Pontificia Universidad Católica del Ecuador - Sede Ambato

Carrera: Ingeniería en Sistemas de la Información

Asignatura: Programación VI

Docente: Ing. Edison Meneses Mg.

Estudiante: Joaquín Bermeo

Tema: Programación Orientada a Objetos – Sistema de Consultas Médicas

Fecha de Entrega: 01/05/2025

Introducción:

1. Objetivo:

Este proyecto tiene como objetivo diseñar un sistema que permita facilitar el registro de pacientes y consultas médicas usando la "Programación Orientada a Objetos (POO)", dicha información y datos podrán ser consultados dentro del sistema ya que se irán almacenando. Este sistema fue diseñado para permitir que el guardado y manejo de información sea de manera ordenada y se pueda acceder fácilmente a la misma gracias a búsqueda individual o completa de pacientes.

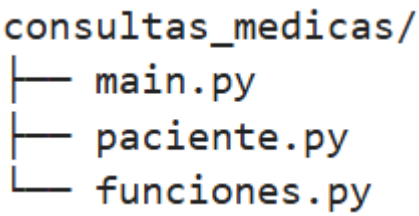
---

2. Contexto:

El sistema fue diseñado de manera modular, es decir, se usaron varios archivos que trabajan en conjunto para que el programa funcione, esto permite que la organización y optimización del código se lleve de manera adecuada y sea más entendible a diferencia de manejar un solo archivo "main". El proyecto lo que busca es utilizar los conceptos básicos y fundamentales de la Programación Orientada a Objetos como el uso de clases, objetos, métodos/funciones, etc., para lograr cumplir el objetivo principal.

Desarrollo:

1. Estructura del proyecto:



Archivo “main.py”:

Este archivo contiene el menú principal y la lógica que gestiona la interacción con el usuario. El programa se mantiene activo mediante el uso de un ciclo *while*, lo que permite que el usuario interactue entre las distintas opciones que se manejan dentro del sistema sin la necesidad de ejecutarlo siempre.

Archivo “paciente.py”:

Aquí se define la clase “Paciente” la cual contiene los atributos y métodos relacionados a la información de un paciente. Los atributos implementados fueron: nombre, cédula, edad, tipo de sangre y consultas (lista de diccionarios), esta última permite que a un paciente se le pueda registrar varias consultas y se almacenen. Por otro lado, se implementaron métodos que trabajan en conjunto con el archivo de funciones “agregar_consulta”, mostrar_información y mostrar_consultas.

Archivo “funciones.py”:

Contiene funciones auxiliares de los distintos procesos del sistema como registrar pacientes consultas, buscar pacientes por su cédula y mostrar información de los pacientes registrados. Al inicio se importa la clase Paciente del archivo “paciente.py”. Cada una de estas funciones cuentan con sus respectivas validaciones para que el programa se ejecute y utilice sin que haya errores. Además de las funciones, aquí se inicializa la lista que se encarga de ir almacenando todos los pacientes que se vayan registrando “lista_pacientes”.

---

2. Diseño de la clase “Paciente”:

La clase “Paciente” es fundamental dentro del funcionamiento del sistema, ya que representa el objeto con el que interactúa el usuario. Los atributos que contiene la clase son los siguientes:

- **nombre:** Atributo que almacena el nombre del paciente.
- **cédula:** Atributo que almacena la cédula del paciente.
- **edad:** Atributo que almacena la edad del paciente.
- **tipo_sangre:** Atributo que almacena el tipo de sangre del paciente.
- **lista_consultas:** Lista de diccionarios vacía que almacena las consultas del paciente.

```python
# Definición de la clase Paciente
class Paciente:
    # Constructor de la clase, inicializa los atributos del paciente
    def __init__(self, nombre, cedula, edad, tipo_sangre):
        self.nombre = nombre          # Atributo que almacena el nombre del paciente
        self.cedula = cedula          # Atributo que almacena la cédula del paciente
        self.edad = edad              # Atributo que almacena la edad del paciente
        self.tipo_sangre = tipo_sangre  # Atributo que almacena el tipo de sangre del paciente
        self.lista_consultas = []     # Inicializa una lista vacia para almacenar las consultas del paciente
```

