



1 8 0 3

INFORMÁTICA II

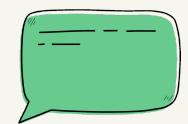
Bioingeniería



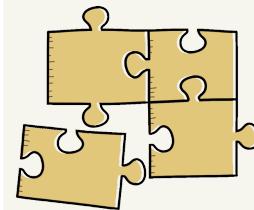
Índice



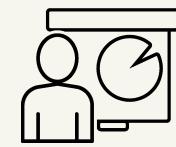
Github



**Para
adelantar..**



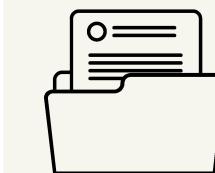
Repasso



Modelo verbal



Ejercicio V2

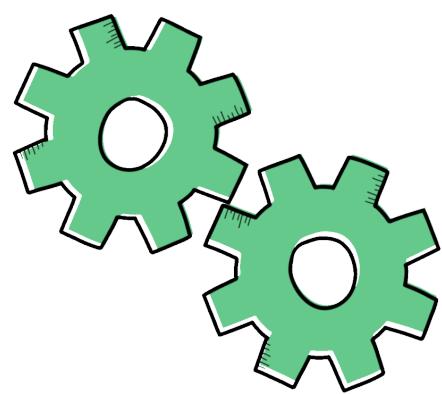


Ejercicio V3



**UNIVERSIDAD
DE ANTIOQUIA**

Facultad de Ingeniería





Visual Studio Code

The screenshot shows the Visual Studio Code interface. On the left, the Extensions Marketplace sidebar is open, displaying a list of extensions. The 'Python' extension by Microsoft is highlighted with a green 'Install' button. The main workspace shows a file named 'serviceWorker.js' with code related to service workers. A code completion dropdown is visible over the 'navigator.serviceWorker' object. The bottom status bar indicates the file is 'master', has 0 changes, and is in 'JavaScript' mode.

File Edit Selection View Go Debug Terminal Help serviceWorker.js - create-react-app - Visual Studio Code - In...

EXTENSIONS: MARKETPLACE

@sortinstalls

- Python 2019.6.24221 54.9M ★ 4.5
Linting, Debugging (multi-threaded, ...
Microsoft [Install](#)
- GitLens — Git sup... 9.8.5 23.1M ★ 5
Supercharge the Git capabilities built...
Eric Amodio [Install](#)
- C/C++ 0.24.0 23M ★ 3.5
C/C++ IntelliSense, debugging, and ...
Microsoft [Install](#)
- ESLint 19.0 21.9M ★ 4.5
Integrates ESLint JavaScript into VS ...
Dirk Baumer [Install](#)
- Debugger for Ch... 4.11.6 20.6M ★ 4
Debug your JavaScript code in the C...
Microsoft [Install](#)
- Language Supp... 0.47.0 18.7M ★ 4.5
Java Linting, Intellisense, formatting, ...
Red Hat [Install](#)
- vscode-icons 8.80 17.2M ★ 5
Icons for Visual Studio Code
VSCode Icons Team [Install](#)
- Vetur 0.21.1 17M ★ 4.5
Vue tooling for VS Code
Pine Wu [Install](#)
- C# 1.21.0 15.6M ★ 4
C# for Visual Studio Code (powered ...
Microsoft [Install](#)

src > JS serviceWorker.js > register > window.addEventListener('load') callback

```
39     checkValidServiceWorker(swUrl, config);
40
41     // Add some additional logging to localhost, p...
42     // service worker/PWA documentation.
43     navigator.serviceWorker.ready.then(() => {
44       product
45       productSub
46       removeSiteSpecificTrackingException
47       removeWebWideTrackingException
48       requestMediaKeySystemAccess
49       sendBeacon
50       serviceWorker (property) Navigator.serviceWork...
51       storage
52       storeSiteSpecificTrackingException
53       storeWebWideTrackingException
54     } userAgent
55   } vendor
56
57   function registerValidSW(swUrl, config) {
58     navigator.serviceWorker
59       .register(swUrl)
60       .then(registration => {
```

TERMINAL 1: node

You can now view `create-react-app` in the browser.

Local: <http://localhost:3000/>
On Your Network: <http://10.211.55.3:3000/>

Note that the development build is not optimized.

Ln 43, Col 19 Spaces: 2 UTF-8 LF JavaScript

master 0 ▲ 0



Github

```
PS C:\Users\veroh\OneDrive\Escritorio\Info 2> cd  
PS C:\Users\veroh\OneDrive\Escritorio\Info 2> git init  
Initialized empty Git repository in C:/Users/veroh/OneDrive/Escritorio/Info 2/.git/  
PS C:\Users\veroh\OneDrive\Escritorio\Info 2> git init Info2_Veronica_Henao_Isaza  
Initialized empty Git repository in C:/Users/veroh/OneDrive/Escritorio/Info 2/Info2_Veronica_Henao_Isaza/.git/
```



The screenshot shows a Visual Studio Code interface with a dark theme. The top bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help menus, and a title bar indicating 'hola.py - Info 2 - Visual Studio Code'. The left sidebar has icons for Explorer, Search, Issues, Pull Requests, and a folder named 'INFO 2' containing a 'hola.py' file. The main editor area displays the following Python code:

```
def saludar():  
    print("Hola")  
saludar()
```





Comandos para recordar Github

Realizar la configuración en tu computador desde la terminal para identificarte

git config --global user.name "John Doe"

git config --global user.email johndoe@example.com

git config --list

Traer(pull) el repositorio desde GitHub

git remote add origin "URL del repositorio"





Github



File Edit Selection View Go Run Terminal Help

hola.py - Info 2 - Visual Studio Code

SOURCE CONTROL

✓ SOURCE CONTROL REPOSITORIES

Info 2... 3m 4 ✓ 0

✓ SOURCE CONTROL

crear repositorio

Publicar Branch

hola.py

Info2_Veronica_Henao_Isaza > hola.py > ...

```
1 def saludar():
2     print("Hola")
3
4 saludar()
```



Github

The screenshot shows the Visual Studio Code interface with a dark theme. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The title bar indicates the file is "hola.py - Info 2 - Visual Studio Code".

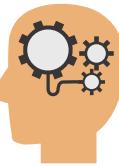
The left sidebar features icons for Source Control, Search, and other development tools. The "SOURCE CONTROL" section displays a repository named "Info 2..." with 0 changes. Below it, there's a "SOURCE CONTROL" section with a "crear repositorio" button and a prominent blue "Publicar Branch" button, which is highlighted with a red rectangle.

The main editor area contains the Python code:

```
1  def saludar():
2      print("Hola")
3
4  saludar()
```

A context menu is open over the code, showing options to "Publish to GitHub private repository" and "Publish to GitHub public repository", both associated with the user "vhenao/Info-2".

A modal dialog box titled "Visual Studio Code" displays an error message: "Can't push refs to remote. Try running 'Pull' first to integrate your changes." It includes three buttons at the bottom: "Abrir registro de GIT" (highlighted with a red rectangle), "Mostrar salida del comando", and "Cancel".



Github

The screenshot shows a GitHub repository page for the user 'YesidORC'. The repository is named 'Info2-2023' and is described as 'Para el curso de informática'. It has 2 forks and was updated 37 minutes ago. The repository is public. Below it is another repository, 'hyperblogFork', which is a fork from 'freddier/hyperblog' and is described as 'Un blog increíble para el curso de Git y Github de Platzi'. It has 11,871 stars and was updated on Sep 29, 2022. The page includes navigation tabs for Overview, Repositories, Projects, Packages, and Stars, along with search and filter options.

Info2-2023 Public
Para el curso de informática

hyperblogFork Public
Forked from freddier/hyperblog
Un blog increíble para el curso de Git y Github de Platzi

Updated 37 minutes ago

Star

Star





Segundo ejercicio de abstracción

Se solicita construir un sistema que permita almacenar datos de pacientes.

Los pacientes tienen información de nombre, cédula, género y servicio, que corresponde al servicio donde están alojados:

El sistema debe permitir tres opciones:

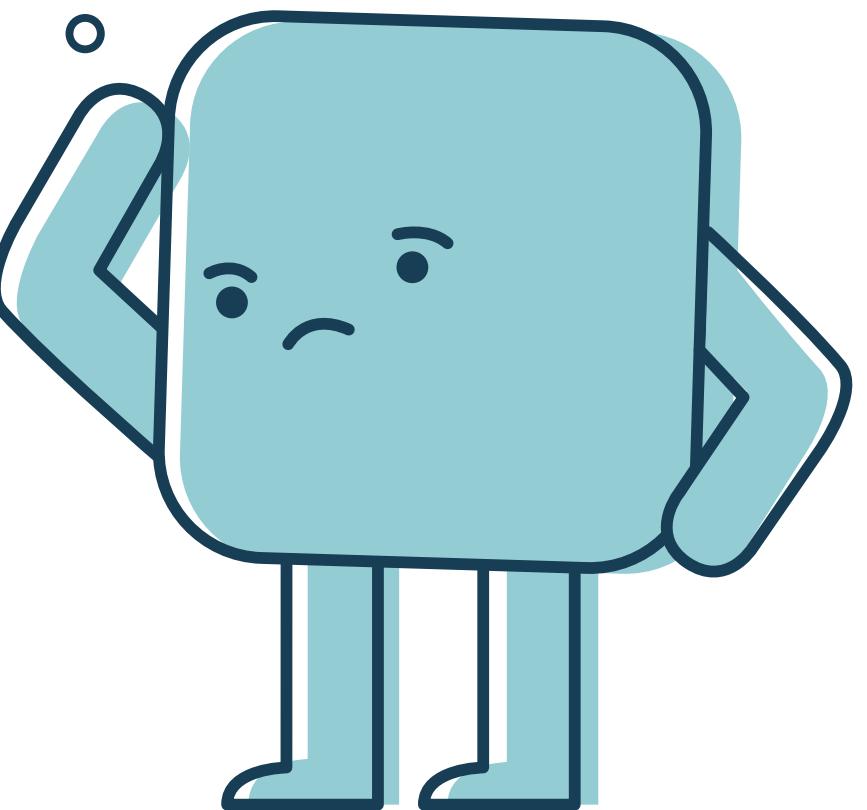
1. Ingresar un paciente Nuevo
2. Ver todos los datos de un paciente existente
3. Ver número de pacientes en el sistema
4. Salir



Segundo ejercicio de abstracción

Antes de empezar a programar, pensemos:

- Número mínimo de clases?
- Atributos y métodos de las clases
- Lógica de funcionamiento





Segundo ejercicio de abstracción

Para el ejercicio anterior podríamos tener:

- **Clases (En singular e iniciando en mayúscula):** Sistema, Paciente.
- **Atributos (en minúscula):** nombre, cédula, género y servicio.
- **Métodos (Funciones a realizar en el sistema):** Ingresar un paciente Nuevo - Ver todos los datos de un paciente existente -Ver número de pacientes en el sistema - Salir
- **Relaciones de agregación:** Donde una de las clases contendrá muchas de la otra, es decir con multiplicidad uno a muchos



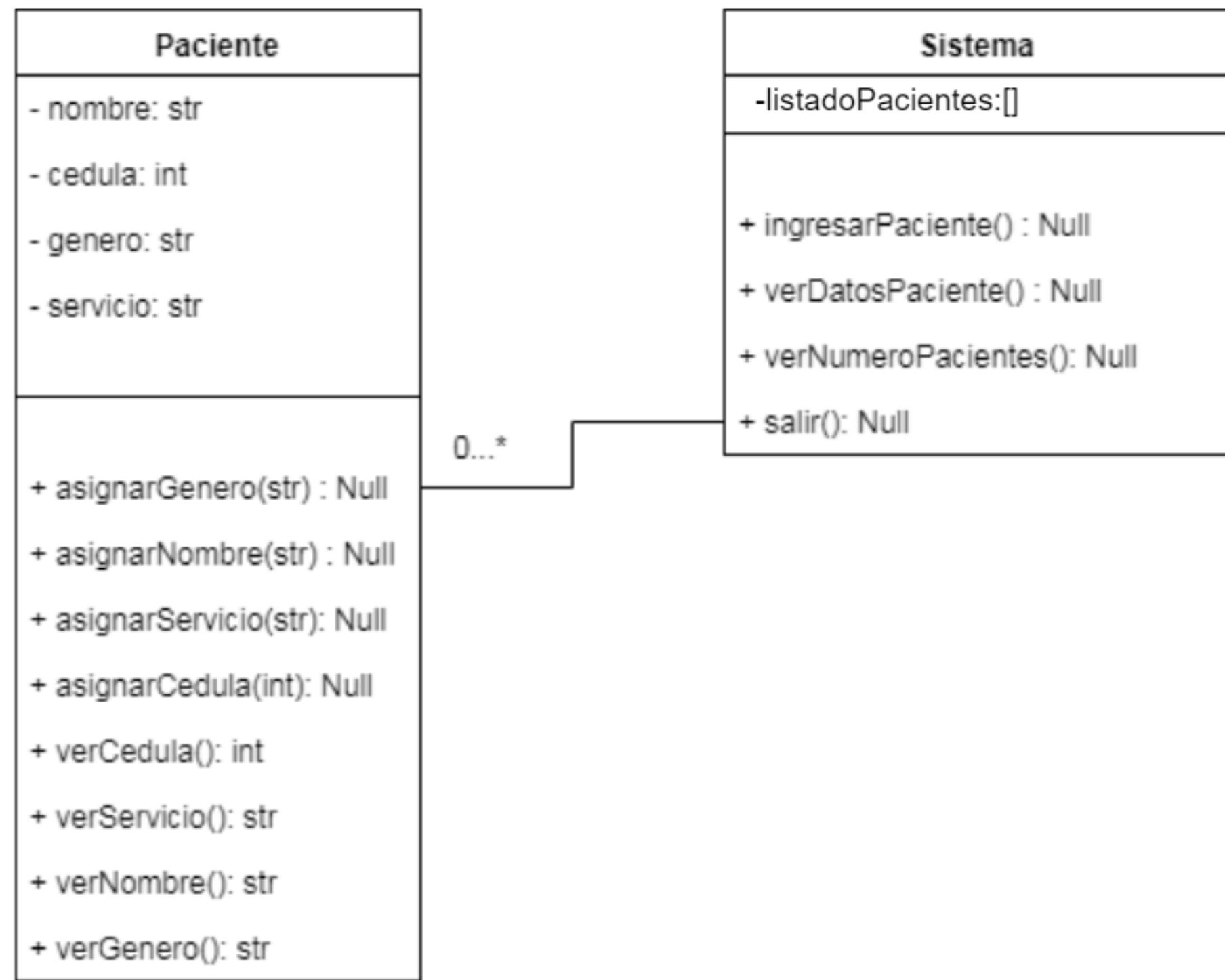
Segundo ejercicio de abstracción, Version1

Las funciones a programar dentro de la clase sistema pueden seguir esta lógica:

1. **Ingresar un paciente Nuevo:** Se solicitan por teclado los datos, se crea un Nuevo objeto Paciente con los datos y se almacena
2. **Ver todos los datos de un paciente existente:** Se ingresa un número de cédula y se busca el Paciente con dicha cédula, cuando se encuentra se muestra en pantalla la información de paciente
3. **Ver número de pacientes en el sistema:** Muestra la cantidad de personas ingresadas
4. **Salir:** Entrega un mensaje de despedida y termina el programa



Diagrama UML, 2do Ejercicio (V1)





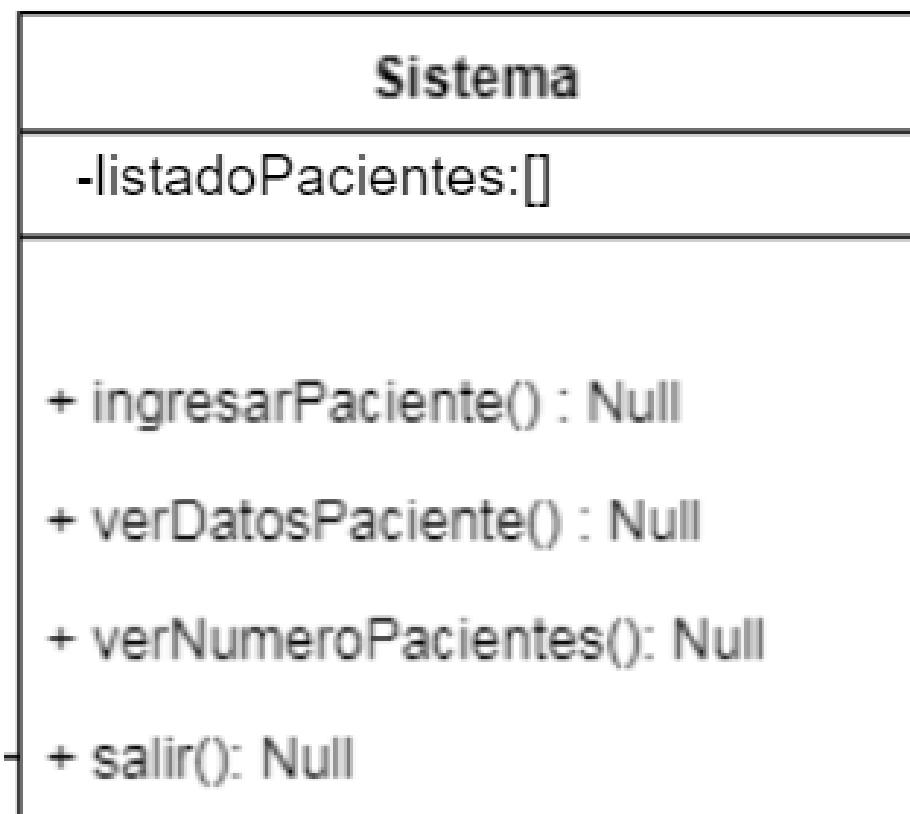
Para adelantar...

Paciente
- nombre: str
- cedula: int
- genero: str
- servicio: str
+ asignarGenero(str) : Null
+ asignarNombre(str) : Null
+ asignarServicio(str): Null
+ asignarCedula(int): Null
+ verCedula(): int
+ verServicio(): str
+ verNombre(): str
+ verGenero(): str

```
class Paciente:  
    def __init__(self):  
        self.__nombre = ""  
        self.__cedula = 0  
        self.__genero = ""  
        self.__servicio = ""  
  
    def verNombre(self):  
        return self.__nombre  
    def verServicio(self):  
        return self.__servicio  
    def verGenero(self):  
        return self.__genero  
    def verCedula(self):  
        return self.__cedula  
  
    def asignarNombre(self,n):  
        self.__nombre = n  
    def asignarServicio(self,s):  
        self.__servicio = s  
    def asignarGenero(self,g):  
        self.__genero = g  
    def asignarCedula(self,c):  
        self.__cedula = c
```



Para adelantar...



```
class Sistema:  
    def __init__(self):  
        #como voy a tener uno o muchos pacientes los manejaré en una lista  
        self.__lista_pacientes = [ ]  
        #esta variable siempre dependerá del tamaño de la lista por lo  
        #que no se podrá modificar  
        # con un método asignar  
        self.__numero_pacientes = len(self.__lista_pacientes)  
    def ingresarPaciente(self):  
        #1 solicito los datos por teclado  
        nombre = input("Ingrese el nombre: ")  
        cedula = int(input("Ingrese la cedula: "))  
        genero = input("Ingrese el genero: ")  
        servicio = input("Ingrese el servicio: ")  
        #2 creo el objeto Paciente y le asigno los datos  
        p = Paciente()  
        p.asignarNombre(nombre)  
        p.asignarCedula(cedula)  
        p.asignarGenero(genero)  
        p.asignarServicio(servicio)  
        #3 guardo el paciente en la lista  
        self.__lista_pacientes.append(p)  
        #4 actualizo la cantidad de pacientes en el sistema  
        self.__numero_pacientes = len(self.__lista_pacientes)  
    def verNumeroPacientes(self):  
        return self.__numero_pacientes
```



Para adelantar...

Sistema
-listadoPacientes:[]
+ ingresarPaciente(): Null
+ verDatosPaciente(): Null
+ verNumeroPacientes(): Null
- salir(): Null

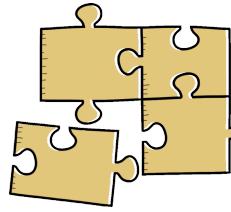
```
def verNumeroPacientes(self):
    return self._numero_pacientes
def verDatosPaciente(self):
    cedula = int(input("Ingrese la cedula a buscar: "))
    #cedula in self._lista_pacientes: no sirve porque en la lista
    # hay Pacientes no numeros:
    for paciente in self._lista_pacientes:
        if cedula == paciente.verCedula():
            #si coincide la cedula del paciente con la buscada
            # | muestro los datos
            print("Nombre: " + paciente.verNombre())
            print("Cedula: " + str(paciente.verCedula()))
            print("Genero: " + paciente.verGenero())
            print("Servicio: " + paciente.verServicio())
```



Para adelantar....

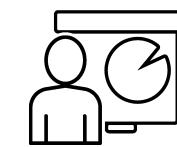
```
#cuando creamos las clases podemos generar objetos de esas clases y con esos objetos
#acceder a las funcionalidades o metodos
mi_sistema = Sistema() # Se Crea una instancia de la clase Sistema.

#ciclo infinito
while True:
    opcion = int(input("1. nuevo paciente - 2. Numero de Pacientes - 3. Datos Paciente - 4. Salir"))
    if opcion == 1:
        mi_sistema.ingresarPaciente()
    elif opcion == 2:
        print("Ahora hay : " + str(mi_sistema.verNumeroPacientes()))
    elif opcion == 3:
        mi_sistema.verDatosPaciente()
    elif opcion == 4:
        break
    else:
        print("Opcion invalida")
```



Repaso

- 1. ¿ Dónde existen las variables ?**
- 2. ¿ En qué momento existen los atributos ?**
- 3. Los parámetros de una función, ¿ en qué momento existen ?**



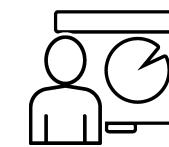
Modelo verbal

Un modelo puede ser tan sencillo como una simple explicación con palabras de lo fundamental de una realidad. A este tipo se le suele llamar modelo verbal.



Brecha en la comunicación: “Sé que cree que entendió lo que piensa que dije, pero no estoy seguro de que se dé cuenta de que lo que escuchó no es lo que quise decir”. Consecuencia de las diferencias en las especialidades del usuario y el analista



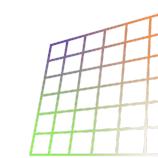


Modelo verbal



Un buen modelo permite **predecir** situaciones futuras y hacer "**experimentos**" que nunca serían posibles en la realidad.

La **limitación** obvia es que un modelo imita, pero no es, la realidad. No toda la información está definida de manera explícita en el modelo verbal, a veces es necesario definir con el cliente varias veces el problema hasta estar claro o introducir funcionalidades que están implícitas.



Segundo ejercicio de abstracción, Versión 2

1. **Ingresar un paciente Nuevo:** Se solicita la información desde el método main y se pasa un **objeto Paciente** a la clase Sistema
2. **Ver todos los datos de un paciente existente:** Se pasa el número de cédula y la clase Sistema devuelve **el objeto** asociado a esa cédula
3. **Ver número de pacientes en el sistema:** Muestra la cantidad de personas ingresadas
4. **Salir:** Entrega un mensaje de despedida y termina el programa

Lógica de funcionamiento versión 2: Usando la **función main**. La idea es que haya un método principal desde donde se gestiona la entrada de datos por teclado y la salida de información a Pantalla

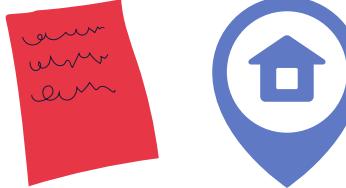
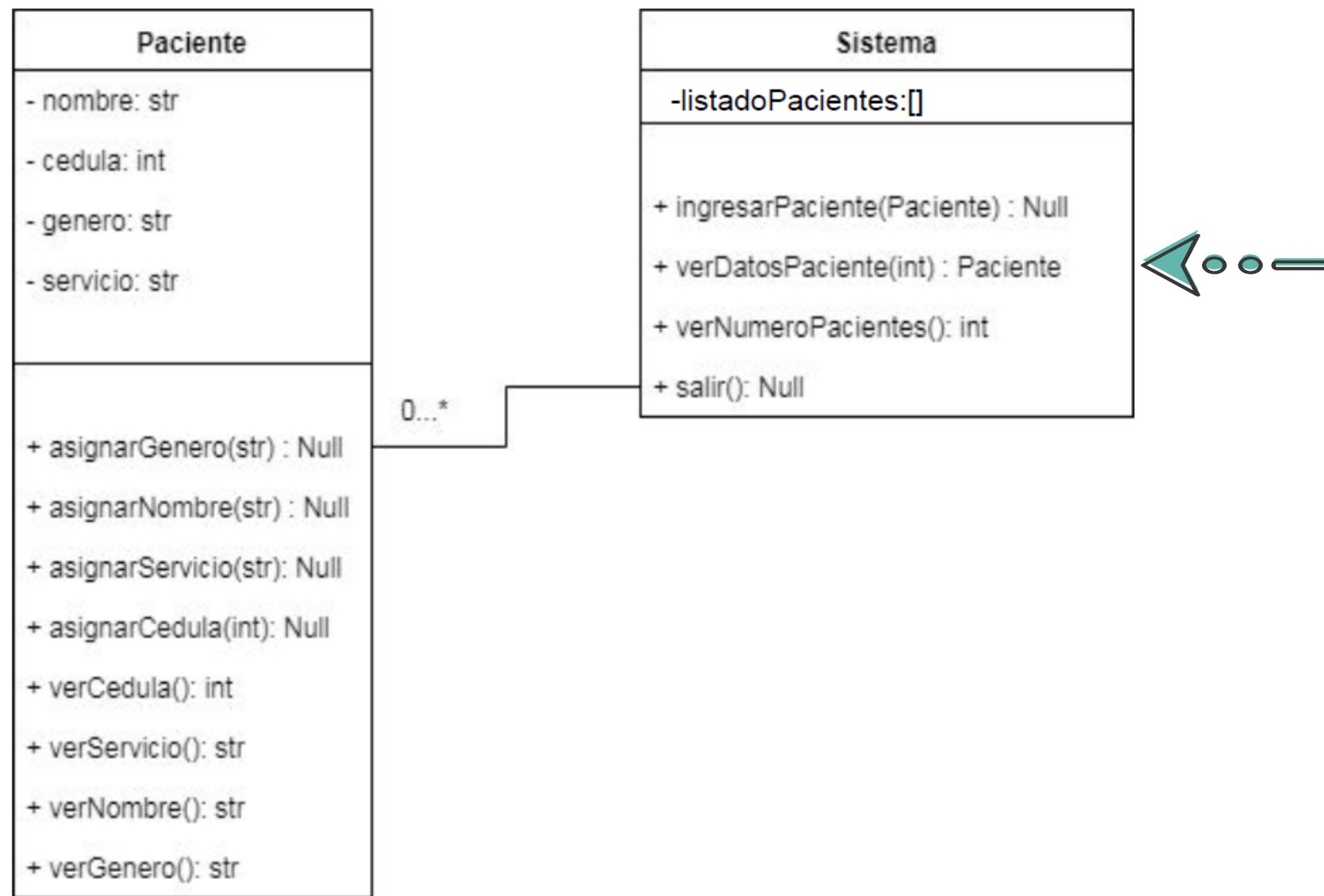


Diagrama UML, 2do ejercicio Versión 2





Segundo ejercicio de abstracción, Versión 2

La clase persona no varía , pero la clase sistema si, según lo solicitado en la diapositiva anterior

```
class Sistema:  
    def __init__(self):  
        self.__lista_pacientes = []  
    def ingresarPaciente(self,pac):  
        self.__lista_pacientes.append(pac)  
    def verDatosPaciente(self, c):  
        #voy a buscar paciente por pacietne  
        for p in self.__lista_pacientes:  
            #por cada paciente de la lista, le digo al paciente que me  
            #retorne la cedula y la comparo con la ingresada por teclado  
            if c == p.verCedula():  
                return p #si encuentro el paciente lo retorno  
    def verNumeroPacientes(self):  
        print("En el sistema hay: " + str(len(self.__lista_pacientes)) + " pacientes")
```



Segundo ejercicio de abstracción, Versión 2

En la función ***main*** vemos como varía la solicitud y tratamiento de los pacientes, con respecto a la versión anterior del programa.

```
def main():
    sis = Sistema()
    #probemos lo que llevamos programado
    while True:
        #TAREA HACER EL MENU
        opcion = int(input("Ingrese 0 para salir, 1 para ingresar nuevo paciente, 2 ver Paciente: "))
        if opcion == 1:
            #ingreso pacientes
            print("A continuacion se solicitaran los datos ...")
            #1. Se solicitan los datos
            nombre = input("Ingrese el nombre: ")
            cedula = int(input("Ingrese la cedula: "))
            genero = input("Ingrese el genero: ")
            servicio = input("Ingrese el servicio: ")
            #2. se crea un objeto Paciente
            pac = Paciente()
            #como el paciente esta vacio debo ingresarle la informacion
            pac.asignarCedula(cedula)
            pac.asignarGenero(genero)
            pac.asignarNombre(nombre)
            pac.asignarServicio(servicio)
            #3. se almacena en la lista que esta dentro de la clase sistema
            sis.ingresarPaciente(pac)
        elif opcion == 2:
```



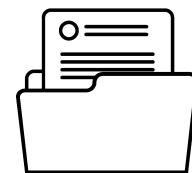
Segundo ejercicio de abstracción, Versión 2

```
def main():
    #aca el python descubre cual es la funcion principal
    if __name__ == "__main__":
        main()

    #menu de opciones
    print("1. Ver datos paciente")
    print("2. Buscar paciente")
    print("3. Salir")

    opcion = int(input("Elija una opción: "))

    #elif opcion == 1:
    #    #1. solicito la cedula que quiero buscar
    #    c = int(input("Ingrese la cedula a buscar: "))
    #    #le pido al sistema que me devuelva en la variable p al paciente que tenga
    #    #la cedula c en la lista
    #    p = sis.verDatosPaciente(c)
    #    #2. si encuentro al paciente imprimo los datos
    #    print("Nombre: " + p.verNombre())
    #    print("Cedula: " + str(p.verCedula()))
    #    print("Genero: " + p.verGenero())
    #    print("Servicio: " + p.verServicio())
    elif opcion == 2:
        #1. solicito la cedula que quiero buscar
        c = int(input("Ingrese la cedula a buscar: "))
        #le pido al sistema que me devuelva en la variable p al paciente que tenga
        #la cedula c en la lista
        p = sis.verDatosPaciente(c)
        #2. si encuentro al paciente imprimo los datos
        print("Nombre: " + p.verNombre())
        print("Cedula: " + str(p.verCedula()))
        print("Genero: " + p.verGenero())
        print("Servicio: " + p.verServicio())
    elif opcion !=0:
        continue
    else:
        break
```



Segundo ejercicio de abstracción, Versión 3

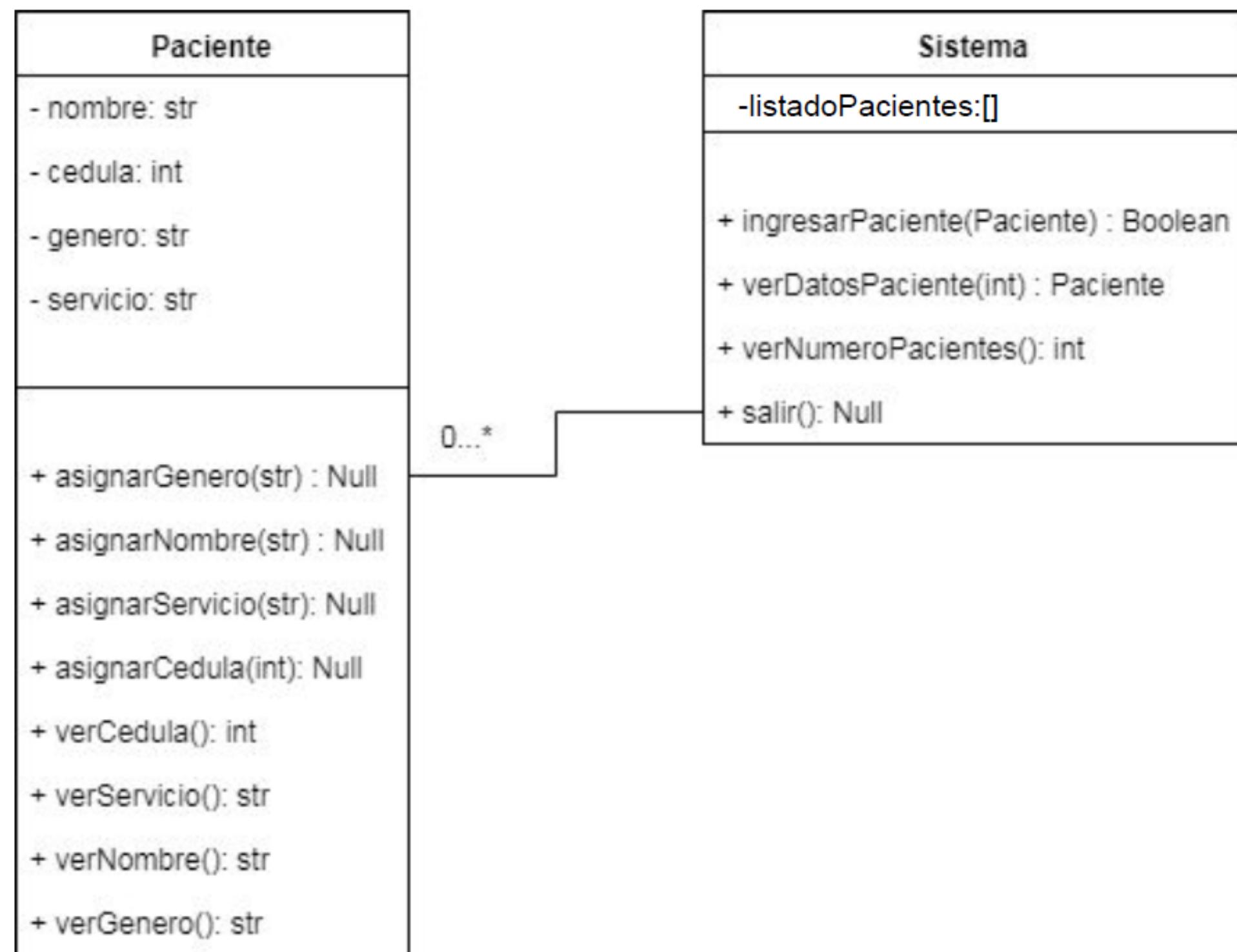
1. **Ingresar un paciente Nuevo:** Se solicita la información desde el **método main** y se pasa un objeto Paciente a la clase Sistema
2. **Ver todos los datos de un paciente existente:** Se pasa el número de cédula y la clase Sistema devuelve el objeto asociado a esa cédula
3. Ver número de pacientes en el sistema
4. Salir

Lógica de funcionamiento versión 3: Usando la función main. La idea es que haya un método principal desde donde se gestiona la entrada de datos por teclado y la salida de información a Pantalla

Manejando verificaciones: Se debe verificar cuando se ingresa un Nuevo paciente que no existe otro con la misma cédula



Segundo ejercicio de abstracción, Versión 3





Problemas adicionales para profundizar..

Proponer mejoras al sistema desarrollado

Ver todos los datos de un paciente existente: Se pasa el número de cédula o el nombre del mismo, puede ser completo o como se recuerde, y la clase Sistema devuelve el objeto asociado a esa cédula o los pacientes que inician con dicho nombre



GRACIAS

HelloWorldSkinProc(SkinProc
on 'hello')
HelloWorld.SetSkinProc(SkinProc)