamente un menú de opciones que nos muestra todas las habitaciones disponibles que cuenta la categoría (para este ejemplo elimine algunas opciones en el apartado de gestionar registro del menú inicial de esta categoría para mostrar cómo interactúa cuando no encuentra habitaciones disponibles), la cual nos mostrará unas opciones que van desde 0 hasta el número de habitaciones libres y al frente de cada opción nos mostrará el número de ID de cada habitación.

```
Ingrese el número estimado de días para la estadia en la habitacion:
5

Datos de la habitacion del Paciente:
Número de ID de la habitación: 1
Categoria de la habitación: UCI
Cedula del Paciente: 111
Nombre del Paciente: Juan
```

Ya cuando escogemos la opción que más nos guste o la que tuviéramos disponible, nos pedirá que ingrese los días estimados para la estadía en la habitación y después nos mostrará un resumen de la información de las habitación y del paciente que la seleccionó.


```

MENU FUNCIONALIDADES
1. Agendar una cita medica
2. Generar fórmula médica
3. Asignar habitación a un paciente
4. Aplicarse una vacuna
5. Facturacion
6. --Regresar al menu inicial--
7. --Salir--
Ingrese una opcion:
3
Ingrese el número de identificación del paciente:
222
Elija el tipo de habitacion que desee, recuerde que segun el tipo va el costo del servicio
1. INDIVIDUAL
2. DOBLE
3. OBSERVACION
4. UCI
5. UCC
Ingrese una opción: 4

```

Ahora explicare que pasa cuando no se encuentra habitaciones disponibles, hacemos lo mismo que la vez anterior hasta cuando nos pide seleccionar la categoria de habitacion necesaria o de preferencia, como se puede ver este paciente dispone de de mas opciones ya que su EPS es distinta al del anterior ejemplo, seleccionando UCI, he optado por esta elección con el fin de favorecer el desarrollo de la explicación.

```

No hay habitaciones disponibles de la categoría UCI
¿Desea asignar una habitación de otra categoría inferior? (s/n)
S

```

Al momento de buscar las habitaciones disponibles que tiene esta categoría y no encontrar habitaciones disponibles nos mostrará un mensaje con dos opciones la cual una es cambia de categoría por una inferior o terminar el proceso, para este ejemplo seleccionamos “s” para continuar.

```

Habitaciones disponibles de la categoría inferior:
0. Número ID: 1
1. Número ID: 2
2. Número ID: 3
Seleccione la habitación deseada (ingrese un número del 0 al 2):
1

```

Se nos despliega un menú de opciones con las habitaciones disponibles que cuenta la categoría inferior, seleccionamos la que guste o la que este disponible, yo en este ejemplo selecciono la opción número 1.

```
"Ingrese el número estimado de días para la estadia en la habitacion:
```

```
8
```

```
Datos de la habitacion del Paciente:
```

```
Número de ID de la habitación: 2
```

```
Categoria de la habitación: OBSERVACION
```

```
Cedula del Paciente: 222
```

```
Nombre del Paciente: Jose
```

Y ya nos pide que ingresemos el numero de días estimados y nos imprime el resumen de la habitación y el paciente y ya termina la funcionalidad, es importante decir que cada habitación también almacena el paciente que la escogió y viceversa es decir que si nosotros queremos escoger una habitación nuevamente y ingresamos la cédula de algún paciente que ya haya seleccionado, la funcionalidad no se lo permitirá y nos mostrará un mensaje diciendo “El paciente ya tiene asignado una habitación”.

4. Funcionalidad de vacunas:

Para esta funcionalidad se deben ingresar datos numéricos, tanto en las selecciones como en la cédula del paciente.

```
MENU FUNCIONALIDADES
```

```
1. Agendar una cita medica
```

```
2. Generar fórmula médica
```

```
3. Asignar habitación a un paciente
```

```
4. Aplicarse una vacuna
```

```
5. Facturacion
```

```
6. --Regresar al menu inicial--
```

```
7. --Salir--
```

```
Ingrese una opcion:
```

```
4
```

```
Ingrese la cédula del paciente:
```

```
111
```

```
Bienvenido Usuario Juan
```

```
Seleccione el tipo de vacuna que requiere
```

```
1. Obligatoria
```

```
2. No obligatoria
```

```
Ingrese una opcion:
```

```
1
```

```
Vacunas Disponibles
```

```
1.Hepatitis
```

```
2.Rotavirus
```

```
Seleccione la vacuna que requiere aplicarse:
```

```
1
```


