Módulo 4 – Ecommerce CLI con POO y roles

1) Propósito
Re-estructurar el ecommerce de consola desarrollado en el Módulo 3 usando **Programación Orientada a Objetos en Python**, incorporando **roles** y utilizando:

- Clases, atributos y métodos (colaboración y composición).
- Herencia simple para diferenciar tipos de usuarios.
- Manejo de **excepciones** para validar y controlar errores.
- **Lectura/escritura de archivos** de texto para guardar información básica del sistema.

2) Descripción general de la app
La aplicación será un **Ecommerce por consola** con dos roles:

- **ADMIN**
  - Gestiona el catálogo de productos.
- **CLIENTE**
  - Navega por el catálogo, agrega productos al carrito y confirma compras.

Al iniciar, el programa pregunta si el usuario se identificará como **ADMIN** o **CLIENTE** y muestra el menú correspondiente.

3) Requisitos funcionales
3.1. Rol ADMIN
El ADMIN debe poder:

1. **Listar productos** del catálogo.
2. **Crear producto nuevo** indicando al menos: id, nombre, categoría, precio.
3. **Actualizar producto** (por ejemplo, cambiar nombre, precio o categoría).
4. **Eliminar producto** del catálogo.
5. **Guardar catálogo en archivo** (ej: catalogo.txt o catalogo.csv).

El catálogo inicial puede cargarse desde el código o, opcionalmente, leerse desde un archivo al inicio.

3.2. Rol CLIENTE
El CLIENTE debe poder:

1. **Ver catálogo de productos**.
2. **Buscar productos** por nombre o categoría.
3. **Agregar productos al carrito**, indicando:
   - id del producto.
   - **cantidad** (entero > 0).
4. **Ver carrito y total**:
   - Listar ítems (nombre, cantidad, precio unitario, subtotal).
   - