Trabajo práctico #2
Manejo de interfaces graficas de usuario y Excepciones

Sistema de registro hospitalario basado en objetos

Programación orientada a objetos

Grupo 02
Equipo 03

Integrantes:
Samuel Botero Rivera,
Samuel Gutierrez Betancur,
Samuel Garcia Rojas

Profesor

Universidad Nacional de Colombia - Sede Medellín

2024-2

Tabla de Contenidos

I. Descripción general de la solución

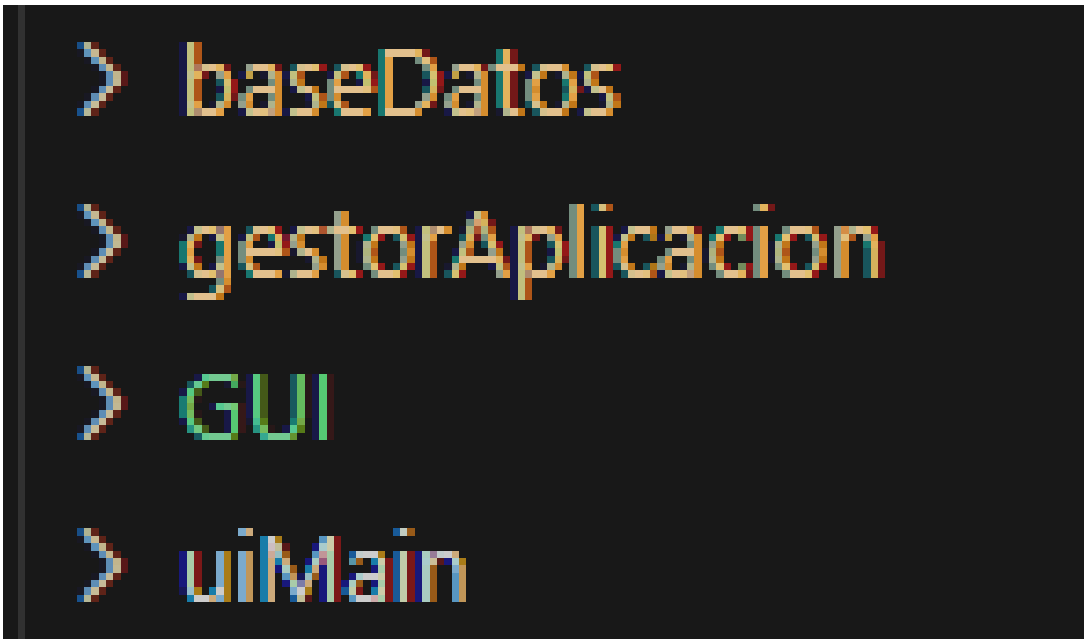
Introducción

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).

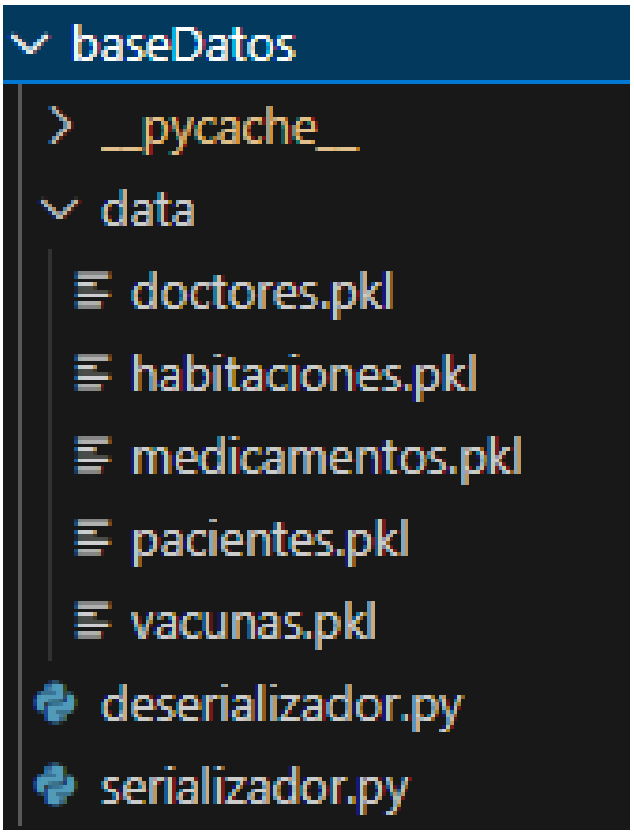
El primer análisis al que nos lleva este proyecto es pensar en todos los elementos involucrados alrededor del funcionamiento de un hospital. Decidimos centrarnos en aspectos muy comunes relacionados al área de la salud y su manejo, tales como, citas médicas, vacunas, medicamentos, reserva de habitaciones, 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 4 necesidades fundamentales que se deben discutir, organizadas de esta manera:



1. Capa de persistencia (baseDatos)

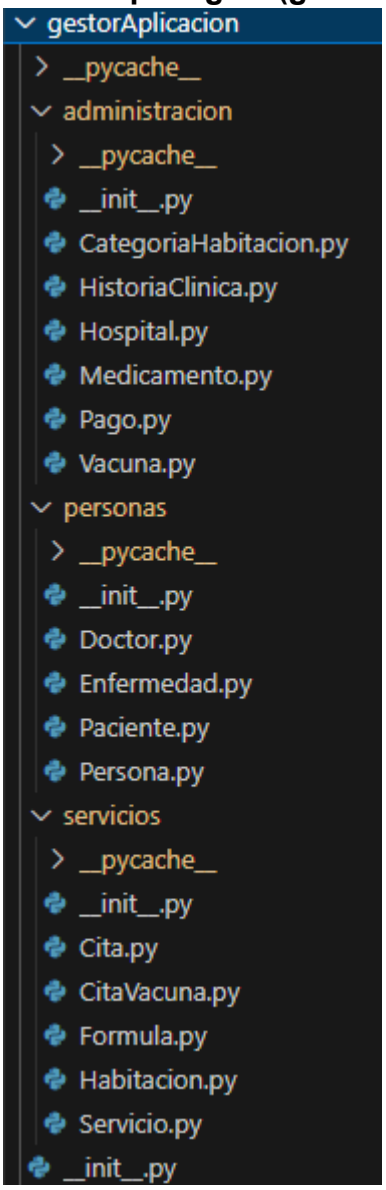


Con esta capa se implementa la persistencia en los datos del programa, los cuales deben ser estables en el tiempo, permitiendo que las funcionalidades aumenten su usabilidad y no estén limitadas al tiempo de ejecución.
Es por esto, que en primer lugar, se hace en la estructura del proyecto, una capa 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 .pkl, 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)

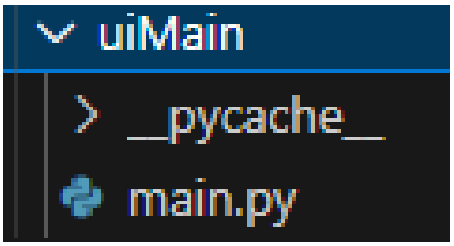


En el paquete de gestorAplicacion tenemos:

- El paquete administracion donde están las clases:
  - El enumerado CategoriaHabitacion
  - HistoriaClinica
  - Hospital
  - Medicamento
  - Vacuna
  - La interfaz Pago

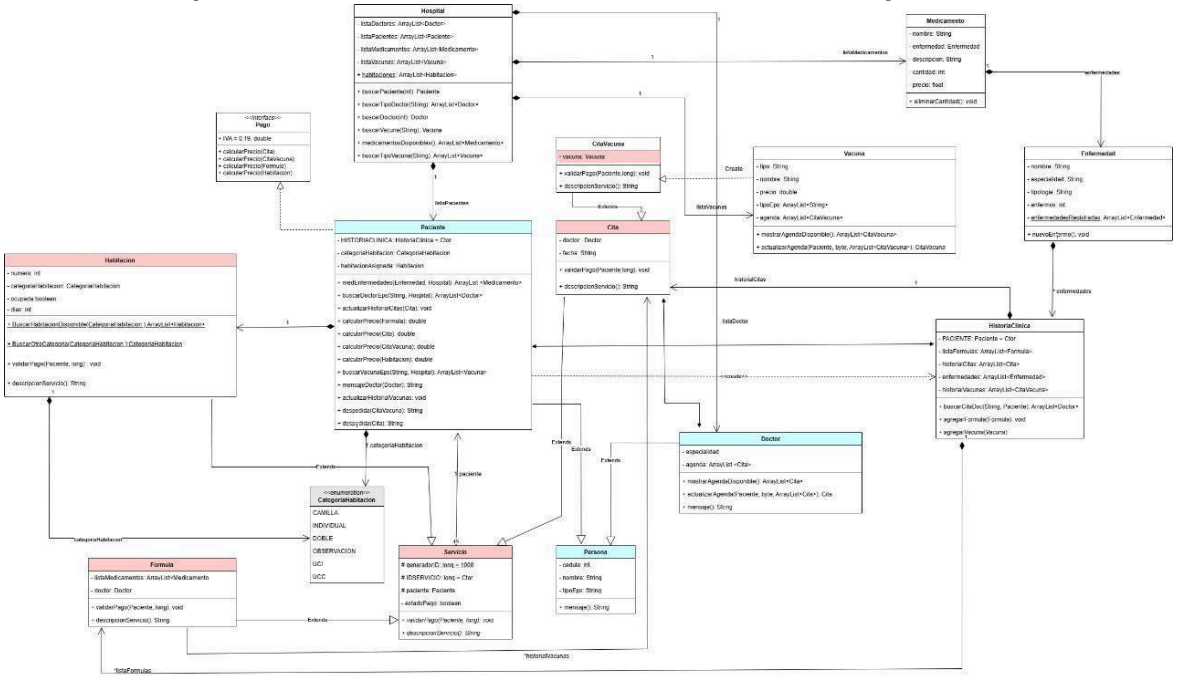
- El paquete personas donde están las clases:
  - Doctor
  - Enfermedad
  - Paciente
  - Persona
- El paquete servicios donde están las clases:
  - Cita
  - CitaVacuna
  - Formula
  - Habitacion
  - La clase abstracta Servicio

3. Capa asociada al Método principal (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 un solo archivo en el paquete, este es el Main del proyecto, donde se encuentra toda la logica detras del proyecto, se incorporan las funcionalidades y se crean las instancias de las diferentes clases.

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



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

Clase abstracta

```python
from abc import ABC, abstractmethod

class Servicio(ABC):
    generadorID = 1000

    def __init__(self, paciente: 'Paciente'):  # Use string type hint for Paciente
        self.idServicio = Servicio.generadorID
        Servicio.generadorID += 1
        self.paciente = paciente
        self.estadoPago = False

    @abstractmethod
    def validarPago(self):
        pass
```

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

```python
@abstractmethod
def validarPago(self):
    pass
```

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, recalcando que en cada clase hija de “Servicio” el método tiene una lógica distinta.

En esta misma clase tenemos otro metodo 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

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”

```python
class Doctor(Persona):
    def __init__(self, cedula: int, nombre: str, tipo_eps: str, especialidad: str):
        super().__init__(cedula, nombre, tipo_eps)
        self.especialidad = especialidad
        self.agenda = [
            self._crear_cita("3 de Abril, 8:00 am", None),  # Use a helper method
            self._crear_cita("4 de Abril, 3:00 pm", None),
            self._crear_cita("5 de Abril, 10:00 am", None)
        ]
```

```python
class Paciente(Persona):
    def __init__(self, cedula: int, nombre: str, tipo_eps: str, categoria_habitacion: CategoriaHabitacion = None):
        super().__init__(cedula, nombre, tipo_eps)
        self.categoria_habitacion = categoria_habitacion
        self.habitacion_asignada = None
        self.historia_clinica = HistoriaClinica(self)

    def get_cedula(self):
        return self._cedula

    def med_enfermedad(self, enfermedad, hospital) -> list:
```

```python
class Persona:
    def __init__(self, cedula: int, nombre: str, tipo_eps: str):
        # Asignar directamente al atributo interno para evitar usar el setter de la propiedad
        self._cedula = cedula
        self.nombre = nombre
        self.tipo_eps = tipo_eps
```

- En el paquete “gestorAplicacion.servicios” las clases “Cita”, “Formula” y “Habitación” heredan de la clase asbtracta “Servicio”, adicionalmente, la clase “CitaVacuna” hereda de la clase “Cita”

```python
class Servicio(ABC):
    generadorID = 1000

    def __init__(self, paciente: 'Paciente'):  #
        self.idServicio = Servicio.generadorID
        Servicio.generadorID += 1
        self.paciente = paciente
        self.estadoPago = False
```

```python
class Cita(Servicio):
    def __init__(self, doctor: Doctor, fecha: str, paciente: Paciente):
        super().__init__(paciente)
        self.doctor = doctor
        self.fecha = fecha
```

```python
class CitaVacuna(Cita):
    def __init__(self, fecha: str, paciente, vacuna):
        super().__init__(None, fecha, paciente)
        self.vacuna = vacuna
```

Ligadura dinámica

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

En la clase Paciente, el método calcular_precio_formula, calcular_precio_cita_vacuna, y calcular_precio_habitacion reciben objetos de diferentes tipos (Formula, CitaVacuna, Habitacion), pero todos heredan de la clase base Servicio. Cada subclase implementa su propia lógica para calcular el precio (ej: CitaVacuna incluye el costo de la vacuna, Habitacion multiplica días por categoría).

Ligadura dinámica:

- Cuando se llama a métodos como calcular_precio_formula(formula) desde el Paciente, Python resuelve en tiempo de ejecución qué implementación de calcular_precio usar, dependiendo del tipo real del objeto.

```python
def calcular_precio_formula(self, formula) -> float: ···

def calcular_precio_cita(self, cita_asignada) -> float: ···

def calcular_precio_cita_vacuna(self, cita_asignada) -> float: ···

def calcular_precio_habitacion(self, habitacion_asignada) -> float: ···
```

La clase CitaVacuna hereda de Cita y sobrescribe el método mostrar_agenda_disponible(), que filtra solo las citas no asignadas. En la función vacunacion(), se llama a este método desde una

instancia de CitaVacuna, pero el código trata al objeto como tipo Cita.

- Aunque la variable se declara como Cita, Python ejecuta la implementación de CitaVacuna si el objeto es de esa subclase.

```
# Mostrar la agenda disponible para la vacuna seleccionada
agenda_disponible = vacuna_seleccionada.mostrar_agenda_disponible()
if not agenda_disponible:
    print("No hay citas disponibles para esta vacuna.")
    return

print("\nCitas disponibles para la vacuna:")
for idx, cita in enumerate(agenda_disponible, start=1):
    print(f"{idx}. Fecha: {cita.get_fecha()}")

try:
    num_cita = int(input("Seleccione la cita de su preferencia: "))
except ValueError:
    print("Opción inválida.")
    return
if num_cita < 1 or num_cita > len(agenda_disponible):
    print("Selección fuera de rango.")
    return
```

Atributo de clase

En la clase “Hospital”, ubicada en el paquete “gestorAplicacion.administracion”, se ha implementado el atributo estatico “habitaciones” como un una lista estática de habitaciones. 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.

```
class Hospital:
    # Atributo de clase (estático) para las habitaciones:
    habitaciones: List[Habitacion] = []

    def __init__(self):
        # Estos atributos son de instancia:
        self.lista_pacientes: List[Paciente] = []
        self._lista_doctores: List[Doctor] = []
        self.lista_medicamentos: List[Medicamento] = []
        self.lista_vacunas: List[Vacuna] = []
```

Método de clase

En la clase “Habitacion”, ubicada en el paquete “gestorAplicacion.servicios”, se

implementa el metodo estático “BuscarHabitacionesDisponibles” que retorna una Lista de objetos de tipo “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

```python
@staticmethod
def buscar_habitacion_disponible(categoria: CategoriaHabitacion):
    from gestorAplicacion.administracion.Hospital import Hospital  # Local
    return [
        habitacion
        for habitacion in Hospital.habitaciones
        if habitacion.categoria == categoria and not habitacion.ocupada
    ]
```

Encapsulamiento

El proyecto demuestra encapsulamiento al agrupar datos y comportamientos relacionados dentro de clases específicas, como Hospital, Paciente, Doctor y Vacuna. Cada clase administra su propio estado y operaciones, asegurando que la manipulación de atributos se realice a través de métodos definidos, lo que protege la integridad de los datos y facilita la mantenibilidad del código.

```python
class Paciente(Persona):
    def __init__(self, cedula: int, nombre: str, tipo_eps: str, categoria_habitacion: CategoriaHabitacion = None):
        super().__init__(cedula, nombre, tipo_eps)
        self.categoria_habitacion = categoria_habitacion
        self.habitacion_asignada = None
        self.historia_clinica = HistoriaClinica(self)
```

Enumeración

La clase enum "CategoriaHabitacion", ubicada en el paquete "gestorAplicacion.administracion", define las diferentes categorías de habitaciones disponibles en un hospital. Utilizando la sintaxis de enum, cada categoría se asocia con un valor numérico que representa su costo, por ejemplo, "CAMILLA" tiene un valor de 50000 y "UCC" de 1500000. Aunque no se define explícitamente un constructor, la asignación de valores se realiza de forma inmutable, garantizando que cada

categoría mantenga su valor original. Además, la clase incluye el método "get_valor" que permite acceder al precio asociado a cada categoría, ayudando en la funcionalidad de "AsignarHabitacion" para determinar el costo a pagar según la EPS del paciente y asignar el precio correspondiente a cada tipo de habitación.

```python
from enum import Enum

class CategoriaHabitacion(E
    CAMILLA = 50000
    INDIVIDUAL = 150000
    DOBLE = 320000
    OBSERVACION = 500000
    UCI = 1300000
    UCC = 1500000

    def get_valor(self):
        return self.value

```

Excepciones

Se implementaron excepciones las cuales se dividen de una clase ErrorAplicacion que hereda de Excepcion

```python
class ErrorAplicacion(Exception):
    def __init__(self, mensaje: str):
        super().__init__(f"Manejo de errores de la Aplicación-> {mensaje}")


class ErrorRegistro(ErrorAplicacion):
    def __init__(self, mensaje: str):
        super().__init__(f"Error en la gestión de registros: {mensaje}")


class ErrorCampoVacio(ErrorEntrada):
    def __init__(self):
        super().__init__()
        self.message = f"El campo no puede estar vacío."
        super().__init__(self.message)

```

Las dos clases de excepciones son ErrorRegistro y ErrorCampoVacio

Todas las demás excepciones heredan de estas dos clases

III. Descripción de las 5 funcionalidades implementadas.

IV.

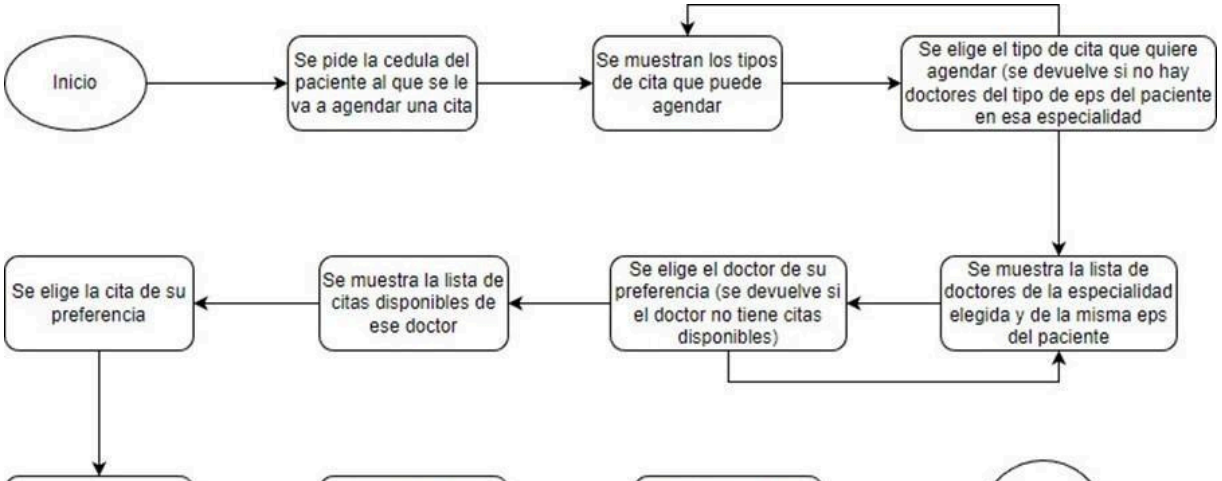
Funcionalidad 1: Agendar citas

El cuerpo de la funcionalidad se encuentra en el paquete “iuMain” en la clase “Main”.

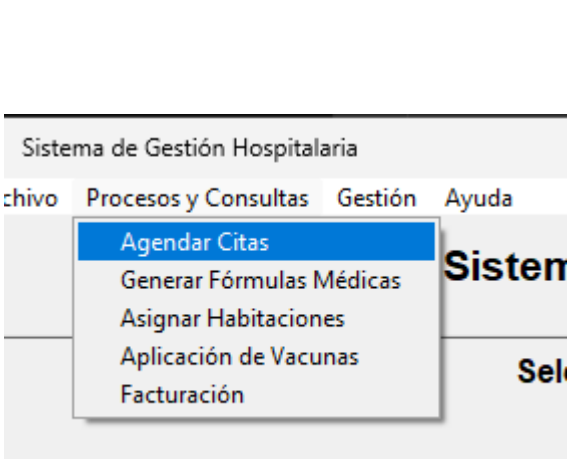
Esta funcionalidad se encarga de realizar el agendamiento de una cita médica en la especialidad que se requiera, con la libertad de elegir el doctor y la fecha entre las que estén disponibles. Durante su proceso de ejecución, intervienen varias clases:

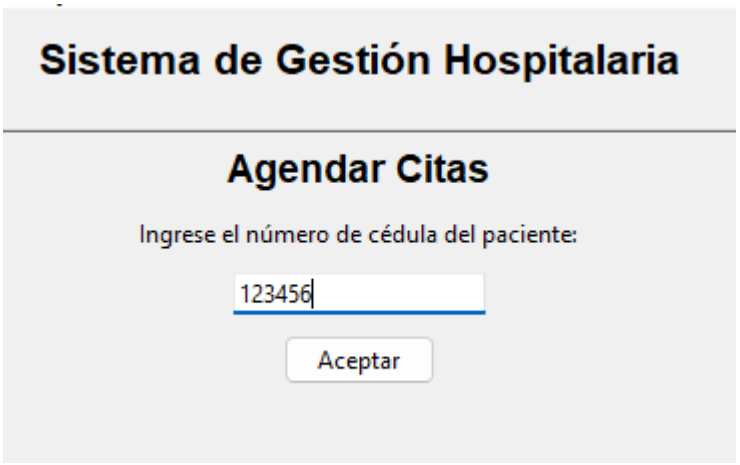
- Hospital: De esta se sacarán las listas de pacientes y doctores para filtrarlas
- Paciente: En esta se filtraran los doctores de acuerdo con el tipo de eps del paciente
- Doctor: De esta se sacarán las citas que tenga disponibles para que el paciente escoja
- Cita: Son los objetos que estarán en la 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

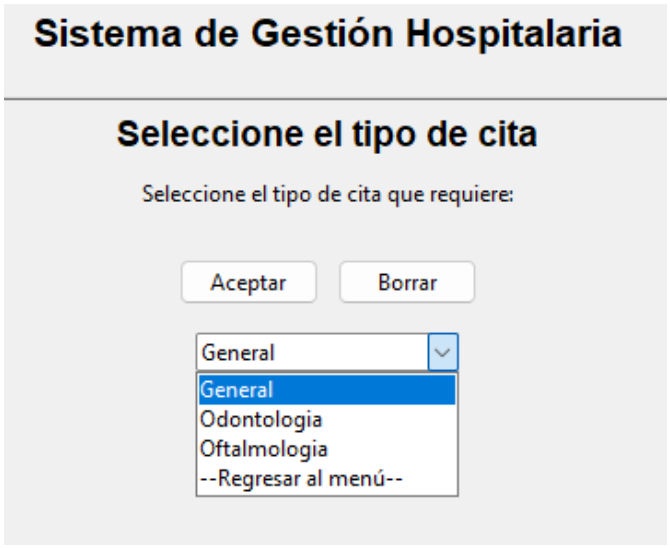
Secuencia de la funcionalidad:

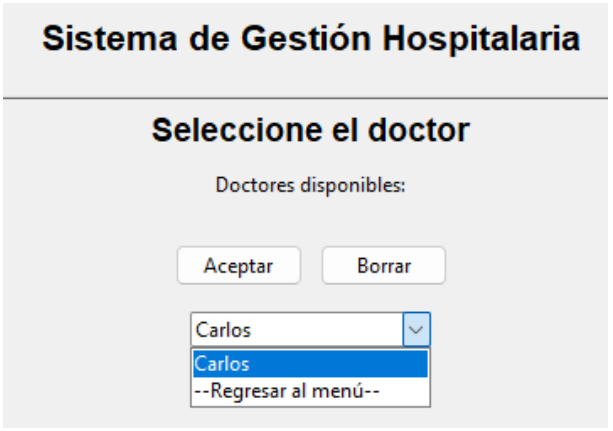


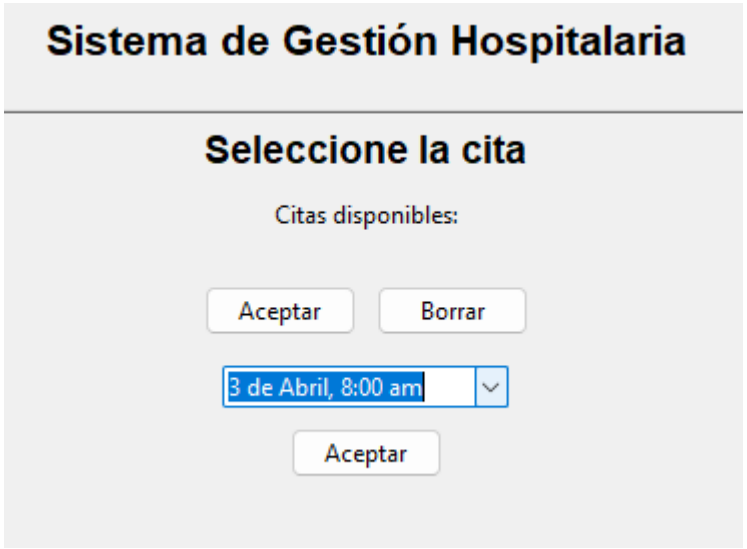
Ejecución de la funcionalidad:

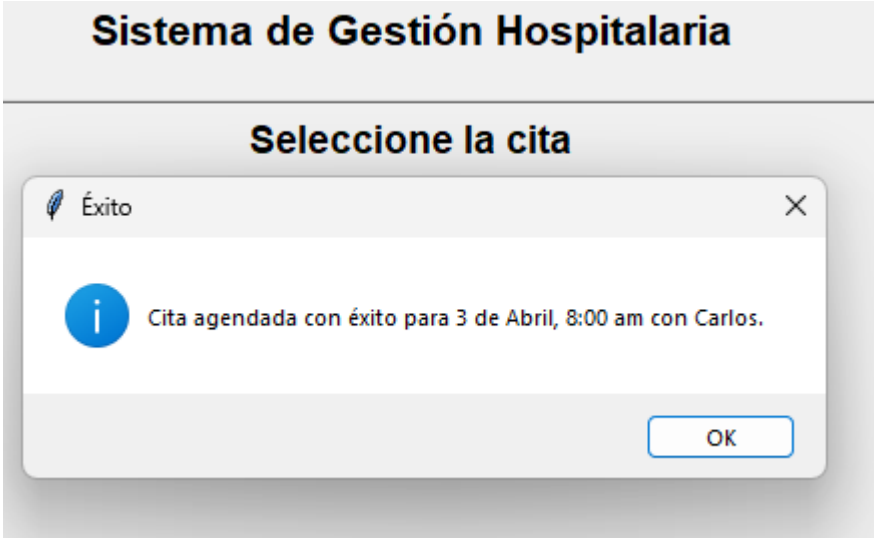












Explicación:

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.

- 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.

- Despues de escojer 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.

Funcionalidad 2: Generar fórmulas médicas

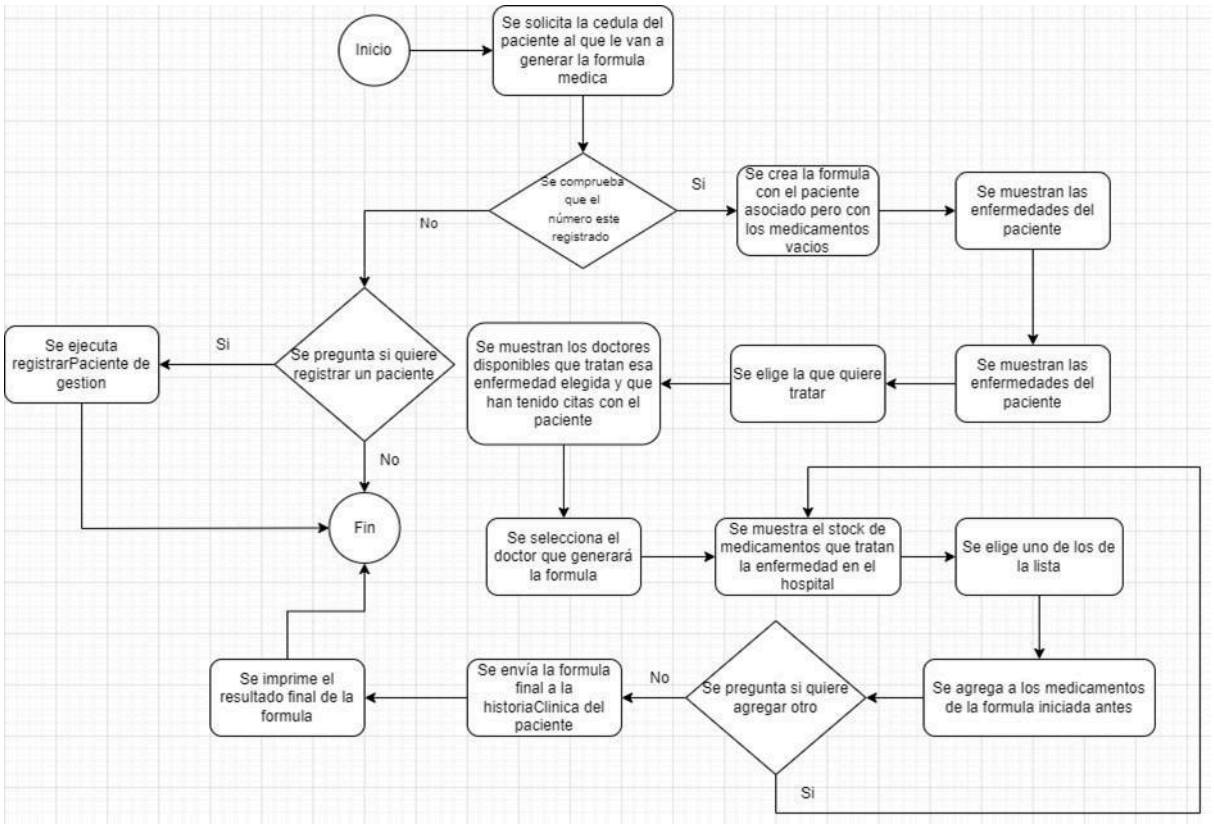
El cuerpo de la funcionalidad se encuentra en el paquete “iuMain” en la clase “Main”.

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:

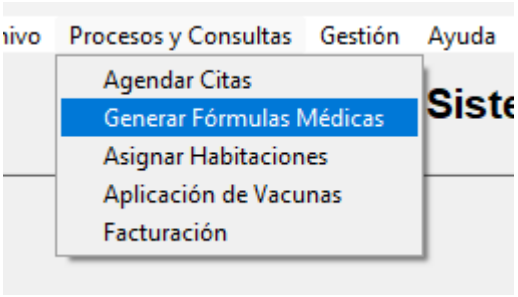
- Hospital: Donde buscaremos en la listaPacientes el paciente asociado por medio del método buscarPaciente

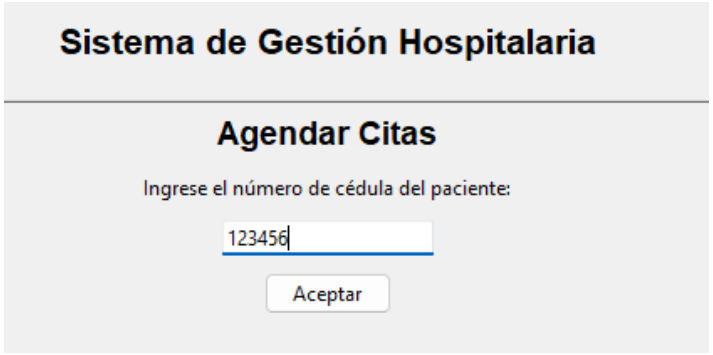
- HistoriaClinica: Es donde se encuentran las enfermedades del paciente y donde está el método “buscarCitaDoc” que encontrará los doctores que tratan la enfermedad seleccionada en el historial de citas de la historia clínica.
- Paciente: Donde se ubica el metodo “medEnfermedad” que buscará los medicamentos 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.

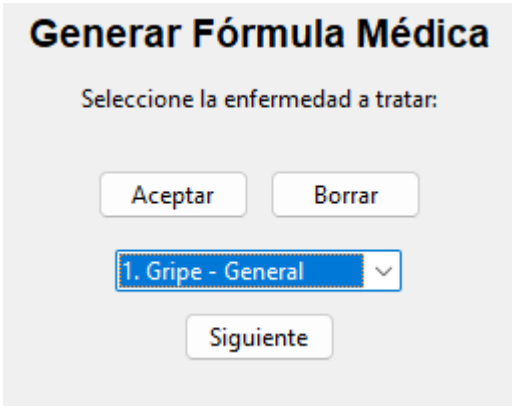
Secuencia de la funcionalidad:

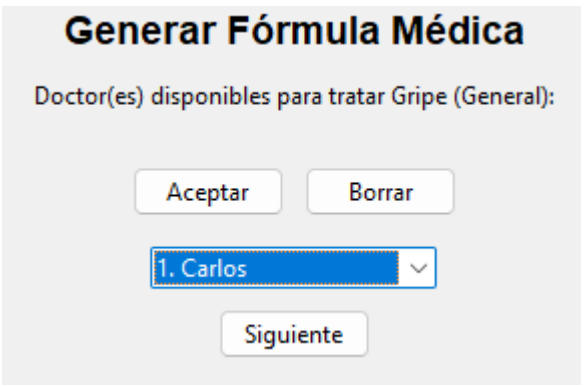


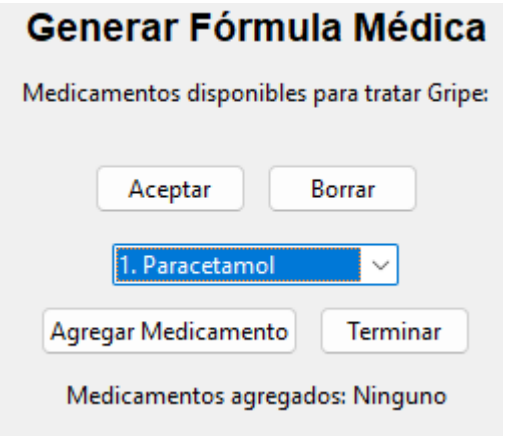
Ejecución de la funcionalidad:

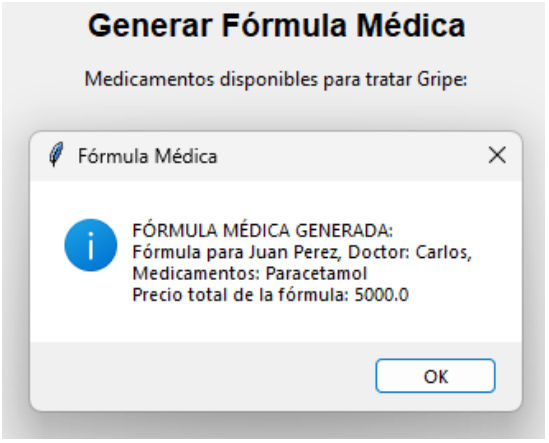












Explicación:

- 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
- 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 continue se deben tener citas con algun doctor con la misma especialidad con la que se trata la enfermedad, ya agendadas en el historialCitas y este método nos retonara un ArrayList con instancias de la clase "Doctor" que sean de la especialidad de la enfermedad y que hayan atendido alguna vez al paciente.

- 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.

- 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.

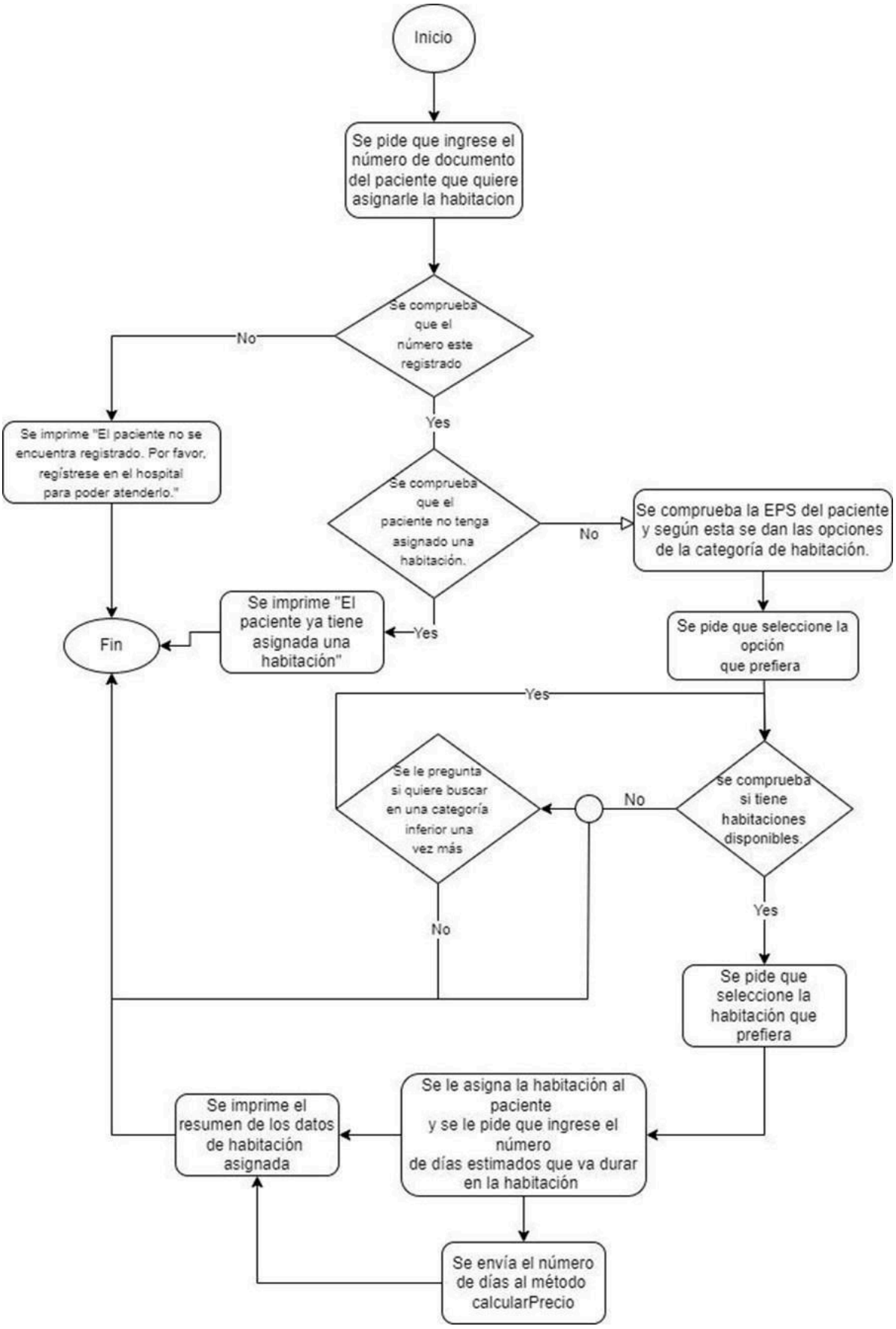
Funcionalidad 3: Asignar Habitaciones

El cuerpo de la funcionalidad se encuentra en el paquete “iuMain” en la clase “Main”.

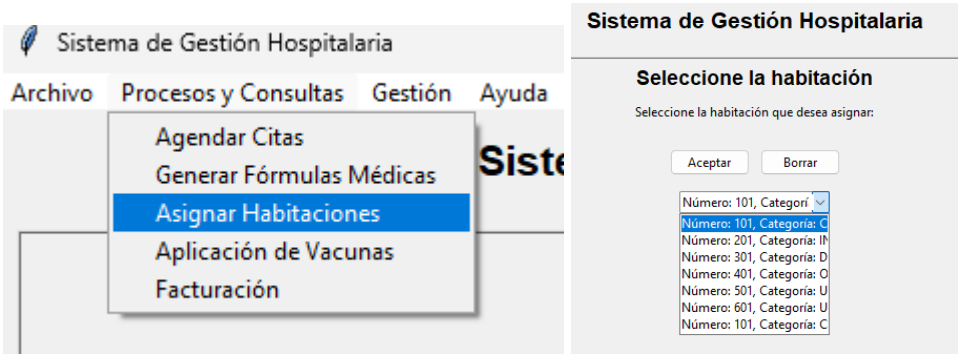
Esta funcionalidad se encarga de que los pacientes registrados tengan la posibilidad de seleccionar entre categorías específicas de habitaciones, según su afiliación a la EPS. Cada categoría ofrece una variedad y selección única de habitaciones, permitiendo a los pacientes elegir aquella que se ajuste 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:

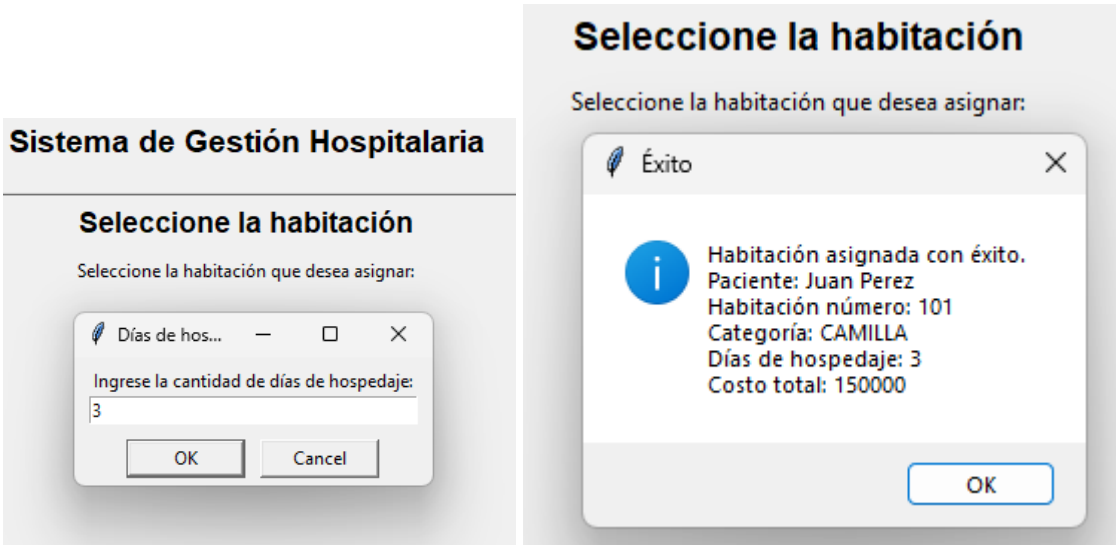
- Hospital: Se encarga de almacenar las listas implicadas en la funcionalidad y además contiene el método buscaPaciente que es primordial para interactuar con los objetos tipo paciente..
- Paciente:Representa a los pacientes registrados en el hospital y está diseñada para permitirles seleccionar categorías 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:


Inicio
Se pide que ingrese el número de documento del paciente que quiere asignarle la habitacion
Se comprueba que el número este registrado
No
Yes
Se imprime "El paciente no se encuentra registrado. Por favor, regístrese en el hospital para poder atenderlo."
Se comprueba que el paciente no tenga asignado una habitación.
No
Se comprueba la EPS del paciente y según esta se dan las opciones de la categoría de habitación.
Yes
Se imprime "El paciente ya tiene asignada una habitación"
Fin
Se pide que seleccione la opción que prefiera
Yes
Se le pregunta si quiere buscar en una categoría inferior una vez más
No
se comprueba si tiene habitaciones disponibles.
No
Yes
Se pide que seleccione la habitación que prefiera
Se le asigna la habitación al paciente y se le pide que ingrese el número de días estimados que va durar en la habitación
Se imprime el resumen de los datos de habitación asignada
Se envía el número de días al método calcularPrecio

Ejecución de la funcionalidad:





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”.

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.

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.

Funcionalidad 4: Aplicación de vacunas

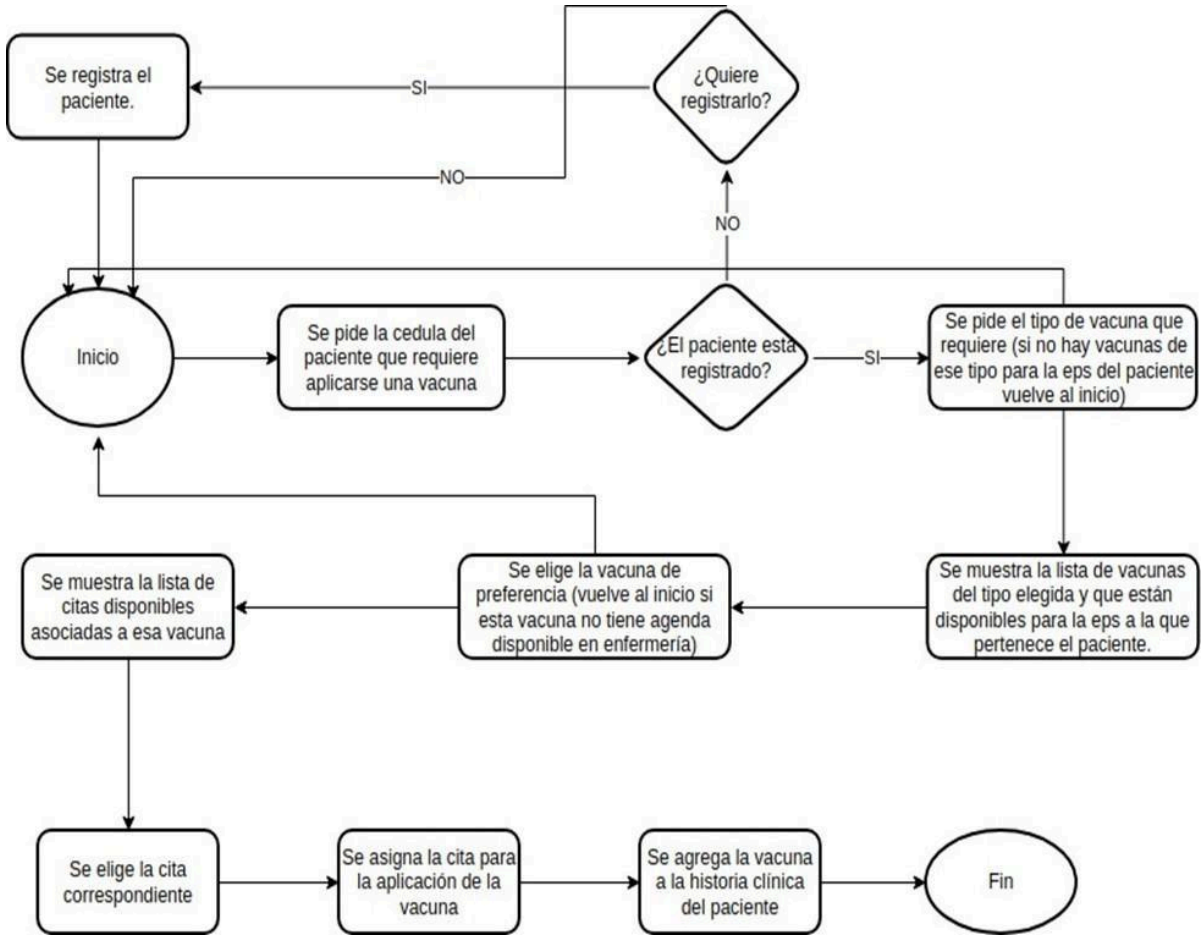
El cuerpo de la funcionalidad se encuentra en el paquete “iuMain” en la clase “Main”.

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.

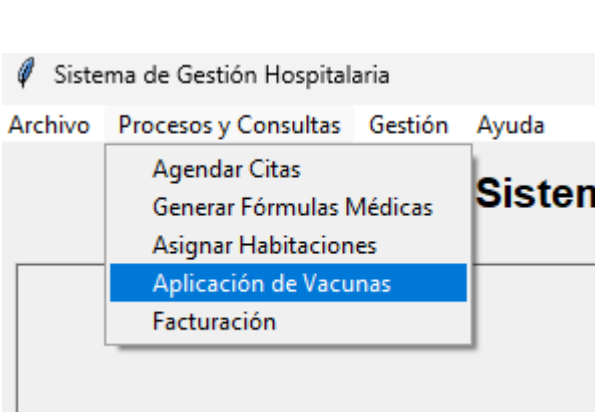
Durante la ejecución de esta funcionalidad intervienen las siguientes clases:

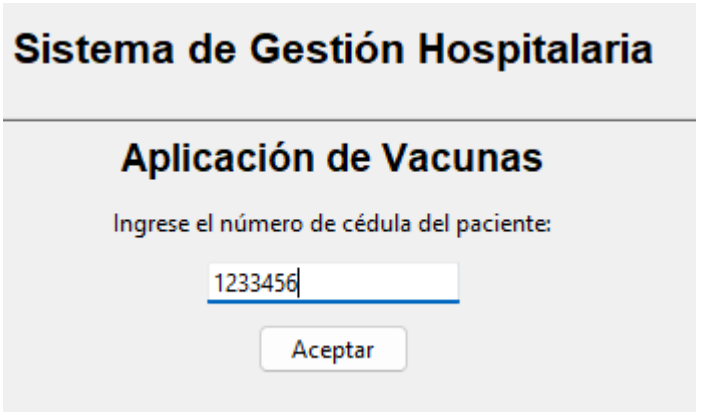
- Hospital: de esta se sacarán las listas de pacientes y vacunas para filtrarlas.
- Paciente: en esta se filtraran las vacunas de acuerdo con el tipo de eps del paciente.
- Vacuna: de esta se sacará la agenda disponible para el usuario.
- CitaVacuna: hace referencia a cada una de las citas que están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

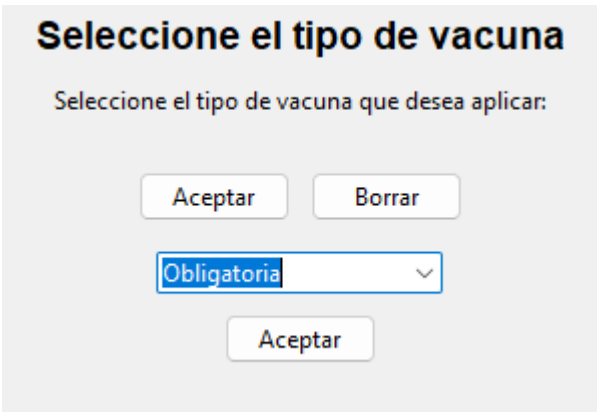
Secuencia de la funcionalidad:

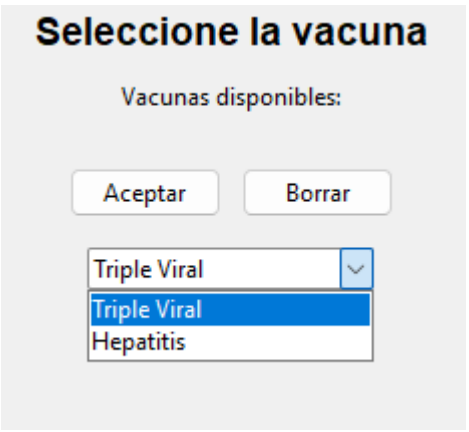


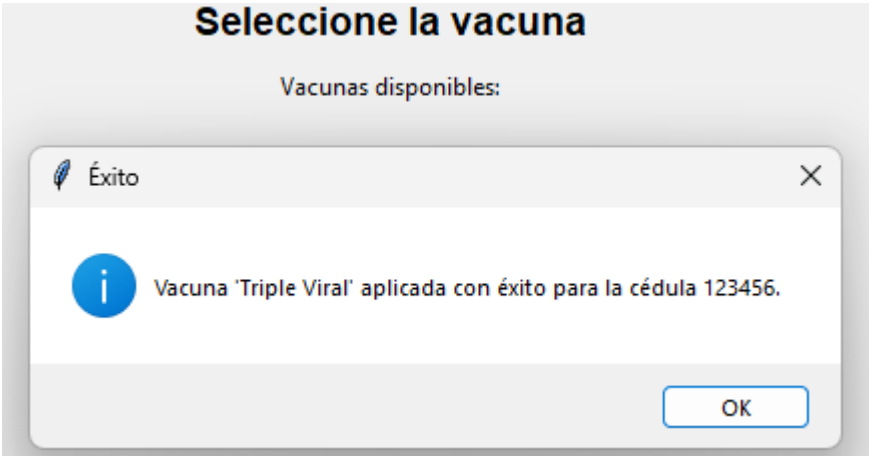
Ejecución de la funcionalidad:











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.

- 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.

- 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.

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

El cuerpo de la funcionalidad se encuentra en el paquete “iuMain” en la clase “Main”.

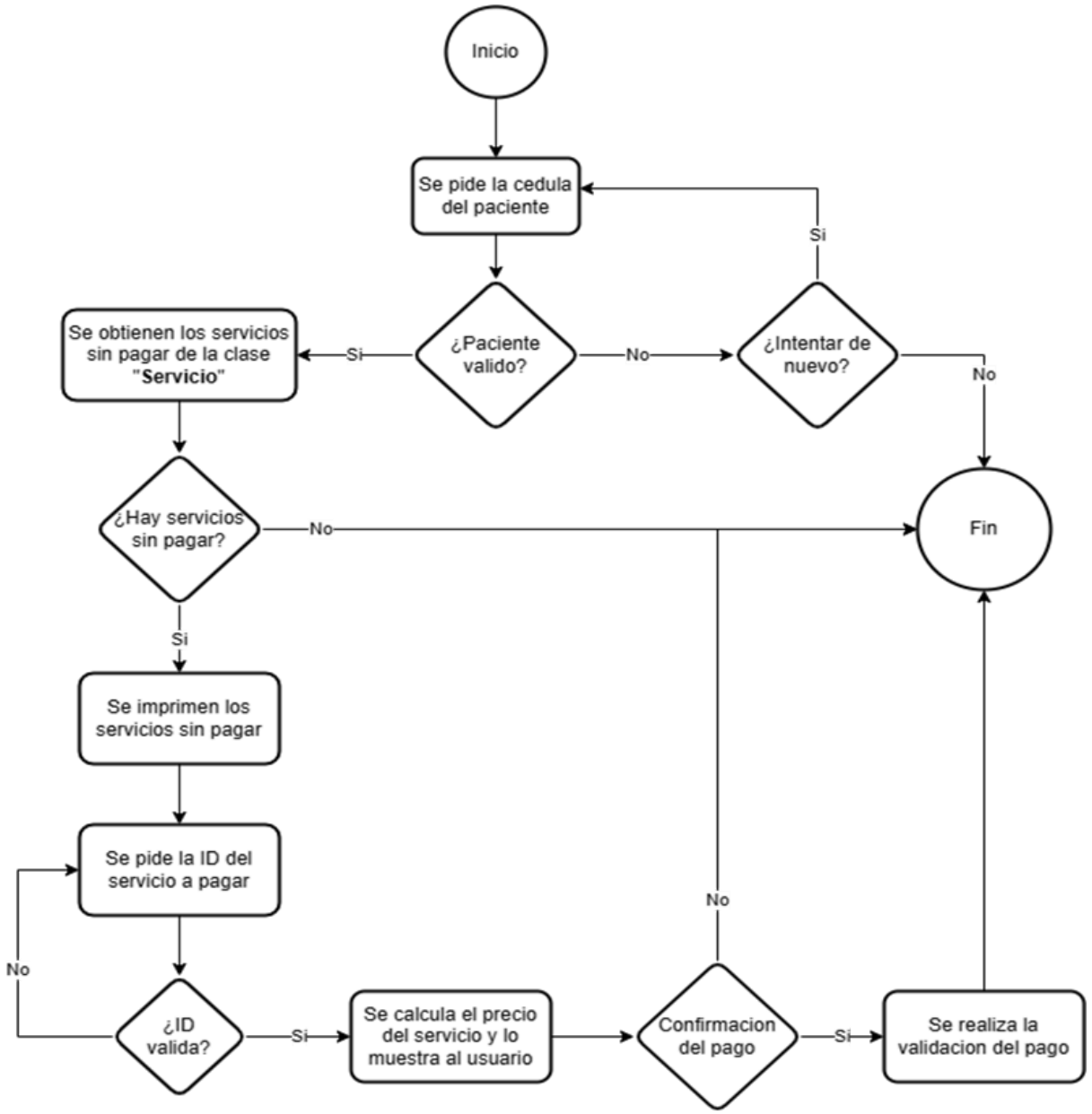
Esta funcionalidad realiza la facturación de servicios médicos para un paciente, buscando al paciente, mostrando los servicios pendientes de pago, permitiendo al usuario seleccionar y pagar un servicio específico, y registrando el pago en el

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.

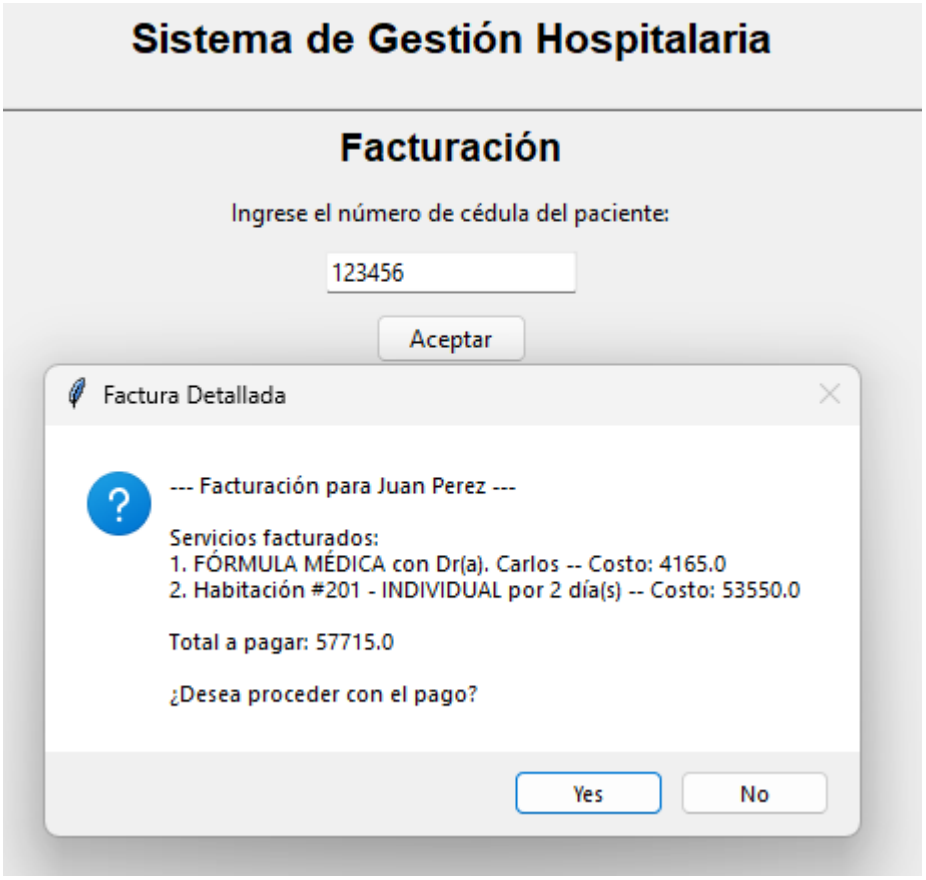
8. Una vez que se ha seleccionado un servicio válido, se calcula el precio del servicio utilizando el método calcularPrecio del objeto pacienteSeleccionado. El cálculo del precio depende del tipo de servicio seleccionado.

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.

Excepciones:

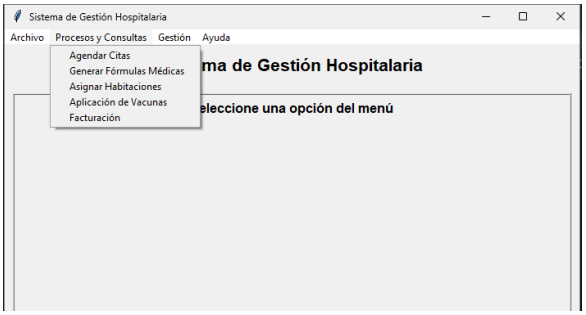
IV. Descripcion de interfaces graficas

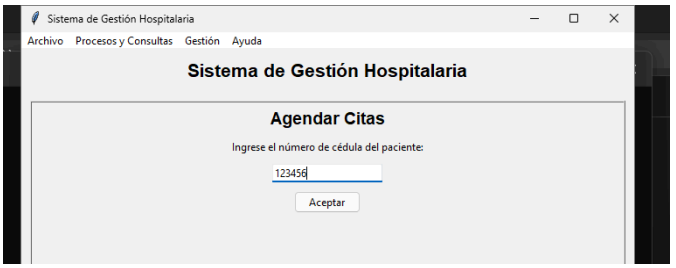
Ventana de inicio

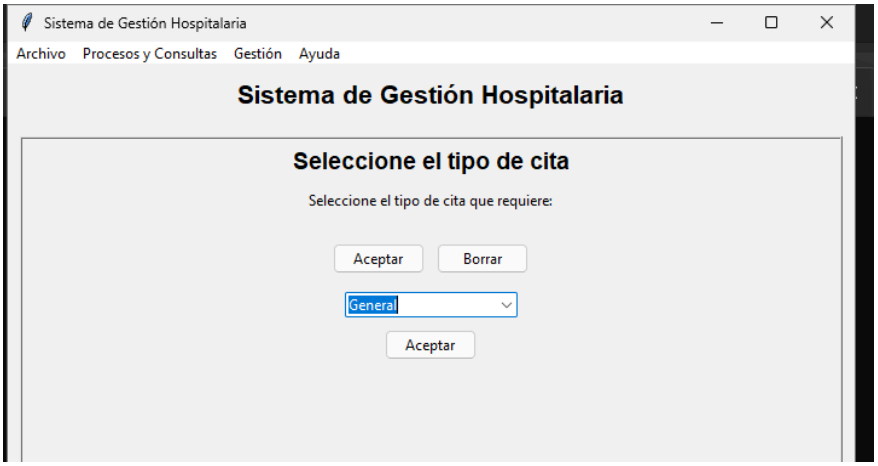
La Ventana de Inicio se desarrolla utilizando una Tk() principal, donde se organizan los componentes mediante Frames anidados. El mensaje de bienvenida y la información de los desarrolladores se implementan con Label(), asegurando un diseño limpio y estructurado. Las imágenes se gestionan con PhotoImage() y Label(), permitiendo su actualización mediante eventos de bind(), como <Enter> y <Leave>, para cambiar dinámicamente la imagen cuando el usuario pasa el mouse sobre ellas.

El botón de ingreso está construido con Button(), y su comando está vinculado a una función que oculta la ventana de inicio y muestra la ventana principal. Se puede manejar esto con withdraw() para ocultar la ventana inicial o destroy() si no se necesita mantenerla en memoria. Además, la disposición de los elementos se optimiza con grid() o place(), asegurando que la interfaz mantenga su estructura independientemente del tamaño de la pantalla.

Ventana principal de Usuario







La Ventana Principal del Usuario está construida utilizando el módulo tkinter y organiza sus elementos mediante Frames y un sistema de menú basado en Menu(). El menú principal contiene opciones como Archivo, Procesos y Consultas, Gestión y Ayuda, cada una de las cuales puede activar diferentes funciones dentro del sistema. Estas opciones están vinculadas a command, lo que permite ejecutar funciones específicas cuando el usuario selecciona una opción. Además, la estructura interna de la ventana utiliza grid() o pack() para distribuir los elementos de manera ordenada.

El área de trabajo principal está implementada con un Frame, el cual es dinámico y se actualiza según la opción seleccionada en el menú. Para lograr esta actualización, se emplea la técnica de destruir y recrear widgets (widget.destroy()) o bien el uso de tkraise() sobre Frames predefinidos. Además, se configura grid_columnconfigure() y grid_rowconfigure() para permitir el redimensionamiento automático del área de trabajo, asegurando que los elementos se ajusten correctamente si el usuario modifica el tamaño de la ventana.

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:

Cédula: 123456
Nombre: Juan
Perez EPS:
Subsidiado
Enfermedades: Gripe, Hipertension

Cédula: 234567
Nombre: Maria Gomez
EPS: Contributivo
Enfermedades: Miopia, Conjuntivitis

Cédula: 345678
Nombre: Luis
Rodriguez EPS:
Particular
Enfermedades:
Caries

Cédula: 456789
Nombre: Ana
Lopez EPS:
Contributivo
Enfermedades: Gingivitis

Doctores:

Cédula:
654321
Nombre:
Carlos
Tipo de EPS en que trabaja: Subsidiado
Especialización: General

Cédula:
765432
Nombre:
Laura
Tipo de EPS en que trabaja: Contributivo
Especialización: Odontologia

Cédula:
876543
Nombre:
Andres
Tipo de EPS en que trabaja: Particular
Especialización: Oftalmologia

Enferme
dades

Nombre:
Gripe

- Especialidad:

General Nombre:

Hipertensión

- Especialidad: General

Nombre: Caries

- Especialidad:

Odontólogo Nombre:

Gingivitis

- Especialidad:

Odontólogo Nombre:

Miopía

- Especialidad:

Oftalmólogo Nombre:

Conjuntivitis

- Especialidad: Oftalmólogo

Medicamentos:

Paracetamol

- Nombre (String): Paracetamol
- Descripción (String): Para la fiebre y dolores
- Cantidad (Integer): 25
- Precio (Integer):

5000 Lisinopril

- Nombre (String): Lisinopril
- Descripción (String): Inhibidor de la ECA
- Cantidad (Integer): 10
- Precio (Integer):

15000 Fluoruro

- Nombre (String): Fluoruro
- Descripción (String): Aplicación tópica para fortalecer los dientes
- Cantidad (Integer): 30
- Precio (Integer):

30000 Clorhexidina

- Nombre (String): Clorhexidina
- Descripción (String): Enjuague bucal antibacteriano
- Cantidad (Integer): 15
- Precio (Integer): 20000

Cirugía LASIK

- Nombre (String): Cirugía LASIK
- Descripción (String): No es un medicamento, es un tratamiento optativo
- Cantidad (Integer): 2
- Precio (Integer):

5000000 Dextrometorfano

- Nombre (String): Dextrometorfano
- Descripción (String): Para aliviar la irritación ocular
- Cantidad (Integer): 30
- Precio (Integer): 10000

Habitaciones:

Nombre:
CAMILLA

- Número: 101
- Ocupada: False
- Paciente

: null Nombre:

INDIVIDUAL

- Número: 201
- Ocupada: False
- Pacient

e: null

Nombre:

DOBLE

- Número: 301
- Ocupada: False
- Paciente:

null Nombre:

OBSERVACION

- Número: 401
- Ocupada: False
- Pacient

e: null

Nombre: UCI

- Número

:501

- Ocupada: False
- Pacient

e:nul

Nombre:Ucc

- Número: 601
- Ocupada: False
- Paciente : null

Vacunas

Nombre: Triple Viral

- Carácter: Obligatoria
- Cobertura: Subsidiada, Contributiva, Particular
- Precio:

50000

Nombre:

Influenza

- Carácter: No Obligatoria
- Cobertura: Subsidiada, Contributiva
- Precio:

40000

Nombre:

Hepatitis

- Carácter: Obligatoria
- Cobertura: Subsidiada, Contributiva
- Precio: 60000

Nombre: Virus del Papiloma Humano (VPH)

- Carácter: No Obligatoria
- Cobertura: Subsidiada, Contributiva, Particular
- Precio: 200000

¿Cómo funciona la aplicación?

Al correr inicialmente la aplicacion se da una breve introduccion al usuario acerca de las aplicaciones que se le pueden dar al programa y las tareas que se pueden facilitar al hacer uso de este

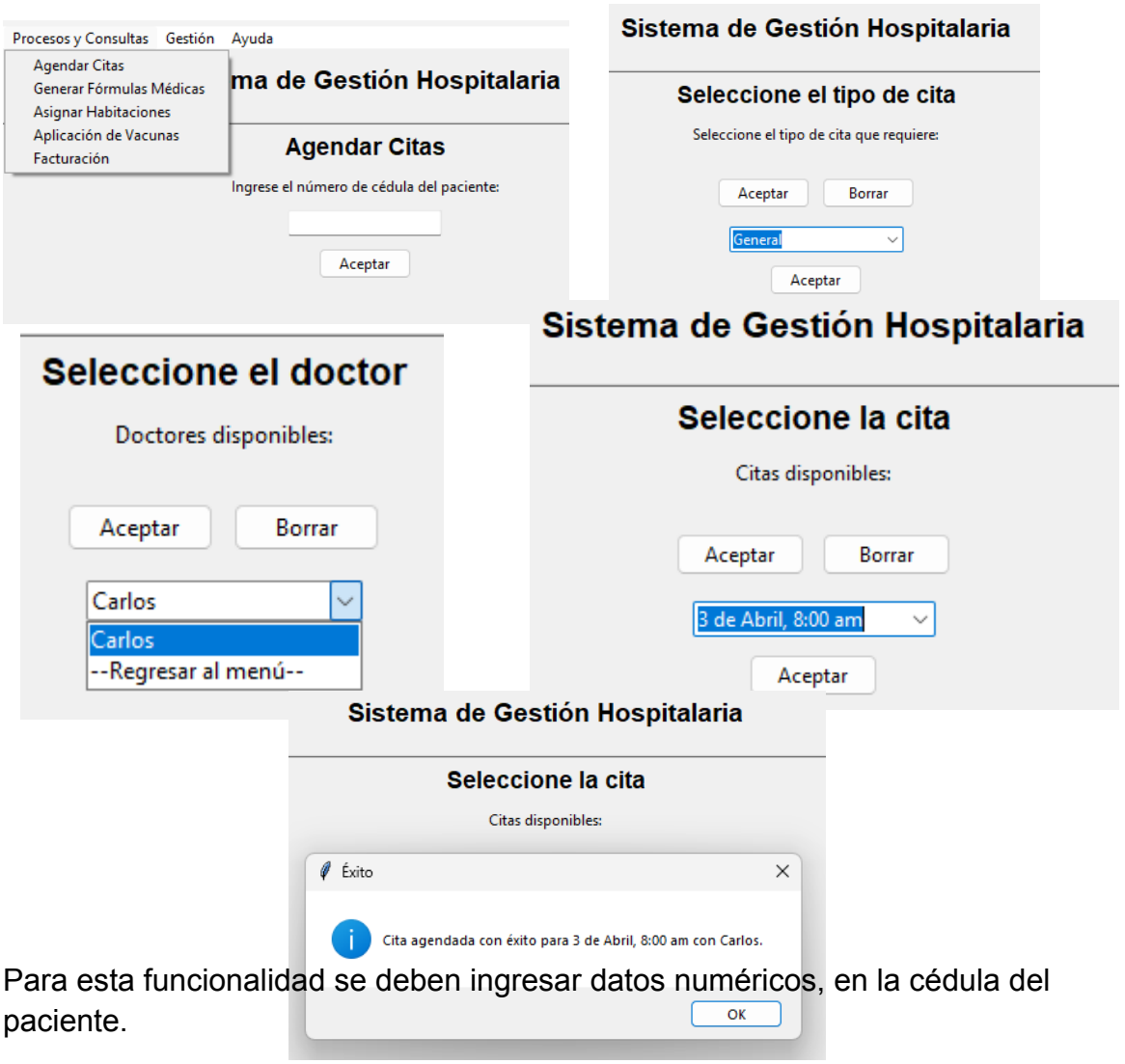
En esta aplicación a pesar de que hay unos datos precargados, se tiene la libertad de crear objetos solamente ingresando datos por teclado, gracias al

apartado de gestión que podemos escoger nada más iniciar la aplicación.

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, en la cédula del paciente.

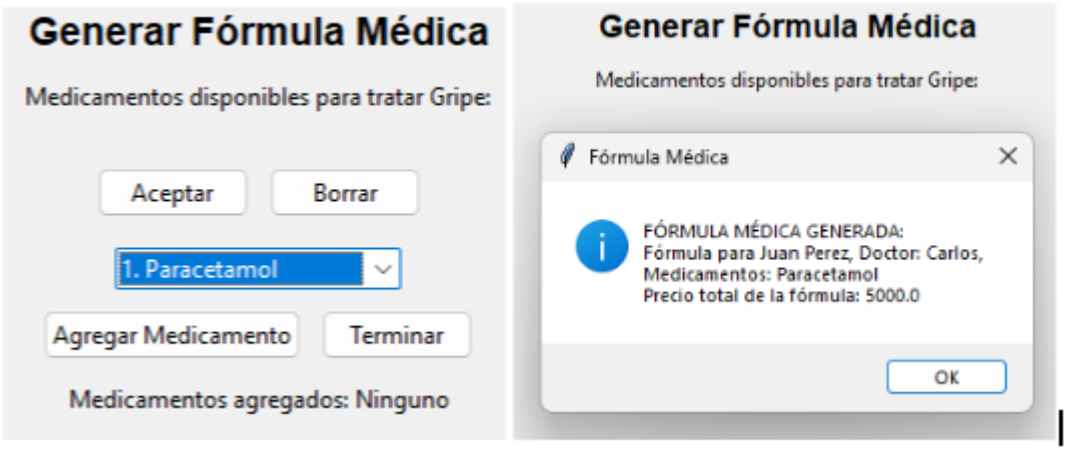
Hay que tener en cuenta que según el tipo de eps del paciente, sólo se mostrarán los doctores que sean del mismo tipo de eps del paciente en cada categoría, en este caso, Juan al tener tipo de eps Subsidiado, y seleccionar general, solo se muestra el doctor Carlos, que es el único General con tipo de eps Subsidiado que hay en los datos precargados.

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.

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.



- 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 selecciona con click.

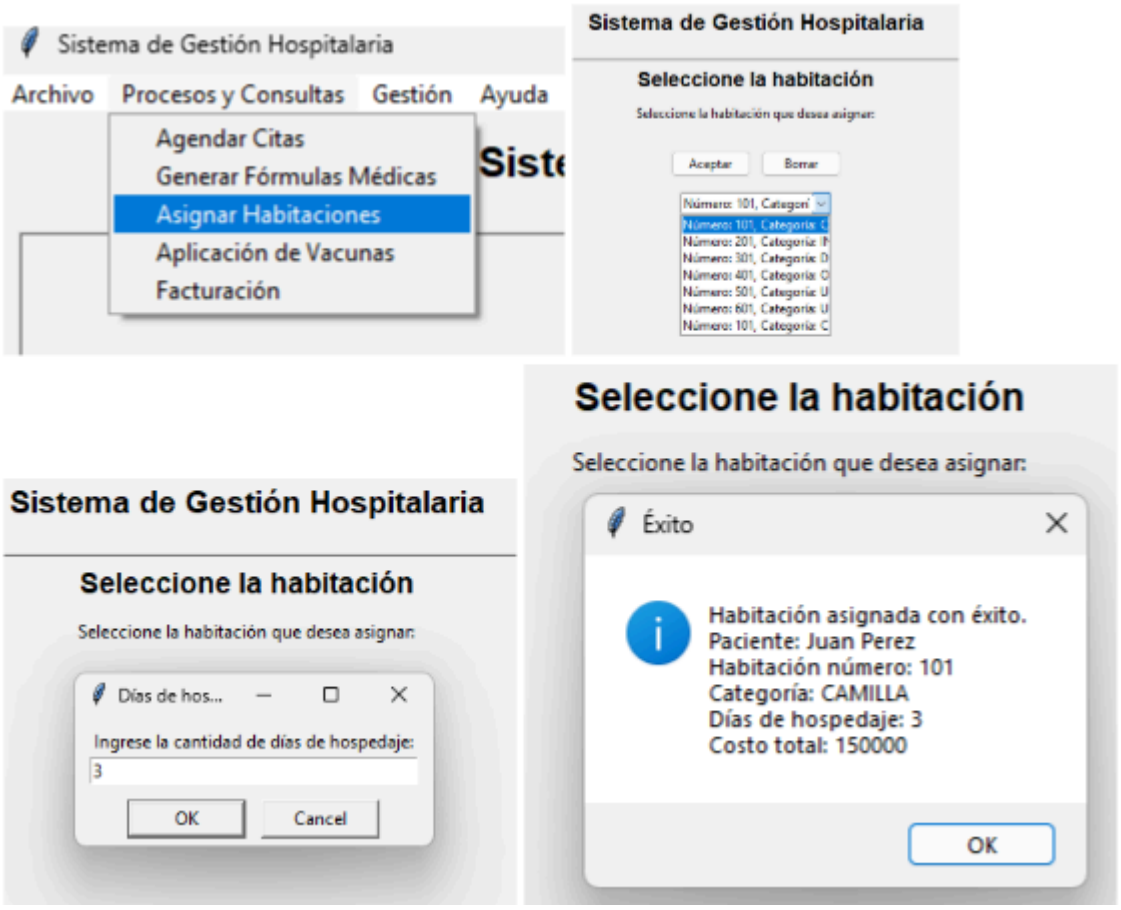
- Al seleccionar la enfermedad nos retornará con los doctores que han atendido al paciente y que tenga la misma especialidad que trata la enfermedad (para que la funcionalidad sirva este paciente deberá tener citas en su historial de citas con algún doctor que tenga la misma especialidad que trata las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

- Ahora se nos imprime la lista actual de medicamentos y se nos pregunta si deseamos agregar otro, si es asi vamos y volvemos a elegir cuál agregar

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

Ahí termina la funcionalidad

3. Funcionalidad de asignar habitación:

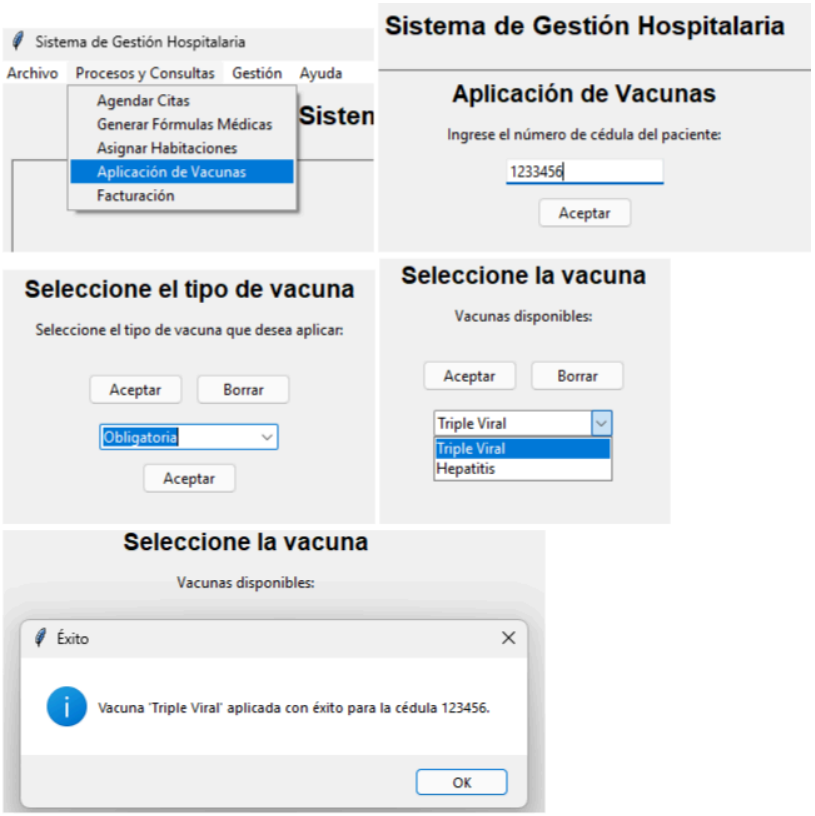


Al comienzo de ejecutar nuestro proyecto al usuario le pedira la cedula, posteriormente aparecerá un menú inicial que consta de las habitaciones disponibles, elegimos una.

Ya después de haber seleccionado la opción nos aparecerá un nuevo menú que consta de una ventana que nos pide ingresar la cantidad de dias.

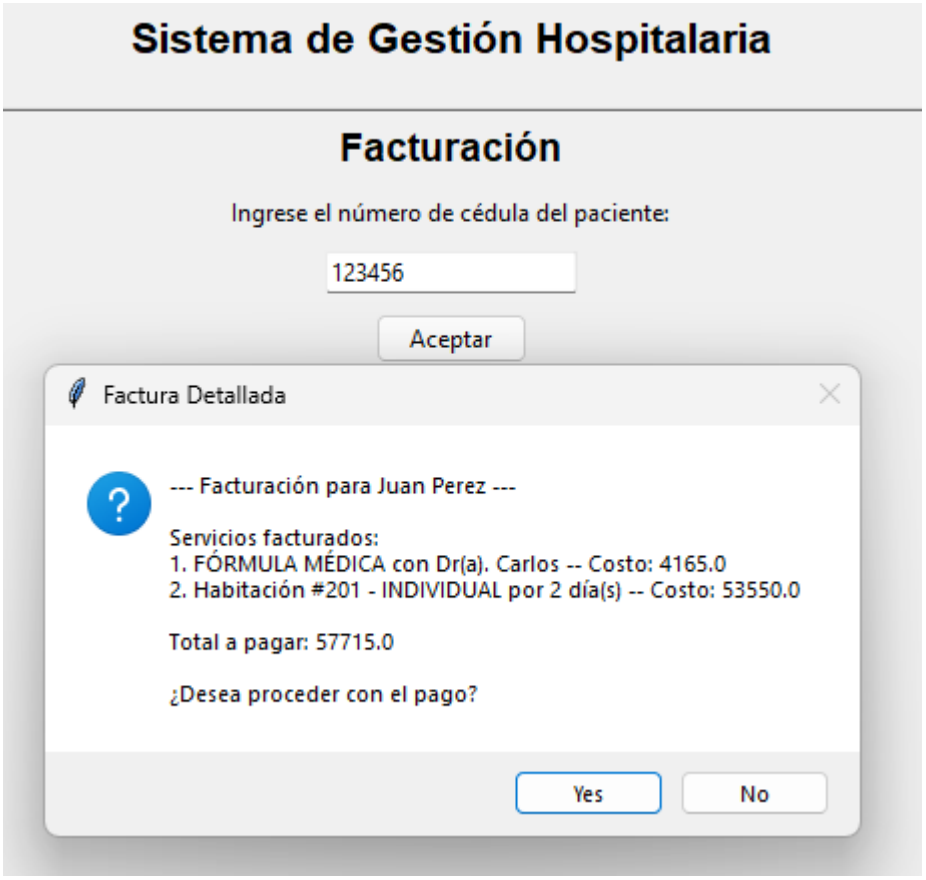
Finalmente nos imprime el resumen de la habitación y el paciente y ya termina la funcionalidad, es importante decir que cada habitación también almacena el paciente que la escogió y viceversa es decir que si nosotros queremos escoger una habitación nuevamente y ingresamos la cédula de algún paciente 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:



Se pide numero de cedula, se elige que tipologia de vacuna se necesita, y se muestran las opciones disponibles. finalmente se muestra la aplicacion de la vacuna con un mensaje emergente.

5. Facturacion



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 “123456”
del paciente Juan.

A continuación se le mostraran los servicios del paciente que estén pendientes por pagar.

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.

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.