Los métodos que se implementaron dentro de a clase “Paciente” son:

- **agregar_consulta:** Método que recibe como parámetros a fecha, diagnostico y tratamiento con los que se crea un diccionario que después es almacenado en la lista “lista_consultas”.

```python
# Método para agregar una nueva consulta al historial del paciente
    def agregar_consulta(self, fecha, diagnostico, tratamiento):
        consulta = {  # Crea un diccionario con los datos de la consulta
            "fecha": fecha,
            "diagnostico": diagnostico,
            "tratamiento": tratamiento
        }
        self.lista_consultas.append(consulta)  # Agrega la consulta a la lista de consultas del paciente

```

- **mostrar_información:** Método que se encarga de mostrar la información básica del paciente, es decir, su nombre, cédula, edad, y tipo de sangre.

```python
# Método para mostrar la información básica del paciente
    def mostrar_informacion(self):
        print('\n>>> DATOS <<<')
        # Muestra los datos básicos del paciente (nombre, cédula, edad, tipo de sangre)
        print(f'Nombre: {self.nombre}\nCedula: {self.cedula}\nEdad: {self.edad}\nTipo de Sangre: {self.tipo_sangre}')

```

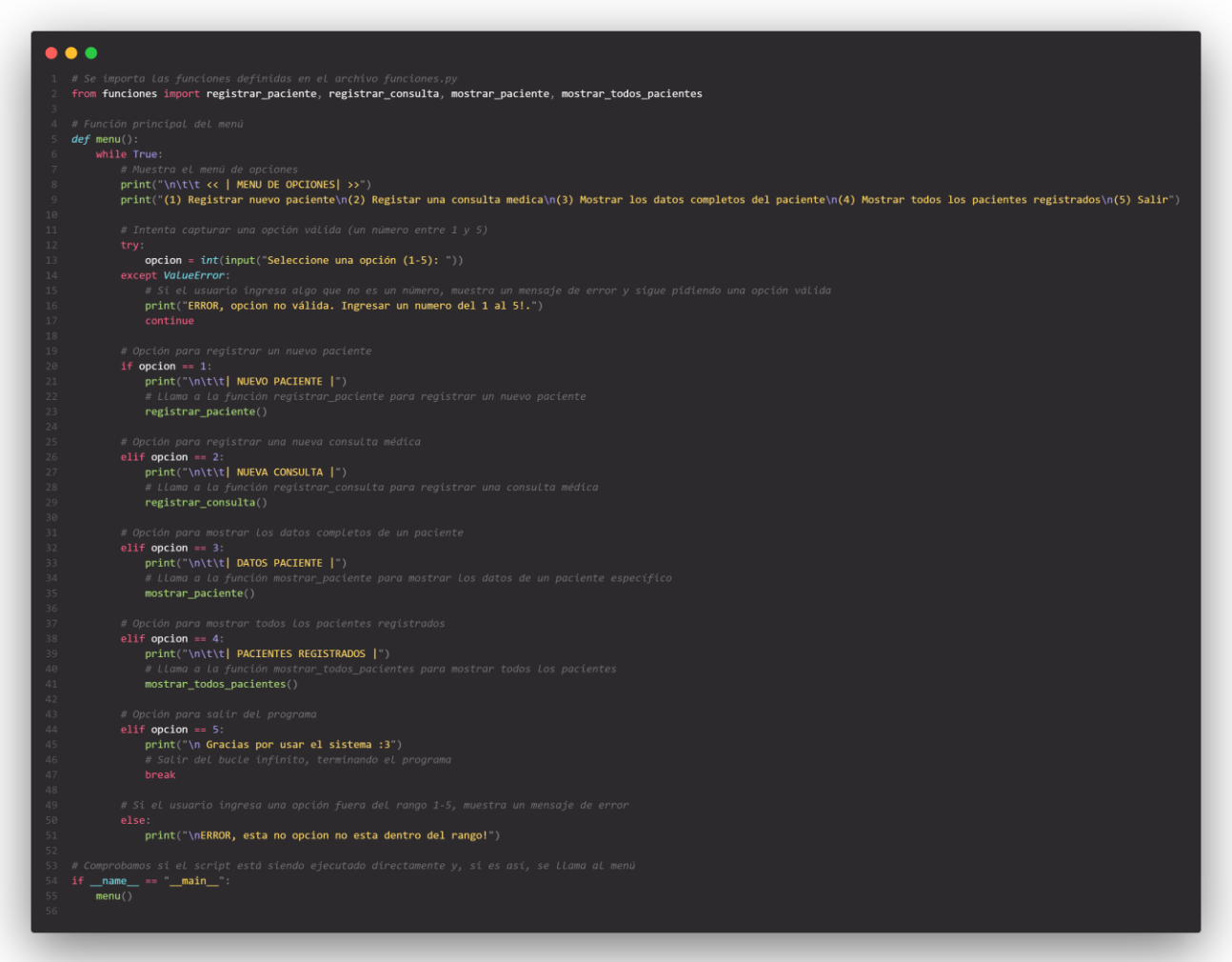
- **mostrar_consultas:** Método que muestra el historial de consultas del paciente, es decir, imprime cada uno de los elementos que han sido registrados en la lista “lista_consultas”, si esta se encuentra vacía se imprime un aviso.

```python
# Método para mostrar el historial de consultas del paciente
    def mostrar_consultas(self):
        contador_mostrar_inf = 1  # Contador para numerar las consultas
        print("\n>>> Historial de Consultas <<<")

        if self.lista_consultas:  # Si el paciente tiene consultas registradas
            # Itera sobre todas las consultas y las imprime
            for consulta in self.lista_consultas:
                print(f'\nConsulta #{contador_mostrar_inf}:')
                # Muestra la información de la consulta: fecha, diagnóstico y tratamiento
                print(f'Fecha: {consulta["fecha"]}\nDiagnostico: {consulta["diagnostico"]}\nTratamiento: {consulta["tratamiento"]}')
                contador_mostrar_inf += 1  # Aumenta el contador de consultas
        else:
            print("AVISO! No existen registros.")  # Si no hay consultas, muestra un mensaje

