ALGORITMOS: Guía de ejercicios N° 2

La compañía de seguros de la guía de ejercicios N° 1, a avanzado en la implementación de su nuevo producto de seguros de vida, requiriendo un modelo que le permita representar a los productores de seguros y los clientes de cada tipo de seguro.
El objetivo es poder consultar información que tienen disponible para cada uno de los promotores en archivos de texto.

a) Implemente un modelo que represente **personas**, **promotores** y **clientes** considerando que:

- **persona**: representa una persona en el modelo a partir de su nombre (atributo **_nom** de tipo cadena de caracteres) y DNI (atributo **_dni** de tipo numérico entero).
  Permite la carga de los valores de interés a través del constructor e incorpora métodos para acceder a estos valores (**getDNI** y **getNombre**). También implementa métodos que permiten establecer relación de orden entre dos objetos de esta clase a partir de su número de documento **(__lt__**, **__le__**, **__gt__, __ge__, __eq__, __ne__**) y genera una representación de cadena de caracteres (método **__str__**) utilizando el número de documento con 8 posiciones y el nombre separado uno del otro utilizando el carácter pipe "|" (por ejemplo " 8765123 | Juan Pérez").
- **promotor**: clase derivada de la clase **persona**, representa un promotor de seguros. Agrega una lista de objetos de la clase cliente descrita más abajo (atributo **_clientes**), esta lista contiene todos los objetos que representan los clientes de un promotor de seguros particular. Además del constructor, que crea una lista de clientes vacía, incorpora los siguientes métodos:
  - **addCliente**: agrega un objeto cliente dado como argumento a la lista utilizando el método **append**.
  - **getCntClientes**: retorna la cantidad de clientes almacenados en la lista de clientes. Este valor es determinado utilizando la función **len** y la lista de clientes como argumento.
  - **getClientes**: retorna la lista de clientes a fin de poder iterar sobre ella (atributo _clientes).
  - **__str__**: sobre escribe el método definido en la clase base a fin de retornar solo el nombre completo del promotor.
- **cliente**: clase derivada de la clase **persona** que incorpora como atributo el promotor con el que se relaciona y el tipo de seguro. El promotor es un objeto de la clase **promotor**, mientras que el tipo de seguro está dado como una cadena de caracteres que puede ser "Seguro de Vida" o "Seguro Automotor (AA 000 AA)" donde **AA 000 AA** corresponde con la patente del automotor asegurado.
  El objeto de la clase promotor puede ser (y será en esta actividad) compartido por múltiples objetos de la clase cliente, mientras que el promotor contendrá

una lista de objetos de la clase cliente. De este modo se representa el hecho que un cliente tiene un promotor y un promotor tiene múltiples clientes. La clase cliente deberá incorporar un método **getPromotor** que retornará el objeto promotor. De este modo a partir de un objeto cliente es posible acceder a los datos de su promotor de seguros.
En lo que respecta al tipo de seguro sucede algo similar, un mismo cliente puede tener contratados diferentes tipos de seguros, por lo que el tipo de seguro en la clase cliente es representado a través de una lista de cadenas de caracteres.
Debido a que a través del constructor se indica un tipo de seguro (forzando a que todos los clientes tienen por lo menos un seguro contratado), los demás seguros que se contraten se agregan a través del método **addTipoSeguro**. Este método recibe como argumento una cadena de caracteres y la agrega a la lista de seguros utilizando el método **append**.
Finalmente, la clase cliente incorpora el método **getSeguros** que retorna la lista de cadenas de caracteres con los seguros contratados por un cliente particular.

Utilice el modelo anterior en una aplicación como la que se muestra a continuación a fin de comprobar que el modelo funciona correctamente:

```
def main():
  # Creamos el promotor
  prom = promotor(24317128, "Diego de la Vega")
  # Creamos un cliente, vinculándolo al promotor
  cli01 = cliente(11765989, "Carlos Avellaneda", prom, "Seguro de Vida")
  # y vinculamos al promotor con el cliente
  prom.addCliente(cli01)
  # Creamos el segundo cliente
  cli02 = cliente(11765989, "Analía Mendizábal", prom, "Seguro Automotor (NPE 321)")
  # Agregamos un segundo tipo de seguro
  cli02.addTipoSeguro("Seguro de Vida")
  # Agregamos el cliente a la lista atendida por el promotor
  prom.addCliente(cli02)
  # Situación similar con el tercer cliente
  cli03 = cliente(11765989, "Jaquín Mendez", prom, "Seguro Automotor (AB 321 ZF)")
  prom.addCliente(cli03)
  # Mostramos los datos del promotor
  print(f"{prom}")
  # Y el listado de sus clientes
  print("Listado de clientes:")
  for cli in prom.getClientes():
     print(f"{cli}")
     print("Productos contratados:")
     for s in cli.getSeguros():
```

```
print(f"\t {s}")
```