Juan tiene tipo de eps subsidiado, y se ha escogido vacunas obligatorias, en los datos sobrecargados se puede ver que Hepatitis y Rotavirus son obligatorias y además presentan disponibilidad para Subsidiado.

```
Seleccione la vacuna que requiere aplicarse:
1

Citas disponibles:
1. 3 de Abril, 8:00 am
2. 3 de Abril, 11:00 am
3. 4 de Abril, 3:00 pm
4. 5 de Abril, 10:00 am

Seleccione la cita de su preferencia:
1

Cita asignada correctamente, puede acudir al centro asistencial con la siguiente informacion:

Resumen de su cita:
Fecha: 3 de Abril, 8:00 am
Paciente: Juan
Vacuna: Hepatitis
Asistente médico: Enfermera

Este es el historial de vacunas aplicadas del paciente seleccionado:
1. Vacuna: Hepatitis

¡Adiós Juan del servicio de vacunas!
```

Posteriormente, se muestran las citas disponibles de Hepatitis, y se le asigna a Juan la cita del 3 de Abril a las 8:00 am, también se registra en su historia clínica.

Es decir que si se vuelve a ingresar con Juan, y se selecciona Hepatitis, el programa no dejará hacer esta elección.

5. Facturación

Opcional: Antes de probar esta funcionalidad se recomienda usar las anteriores para generar más servicios y que se pueda apreciar mejor la funcionalidad. Sin embargo es perfectamente usable sin necesidad de agregarlos como se verá a continuación.

Después de seleccionar la opción de facturación del menú de funcionalidades encontrará el primer ingreso: la cédula del paciente que recibió los servicios que desea pagar. El ingreso debe ser numérico además de ser una cédula que exista en la base de datos. Si el ingreso es invalido tendra la opcion de intentar de nuevo o regresar al menú de funcionalidades. En este ejemplo ingresamos la cédula "111" del paciente Juan.

```
MENU FUNCIONALIDADES
1. Agendar una cita medica
2. Generar fórmula médica
3. Asignar habitación a un paciente
4. Aplicarse una vacuna
5. Facturacion
6. Regresar al menu inicial
7. Salir
Ingrese una opcion:
5
Ingrese la cedula del paciente:
111
```

A continuación se le mostraran los servicios del paciente que estén pendientes por pagar. Usted deberá seleccionar el servicio que desea pagar ingresando el ID del servicio. El ingreso debe ser numérico y debe ser un ID de los que se muestran en la pantalla. El programa no continuará hasta que ingrese un ID valido. En este ejemplo ingresamos el ID del Ejemplo Servicio 2

```
Servicios pendientes de pago:
1 - Ejemplo servicio
2 - Ejemplo servicio
3 - Ejemplo servicio
Ingrese la ID del servicio que va a pagar:
2
```

Para finalizar la funcionalidad de pago, se le mostrará en pantalla el precio del servicio que desea pagar y se le pedirá confirmación del pago. Si confirma el pago, se le informará en pantalla la realización del pago. De lo contrario se cancelará el proceso de pago y regresará al menú de funcionalidades. En este ejemplo se confirma el pago de Ejemplo Servicio 2.

```
Total a pagar: $11900.0
Realizar pago? (S/N)
s
Pago realizado
```

Si el pago del servicio fue realizado con éxito podrá observar en futuras interacciones con esta funcionalidad que obviamente no aparecerá de nuevo. En este ejemplo el Ejemplo Servicio 2 no aparece al tratar de pagar otro servicio para el paciente Juan.

MENU FUNCIONALIDADES

1. Agendar una cita medica
2. Generar fórmula médica
3. Asignar habitación a un paciente
4. Aplicarse una vacuna
5. Facturacion
6. Regresar al menu inicial
7. Salir

Ingrese una opcion:

5

Ingrese la cedula del paciente:

111

Servicios pendientes de pago:

1 - Ejemplo servicio

3 - Ejemplo servicio

Ingrese la ID del servicio que va a pagar: