Actividad 4 Módulo 2

Nombre: Carlos Saldivia Susperreguy

Capturas:

actividad4.py

```python
# ¿Qué es la Programación Orientada a Objetos (POO)?
# La Programación Orientada a Objetos (POO) es un paradigma de programación que permite modelar el mundo real utilizando "objetos".
# Cada objeto es una entidad que combina datos (atributos) y comportamientos (métodos) en una sola estructura llamada clase.
# Funciona como un plano para crear instancias específicas (objetos).

# ¿Cuáles son sus cuatro principios fundamentales?
# 1.  Abstracción: Ocultar la complejidad interna y mostrar solo las características esenciales de un objeto.
# 2.  Encapsulamiento: Agrupar datos y métodos que operan sobre esos datos dentro de un objeto, protegiendo los datos de modificaciones externas no autorizadas.
# 3.  Herencia: Permitir que una clase (hija) herede atributos y métodos de otra clase (padre), promoviendo la reutilización de código.
# 4.  Polimorfismo: Capacidad de que objetos de diferentes clases respondan al mismo mensaje (método) de maneras distintas.

# ¿Qué ventajas aporta la POO frente a la programación estructurada?
# -   Mejora la organización del código: Al agrupar el código en objetos, la estructura es más intuitiva y clara.
# -   Reutilización de componentes: Las clases y objetos pueden ser reutilizados fácilmente en diferentes partes del programa o en otros proyectos (herencia).
# -   Mantenimiento y escalabilidad: Es más fácil modificar o expandir el sistema sin afectar otras partes, gracias a la modularidad.
# -   Fomenta el trabajo colaborativo: Diferentes desarrolladores pueden trabajar en distintas clases simultáneamente.
```

```python
# Clase Usuario
class Usuario:
    def __init__(self, nombre, correo, contraseña):
        self.nombre = nombre
        self.correo = correo
        self.__contraseña = contraseña

    def saludar(self):
        return f"Hola {self.nombre}, bienvenido."

    def mostrar_info(self):
        return f"Nombre: {self.nombre}, Correo: {self.correo}"

    @property
    def contraseña(self):
        return self.__contraseña

    @contraseña.setter
    def contraseña(self, nueva_contraseña):
        self.__contraseña = nueva_contraseña
```

```python
# Clase Administrador (hereda de Usuario)
class Administrador(Usuario):
    def __init__(self, nombre, correo, contraseña):
        super().__init__(nombre, correo, contraseña)
        self.permisos = []

    def mostrar_info(self):
        return f"{super().mostrar_info()}, Permisos: {self.permisos}"

    def agregar_permiso(self, permiso):
        self.permisos.append(permiso)
```

```python
# Instanciación y pruebas
usuario1 = Usuario("Carlos", "carlos@campuskibernum.cl", "usuario123")
admin1 = Administrador("Mario", "mario@campuskibernum.cl", "admin123")

print(usuario1.saludar())
print(usuario1.mostrar_info())
print("Contraseña actual del usuario:", usuario1.contraseña)
usuario1.contraseña = "nuevaContraseña"
print("Nueva contraseña del usuario:", usuario1.contraseña)

print("\n" + admin1.saludar())
print(admin1.mostrar_info())
admin1.agregar_permiso("modificar_usuarios")
admin1.agregar_permiso("eliminar_usuarios")
print(admin1.mostrar_info())
```

```
Hola Carlos, bienvenido.
Nombre: Carlos, Correo: carlos@campuskibernum.cl
Contraseña actual del usuario: usuario123
Nueva contraseña del usuario: nuevaContraseña

Hola Mario, bienvenido.
Nombre: Mario, Correo: mario@campuskibernum.cl, Permisos: []
Nombre: Mario, Correo: mario@campuskibernum.cl, Permisos: ['modificar_usuarios', 'eliminar_usuarios']
```