b) Modifica que aplicación anterior a fin de que la información de los clientes sea obtenida a partir de un archivo de texto conteniendo la siguiente información:

```
23435213; Ana María Trigati; Seguro de Vida | Seguro automotor (NUV210)
26678123; Alicia Mercado; Seguro de Vida;
32564876; Gustavo Lamperti; Seguro automotor (AU 123 VM)
27654987; Paulina Gutierrez; Seguro automotor (AG 023 AA)
23987189; Raúl Álvarez; Seguro automotor (AF 956 CA) | Seguro automotor (MON830)
17893076; Laura Arellano; Seguro de Vida
18763908; Nicolás Sampieri; Seguro automotor (COW217)
28765932; María José Fiorito; Seguro automotor (GUZ712)
26898320; José Luis Pérez; Seguro de Vida
24980167; Leopoldo Martínez; Seguro automotor (WNK321) | Seguro de Vida
12345678; Juan Carlos Nicoletti; Seguro de Vida | Seguro automotor (AH 431 BD)
```

Este listado corresponde con los clientes del promotor Diego de la Vega. Por lo que el objeto de la clase **promotor** debe crearse con datos fijos, mientras que solo los clientes se crean con la información contenida en el archivo. Para realizar esta actividad el procedimiento es similar al realizado en la guía de ejercicios 1.

- La subdivisión del contenido de cada renglón en la información requerida para crear cada objeto de la clase cliente se logra utilizando el método **split**. Con esto se tendrá una lista que, si el contenido del renglón es correcto, tendrá 3 elementos.
- Antes de utilizar cada elemento es recomendable utilizar el método **strip** a fin de eliminar caracteres adicionales tales como marcas de salto de línea o espacios en blanco.
- El último campo es el tipo de seguro. En algunos casos un mismo cliente tiene contratado más de un tipo de seguro, por lo que la lista de seguros contratados se tiene utilizando nuevamente el método **split** sobre el último elemento de la lista con los datos de cada renglón. Cada renglón tendrá como mínimo un producto, por lo que por cada renglón se tendrá necesariamente un tipo de seguro. En algunos casos, se tendrán más y es necesario agregar el nuevo tipo de seguro como una nueva cadena en el objeto cliente.

c) Agregue los métodos **ordenarClientes** y **buscarCliente** a la clase **promotor** de modo que el ordenamiento se realice sobre todos los clientes según su DNI utilizando alguno de los algoritmos tratados en esta semana. Note que, debido a que la clase persona incorpora métodos para comparar objetos de su tipo según el DNI es posible comparar objetos clientes de modo directo y la relación de orden estará dada por su número de DNI. Se recomienda utilizar una implementación del método quick sort, pero no es obligatorio que elija este método (fundamente su elección). En lo que respecta al algoritmo de búsqueda, implemente el método de búsqueda por bisección (o búsqueda binaria), a fin de que sea posible buscar un

cliente a partir de su DNI, retornando el objeto de la clase cliente o **None** si no existe un cliente con el número de DNI indicado.
Modifique la aplicación a fin de que permita ver los datos del promotor y el listado de todos los DNI y nombres de sus clientes. Luego permita ingresar números de DNI, busque este valor sobre la lista de clientes ordenada utilizando los métodos implementados. Si se encuentra coincidencia con el DNI de un cliente deberá presentar sus datos por pantalla, así como los seguros que contrató. En caso de no encontrar un cliente con el DNI indicado se deberá mostrar un mensaje alusivo. La aplicación debe repetir el proceso de ingreso de DNI y búsqueda hasta que el valor ingresado sea cero o negativo.

Nota de implementación: Si utiliza el método quick sort, debido a que la implementación considerada es recursiva, es recomendable crear un método **ordenarClientes** que no tome argumentos y otro método **_ordenar_quickSort** que tome como argumento la lista a considerar. La implementación del primero invoca al segundo, siendo este el que se invoca a si mismo hasta terminar el ordenamiento. De este modo el método **ordenarClientes** es transparente al usuario, mientras que **_ordenar_quickSort** permanece oculto y realiza la tarea requerida utilizando recursividad.