```

3. **Funcionamiento del menú y funciones:**

Menú:

Se creó una función llamada menu() que está dentro del archivo “main”. Dentro de la función se maneja un ciclo “while True:” que permite que el sistema se mantenga activo hasta que el usuario desee salir. Se lee una única entrada de un número entero que es la opción que ingresa el usuario para ir navegando entre las distintas opciones (1. Registrar nuevo paciente 2. Registrar una consulta médica 3. Mostrar datos completos de un paciente 4. Mostrar todos los pacientes registrados 5. Salir), este proceso de selección se lo realizó mediante el uso de if, elif y else donde se compara el valor ingresado por el usuario y permite acceder a las funciones principales del sistema que se exportan del archivo funciones.py. Además, se aplicó la respectiva validación con un try para que el sistema no colapse cuando se ingrese caracteres que no sean números.



```python
# Se importa las funciones definidas en el archivo funciones.py
from funciones import registrar_paciente, registrar_consulta, mostrar_paciente, mostrar_todos_pacientes

# Función principal del menú
def menu():
    while True:
        # Muestra el menú de opciones
        print("\n\t\t << | MENU DE OPCIONES| >>")
        print("(1) Registrar nuevo paciente\n(2) Registar una consulta medica\n(3) Mostrar los datos completos del paciente\n(4) Mostrar todos los pacientes registrados\n(5) Salir")

        # Intenta capturar una opción válida (un número entre 1 y 5)
        try:
            opcion = int(input("Seleccione una opción (1-5): "))
        except ValueError:
            # Si el usuario ingresa algo que no es un número, muestra un mensaje de error y sigue pidiendo una opción válida
            print("ERROR, opcion no válida. Ingresar un numero del 1 al 5!.")
            continue

        # Opción para registrar un nuevo paciente
        if opcion == 1:
            print("\n\t\t| NUEVO PACIENTE |")
            # Llama a la función registrar_paciente para registrar un nuevo paciente
            registrar_paciente()

        # Opción para registrar una nueva consulta médica
        elif opcion == 2:
            print("\n\t\t| NUEVA CONSULTA |")
            # Llama a la función registrar_consulta para registrar una consulta médica
            registrar_consulta()

        # Opción para mostrar los datos completos de un paciente
        elif opcion == 3:
            print("\n\t\t| DATOS PACIENTE |")
            # Llama a la función mostrar_paciente para mostrar los datos de un paciente especifico
            mostrar_paciente()

        # Opción para mostrar todos los pacientes registrados
        elif opcion == 4:
            print("\n\t\t| PACIENTES REGISTRADOS |")
            # Llama a la función mostrar_todos_pacientes para mostrar todos los pacientes
            mostrar_todos_pacientes()

        # Opción para salir del programa
        elif opcion == 5:
            print("\n Gracias por usar el sistema :3")
            # Salir del bucle infinito, terminando el programa
            break

        # Si el usuario ingresa una opción fuera del rango 1-5, muestra un mensaje de error
        else:
            print("\nERROR, esta no opcion no esta dentro del rango!")

# Comprobamos si el script está siendo ejecutado directamente y, si es así, se llama al menú
if __name__ == "__main__":
    menu()
```

Funciones:

1. **registrar_paciente():**

Esta función se encarga de capturar la información necesaria para posteriormente crear el objeto “paciente” utilizando la clase creada en el archivo “paciente.py”. En la misma función se valida que el usuario no pueda repetir la cédula en varios pacientes con la implementación de la función “buscar_paciente_cedula(cedula)”, ya que se trata de un identificador único, de igual forma trabaja en conjunto con la función “validar_cedula”. Por otro lado, la edad es validada para que el usuario no pueda ingresar números negativos y que solo pueda ingresar números enteros. Finalmente, se creó una lista con todos los tipos de sangre que existen lo que permite comparar la entrada del usuario con la lista y de esta manera validar que se ingrese un tipo de sangre real.

```python
def registrar_paciente():
    # Lista de tipos de sangre válidos
    tipos_sangre = ["A+","A-","B+","B-","AB+","AB-","O+","O-"]

    # Captura del nombre del paciente
    nombre = input("Nombre: ")
    # Captura de la cédula, asegurándose de que no sea repetida
    cedula = validar_cedula()
    while buscar_paciente_cedula(cedula) != None:  # Verifica que la cédula no esté registrada
        print("ERROR, cédula repetida!")
        cedula = validar_cedula()

    # Captura de la edad, validando que sea un número positivo
    while True:
        try:
            edad = int(input("Edad: "))
            if edad < 0:
                print("ERROR, no puede ingresar una edad negativa!")
                continue
            break
        except ValueError:  # Si se ingresa algo que no sea un número, muestra un error
            print("ERROR, edad no válida!")
            continue

    # Captura del tipo de sangre, validando que sea uno de los tipos permitidos
    tipo_sangre = input("Tipo de sangre (A+, A-, B+, B-, AB+, AB-, O+, O-): ").strip().upper()
    while tipo_sangre not in tipos_sangre:  # Si el tipo de sangre no es válido
        print("ERROR, tipo de sangre no valido!")
        tipo_sangre = input("Tipo de sangre (A+, A-, B+, B-, AB+, AB-, O+, O-): ").strip().upper()

    # Crea un objeto Paciente y lo agrega a la lista de pacientes
    paciente = Paciente(nombre, cedula, edad, tipo_sangre)
    lista_pacientes.append(paciente)
    print(">> Paciente registrado con éxito :).")
```

2. **validar_cedula():**

Como su nombre lo indica, dentro de esta función se pide al usuario ingresar la cédula para validar la parte básica, es decir, que no haya letras y sean 10 dígitos. El dato ingresado se lo

mantiene como tipo String, ya que existen números de cédula que empiezan por el número 0 y al momento de transformarlo a entero no se cuenta el número mencionado y se tomaría como inválida a la cédula ingresada.

```python
# Función para validar la cédula (debe ser un número de 10 dígitos)
def validar_cedula():
    cedula = input("Ingrese la cedula: ")
    while cedula.isdigit() == False or len(cedula) < 10:  # Si la cédula no es válida
        print("ERROR, cedula no valida!")
        cedula = input("Ingrese la cedula: ")
    return cedula
```

3. **buscar_paciente_cedula(cedula):**

Esta función se encarga de buscar a un paciente por su número de cédula dentro de la lista que contiene a todos los pacientes registrados, si existe el paciente retorna al mismo sino retorna None. El único parámetro que recibe es la cédula.

```python
# Función para buscar un paciente por cédula en la lista
def buscar_paciente_cedula(cedula):
    # Recorre la lista de pacientes para encontrar uno con la misma cédula
    for paciente in lista_pacientes:
        if paciente.cedula == cedula:
            return paciente
    return None  # Si no lo encuentra, devuelve None
```

4. **registrar_consulta():**

En esta función se realiza el proceso de registrar consultas de cada paciente, al inicio se valida que por lo menos exista un paciente registrado para poder continuar, en el caso de que no existan pacientes lo único que hará es mostrar una alerta de que no existen pacientes registrados. Luego de la primera validación se pide al usuario ingresar una cedula utilizando la función “validar_cedula” y se guarda en una función, después se busca el paciente con la función “buscar_paciente_cédula(cedula)” y en el caso de que exista se pasará a la siguiente parte, caso contrario se muestra que no se encontró al paciente. Finalmente, se captura la información para generar el diagnostico (fecha, diagnostico y tratamiento). Se utilizó la librería

datetime para que solo se pueda ingresar el formato de fecha DD/MM/AAAA y que tampoco se ingrese fechas futuras.

```python
# Función para registrar una consulta para un paciente
def registrar_consulta():
    if lista_pacientes:  # Si hay pacientes registrados
        cedula = validar_cedula()  # Valida la cédula del paciente
        paciente = buscar_paciente_cedula(cedula)  # Busca el paciente en la lista
        if paciente:  # Si el paciente es encontrado
            # Bucle para validar que la fecha no sea futura
            while True:
                try:
                    fecha = input("Fecha (DD/MM/AAAA): ")
                    fecha_valida = datetime.strptime(fecha, "%d/%m/%Y")  # Intenta convertir la fecha
                    if fecha_valida > datetime.today():  # Si la fecha es futura
                        print("ERROR, La fecha no puede ser futura!")
                    else:
                        break  # Si la fecha es válida, rompe el bucle
                except ValueError:  # Si la fecha no es válida, muestra un error
                    print("ERROR, fecha no valida!")

            # Captura del diagnóstico y tratamiento
            diagnostico = input("Diagnóstico: ")
            tratamiento = input("Tratamiento: ")

            # Llama al método para agregar la consulta al historial del paciente
            paciente.agregar_consulta(fecha, diagnostico, tratamiento)
            print(">> Consulta registrada correctamente :).")
        else:
            print("Paciente no encontrado :(")
    else:
        print("No hay pacientes registrados!")

```

5. **mostrar_paciente():**

Al igual que la función anterior se valida que la lista de pacientes no esté vacía, luego se usa "validar_cedula" para que el usuario ingrese una cédula válida para buscar el paciente con la función "buscar_paciente_cédula(cedula)". Si se encuentra al paciente se muestra la información completa del mismo, es decir, su información básica y todo el historial de consultas utilizando métodos de la clase "Paciente".

```python
def mostrar_paciente():
    if lista_pacientes:  # Si hay pacientes registrados
        cedula = validar_cedula()  # Valida la cédula del paciente
        paciente = buscar_paciente_cedula(cedula)  # Busca el paciente en la lista
        if paciente:  # Si el paciente es encontrado
            paciente.mostrar_informacion()  # Muestra la información básica del paciente
            paciente.mostrar_consultas()  # Muestra el historial de consultas
        else:
            print("Paciente no encontrado :(")
    else:
        print("No hay pacientes registrados!")

```

6. **mostrar_todos_pacientes():**

Esta función también valida que haya al menos un usuario registrado en la lista "lista_pacientes", después se utiliza un for para ir imprimiendo cada uno de los pacientes, pero en este caso solo mostrando su información básica (nombre, cédula, edad y tipo de sangre). Además, se muestra la cantidad total de usuarios registrados y se usa un contador para enumerar a los pacientes y la visualización de la información sea más organizada.

```python
# Función para mostrar todos los pacientes registrados
def mostrar_todos_pacientes():
    contador = 1  # Inicia el contador para mostrar el número de paciente
    if lista_pacientes:  # Si hay pacientes registrados
        print(f'>> Número de pacientes registrados: {len(lista_pacientes)}')
        for paciente in lista_pacientes:  # Recorre todos los pacientes
            print(f'\n| PACIENTE #{contador}|')
            paciente.mostrar_informacion()  # Muestra los datos del paciente
            contador += 1  # Aumenta el contador para el siguiente paciente
    else:
        print("No hay pacientes registrados!")  # Si no hay pacientes, muestra un mensaje

```

Capturas de pantalla de la ejecución:

1. **Menú:**

```
                << | MENU DE OPCIONES| >>
(1) Registrar nuevo paciente
(2) Registar una consulta medica
(3) Mostrar los datos completos del paciente
(4) Mostrar todos los pacientes registrados
(5) Salir
Seleccione una opción (1-5): a
ERROR, opcion no válida. Ingresar un numero del 1 al 5!.

                << | MENU DE OPCIONES| >>
(1) Registrar nuevo paciente
(2) Registar una consulta medica
(3) Mostrar los datos completos del paciente
(4) Mostrar todos los pacientes registrados
(5) Salir
Seleccione una opción (1-5): 1

                | NUEVO PACIENTE |
Nombre:
```

2. **Registrar Pacientes:**

```
                | NUEVO PACIENTE |
Nombre: Joaquin Bermeo
Ingrese la cedula: 123456789a
ERROR, cedula no valida!
Ingrese la cedula: 1804653820
Edad: -1
ERROR, no puede ingresar una edad negativa!
Edad: 18
Tipo de sangre (A+, A-, B+, B-, AB+, AB-, O+, O-): x
ERROR, tipo de sangre no valido!
Tipo de sangre (A+, A-, B+, B-, AB+, AB-, O+, O-): A+
>> Paciente registrado con éxito :).
```

```
                | NUEVO PACIENTE |
Nombre: Pepito
Ingrese la cedula: 1804653820
ERROR, cédula repetida!
Ingrese la cedula: 1802530582
Edad: 20
Tipo de sangre (A+, A-, B+, B-, AB+, AB-, O+, O-): b-
>> Paciente registrado con éxito :).
```

3. Registrar Consulta:

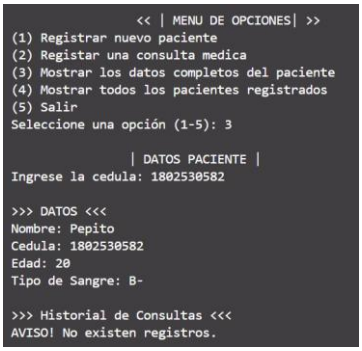
```
                << | MENU DE OPCIONES| >>
(1) Registrar nuevo paciente
(2) Registar una consulta medica
(3) Mostrar los datos completos del paciente
(4) Mostrar todos los pacientes registrados
(5) Salir
Seleccione una opción (1-5): 2

                | NUEVA CONSULTA |
Ingrese la cedula: a
ERROR, cedula no valida!
Ingrese la cedula: 1234567890
Paciente no encontrado :(

                << | MENU DE OPCIONES| >>
(1) Registrar nuevo paciente
(2) Registar una consulta medica
(3) Mostrar los datos completos del paciente
(4) Mostrar todos los pacientes registrados
(5) Salir
Seleccione una opción (1-5): 2

                | NUEVA CONSULTA |
Ingrese la cedula: 1804653820
Fecha (DD/MM/AAAA): 12
ERROR, fecha no valida!
Fecha (DD/MM/AAAA): 12/04/2030
ERROR, La fecha no puede ser futura!
Fecha (DD/MM/AAAA): 12/04/2025
Diagnóstico: Gripe
Tratamiento: Pastillas
>> Consulta registrada correctamente :).
```

```
                | NUEVA CONSULTA |
Ingrese la cedula: 1804653820
Fecha (DD/MM/AAAA): 20/04/2025
Diagnóstico: Fiebre
Tratamiento: Medicamentos
>> Consulta registrada correctamente :).
```

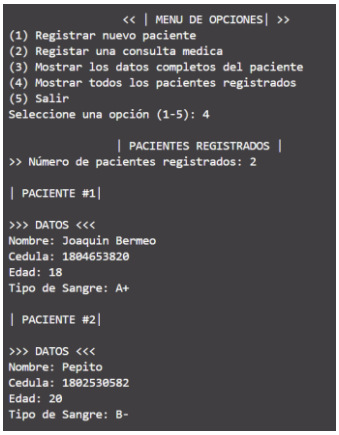
4. Mostrar los datos completos del paciente:

```
                << | MENU DE OPCIONES| >>
(1) Registrar nuevo paciente
(2) Registar una consulta medica
(3) Mostrar los datos completos del paciente
(4) Mostrar todos los pacientes registrados
(5) Salir
Seleccione una opción (1-5): 3

                | DATOS PACIENTE |
Ingrese la cedula: a
ERROR, cedula no valida!
Ingrese la cedula: 123456789a
ERROR, cedula no valida!
Ingrese la cedula: 1234567890
Paciente no encontrado :(
```

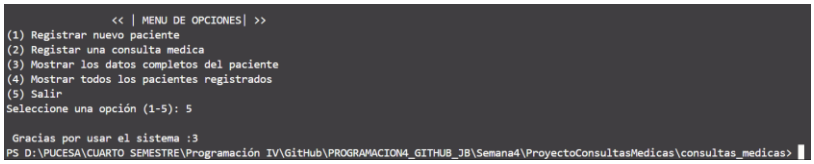
```
                << | MENU DE OPCIONES| >>
(1) Registrar nuevo paciente
(2) Registar una consulta medica
(3) Mostrar los datos completos del paciente
(4) Mostrar todos los pacientes registrados
(5) Salir
Seleccione una opción (1-5): 3

                | DATOS PACIENTE |
Ingrese la cedula: 1804653820

>>> DATOS <<<
Nombre: Joaquin Bermeo
Cedula: 1804653820
Edad: 18
Tipo de Sangre: A+

>>> Historial de Consultas <<<

Consulta #1:
Fecha: 12/04/2025
Diagnostico: Gripe
Tratamiento: Pastillas

Consulta #2:
Fecha: 20/04/2025
Diagnostico: Fiebre
Tratamiento: Medicamentos
```



5. Mostrar todos los pacientes registrados:



6. Salir



Conclusiones

¿Qué aprendí durante el desarrollo del proyecto?

- Este proyecto me permitió aplicar y mejorar los conceptos de la Programación Orientada a Objetos, como el uso de clases, objetos y métodos. Por otro lado, aprendí a estructurar un proyecto de manera modular (separado en varios archivos) y a trabajar de manera mucho más optimizada y organizada.

¿Qué partes fueron más difíciles o interesantes?

- Personalmente una de las partes más difíciles e interesantes fue crear la clase Persona y usar de manera adecuada sus métodos, ya que no es igual que utilizar funciones que no se encuentran dentro de una clase, al inicio es complejo adaptarse, pero después se vuelve interesante porque el código se ve mucho más limpio. Otra de las partes más difíciles fue validar de manera adecuada todos los datos, no por complejidad sino

porque se debe ir analizando a detalle que información no debe estar permitida y la que sí.

Posibles mejoras o nuevas funcionalidades

- Almacenar los datos de los pacientes y consultas en un archivo externo como un ".txt" o ". json" para que la información no sea volátil y se pueda tener una "base de datos".
- Se podría añadir otra opción la cuál permita editar los datos del paciente o directamente eliminar un paciente.
- Validaciones más estrictas de los datos de entrada como por ejemplo que se valide si la cédula es verdadera.