REPORTE
ACTIVIDAD 3
HERENCIA, AGREGACION, Y COMPOSICION EN PYTHON
MARCO ANTONIO VARGAS LIRA

Este proyecto es un **sistema básico de ventas** desarrollado en Python usando **Programación Orientada a Objetos (POO)**. Permite manejar clientes, productos, ventas y la tienda donde se registran esas ventas.

Estructura y funcionamiento

1. **Usuarios**
   Se define una clase abstracta Usuario que establece los atributos básicos de cualquier usuario: **nombre** y **correo**. Esta clase obliga a que sus hijos implementen un método mostrar_info(), asegurando que cada tipo de usuario pueda mostrar su información correctamente.

2. **Clientes y Administradores**

    - La clase Clientes hereda de Usuario e incorpora un **saldo**, representando el dinero disponible del cliente.

    - La clase administrador también hereda de Usuario y contiene una lista de **permisos**, mostrando qué acciones puede realizar dentro del sistema.
      Ambos implementan el método mostrar_info() para presentar sus datos de manera clara.

3. **Productos**
   Cada producto tiene un **nombre** y un **precio**, y el sistema puede validar que el precio sea positivo. También mantiene un conteo total de productos creados.

4. **Ventas**
   La clase venta conecta un cliente con una lista de productos comprados, permitiendo agregar productos y calcular el total de la venta de forma sencilla.

5. **Tienda**
   La clase Tienda registra las ventas realizadas, almacenándolas en una lista. Esto permite saber cuántas ventas se han hecho y acceder a cada una

cuando sea necesario.

Flujo de uso

1. Se crea un cliente y algunos productos.
2. Se genera una venta, agregando productos al cliente.
3. La venta se registra en la tienda.
4. Finalmente, se puede mostrar:
    - Información del cliente
    - Total de la venta
    - Cantidad de ventas registradas

Conclusión

El proyecto es un ejemplo claro de cómo organizar un sistema de ventas usando POO:

- Utiliza **herencia** para evitar duplicar código entre usuarios.
- Aplica **abstracción** con métodos obligatorios (mostrar_info).
- Separa responsabilidades: productos, ventas, clientes y la tienda están en clases independientes.
- Facilita la **expansión futura**, como agregar más tipos de usuarios, productos o reportes de ventas.

En pocas palabras, es un sistema modular y escalable que refleja buenas prácticas de programación orientada a objetos en Python.

```python
from abc import ABC, abstractmethod



class Usuario(ABC) :

    def __init__(self, nombre: str, correo: str):

        self.nombre = nombre

        self.correo = correo



        @abstractmethod

        def mostrar_info(self) -> str:

            pass
```

```python
from usuario import Usuario



class Clientes(Usuario):

    def __init__(self, nombre: str, correo: str, saldo: float):

        super().__init__(nombre, correo)

        self.saldo = saldo



    def mostrar_info(self) -> str:

        return f"Cliente: {self.nombre}, Correo: {self.correo}, Saldo:
${self.saldo:.2f}"







class producto:

    contador_productos = 0



    def __init__(self, nombre: str, precio: float):

        self.nombre = nombre

        self.precio = precio

        producto.contador_productos += 1



    @staticmethod

    def es_precio_valido(precio: float) -> bool:

        return precio > 0
```

```python
    @classmethod
    def total_productos(cls) -> int:
        return cls.contador_productos


from usuario import Usuario


class administrador(Usuario):
    def __init__(self, nombre: str, correo: str, permisos: list[str]):
        super().__init__(nombre, correo)
        self.permisos = permisos


        def mostrar_info(self) -> str:
            return f"Administrador: {self.nombre}, correo:
{self.correo}, permisos: {','.join(self.permisos)}"


from producto import producto
from cliente import Clientes


class venta:
    def __init__(self, cliente: Clientes):
        self.cliente = cliente
        self.productos: list[producto] =[]


    def agregar_producto(self, producto: producto) -> None:
        self.productos.append(producto)
```

```
    def total(self) -> float:
        return sum(p.precio for p in self.productos)

from venta import venta

class Tienda:
    def __init__(self, nombre: str):
        self.nombre = nombre
        self.ventas: list[venta] = []

    def registrar_venta(self, venta: venta) -> None:
        self.ventas.append(venta)




from cliente import Clientes
from producto import producto
from venta import venta
from tienda import Tienda

# Crear cliente
cliente1 = Clientes("luis", "luis@mail.com", 1000)

# Crear productos
```

```python
p1 = producto("teclado", 250)

p2 = producto("mouse", 150)



# Crear venta y agregar productos

venta1 = venta(cliente1)

venta1.agregar_producto(p1)

venta1.agregar_producto(p2)



# Crear tienda y registrar venta

tienda = Tienda("techstore")

tienda.registrar_venta(venta1)



# Mostrar resultado

print(cliente1.mostrar_info())

print(f"total de la venta: ${venta1.total():.2f}")

print(f"ventas registradas: {len(tienda.ventas)}")
```

https://github.com/themarco11/Python-INTERMEDIO.git