Solución Desafío Guiado - Membresía

Requerimiento 1

Archivo membresia.py

1. Importar `ABC` y `abstractmethod` desde abc, y definir la clase abstracta `Membresia`, con un constructor que reciba por parámetro un correo de suscriptor y un número de tarjeta, que se deben asignar en los atributos privados de instancia correspondientes.

```
from abc import ABC, abstractmethod
from typing import Union

class Membresia(ABC):
    def __init__(self, correo_suscriptor: str, numero_tarjeta: str) -> None:
        self.__correo_suscriptor = correo_suscriptor
        self.__numero_tarjeta = numero_tarjeta
```

2. En la misma clase, definir las propiedades que permitan obtener los valores del correo del suscriptor y del número de tarjeta de la instancia.

```
    @property
    def correo_suscriptor(self) -> str:
        return self.__correo_suscriptor

    @property
    def numero_tarjeta(self) -> str:
        return self.__numero_tarjeta
```

3. En la misma clase, definir el método abstracto que permite cambiar la suscripción. Debe recibir por parámetro la nueva membresía, como número entero.

```
    @abstractmethod
    def cambiar_suscripcion(self, nueva_membresia: int) -> Union[Basica,
Familiar, SinConexion, Pro]:
        pass
```

4. En la misma clase, definir el método que genera nuevas membresías según el número de membresía entregado.

```
    def _crear_nueva_membresia(self, nueva_membresia: int) -> Union[Basica,
Familiar, SinConexion, Pro]:
```

```
        if nueva_membresia == 1:
            return Basica(self.correo_suscriptor, self.numero_tarjeta)
        elif nueva_membresia == 2:
            return Familiar(self.correo_suscriptor, self.numero_tarjeta)
        elif nueva_membresia == 3:
            return SinConexion(self.correo_suscriptor, self.numero_tarjeta)
        elif nueva_membresia == 4:
            return Pro(self.correo_suscriptor, self.numero_tarjeta)
```

Requerimiento 2

Archivo membresia.py

1. En el mismo archivo, definir la clase `Gratis`, que hereda de `Membresia`. Declarar en ella los atributos de clase con los valores solicitados.

```
class Gratis(Membresia):
    costo = 0
    cantidad_dispositivos = 1
```

2. En la misma clase, definir el método para cambiar la suscripción. En caso de que la nueva membresía sea menor a 1 o mayor a 4, debe retornar la instancia actual. En caso contrario, hace el llamado al método para crear una nueva membresía.

```
    def cambiar_suscripcion(self, nueva_membresia: int) -> Union[Basica,
Familiar, SinConexion, Pro]:
        if nueva_membresia < 1 or nueva_membresia > 4:
            return self
        else:
            return self._crear_nueva_membresia(nueva_membresia)
```

Requerimiento 3

Archivo membresia.py

1. En el mismo archivo, definir la clase `Basica`, que hereda de `Membresia`. Declarar en ella los atributos de clase con los valores solicitados.

```
class Basica(Membresia):
    costo = 3000
    cantidad_dispositivos = 2
```

2. En la misma clase, sobrescribir el constructor para establecer los 7 días de regalo en caso de las membresías `Familiar` y `SinConexion`, y 15 en el caso de las `Pro`. Debe también llamar al constructor de la clase padre.

```
    def __init__(self, correo_suscriptor: str, numero_tarjeta: str) -> None:
        super().__init__(correo_suscriptor, numero_tarjeta)

        if isinstance(self, Familiar) or isinstance(self, SinConexion):
            self.__dias_regalo = 7

        elif isinstance(self, Pro):
            self.__dias_regalo = 15
```

3. En la misma clase, definir el método para cancelar la suscripción

```
    def cancelar_suscripcion(self) -> Gratis:
        return Gratis(self.correo_suscriptor, self.numero_tarjeta)
```

4. En la misma clase, definir el método para cambiar la suscripción. En caso de que la nueva membresía sea menor a 2 o mayor a 4, debe retornar la instancia actual. En caso contrario, hace el llamado al método para crear una nueva membresía.

```
    def cambiar_suscripcion(self, nueva_membresia: int) -> Union[Basica,
Familiar, SinConexion, Pro]:
        if nueva_membresia < 1 or nueva_membresia > 4:
            return self
        else:
            return self._crear_nueva_membresia(nueva_membresia)
```

Requerimiento 4

Archivo membresia.py

1. En el mismo archivo, definir la clase `Familiar`, que hereda de `Basica`. Declarar en ella los atributos de clase con los valores solicitados.

```
class Familiar(Basica):
    costo = 5000
    cantidad_dispositivos = 5
```

2. En la misma clase, definir el método para cambiar la suscripción. En caso de que la nueva membresía no sea 1, ni 3 ni 4, retorna la instancia actual. En caso contrario, hace el llamado al método para crear una nueva membresía.

```
    def cambiar_suscripcion(self, nueva_membresia: int) -> Union[Basica,
Familiar, SinConexion, Pro]:
        if nueva_membresia not in [1, 3, 4]:
            return self
        else:
            return self._crear_nueva_membresia(nueva_membresia)
```

3. Definir método para modificar control parental, sin implementación

```
    def modificar_control_parental(self) -> None:
        pass
```

Requerimiento 5

Archivo membresia.py

1. En el mismo archivo, definir la clase `SinConexion`, que hereda de `Basica`. Declarar en ella los atributos de clase con los valores solicitados.

```
class SinConexion(Basica):
    costo = 3500
```

2. En la misma clase, definir el método para cambiar la suscripción. En caso de que la nueva membresía no sea 1, ni 2 ni 4, retorna la instancia actual. En caso contrario, hace el llamado al método para crear una nueva membresía.

```
    def cambiar_suscripcion(self, nueva_membresia: int) -> Union[Basica,
Familiar, SinConexion, Pro]:
        if nueva_membresia not in [1, 2, 4]:
            return self
        else:
            return self._crear_nueva_membresia(nueva_membresia)
```

3. Definir método para incrementar cantidad máxima sin conexión

```
    def incrementar_cantidad_maxima_offline(self) -> None:
        pass
```

Requerimiento 6

Archivo membresia.py

1. En el mismo archivo, definir la clase `Pro`, que hereda de `Familiar` y `SinConexion`. Declarar en ella los atributos de clase con los valores solicitados.

```
class Pro(Familiar, SinConexion):
    costo = 7000
    cantidad_dispositivos = 6
```

2. En la misma clase, definir el método para cambiar la suscripción. En caso de que la nueva membresía sea menor a 1 o mayor a 3, retorna la instancia actual. En caso contrario, hace el llamado al método para crear una nueva membresía.

```
    def cambiar_suscripcion(self, nueva_membresia: int) -> Union[Basica,
Familiar, SinConexion, Pro]:
        return min(max(nueva_membresia, 1), 3)
```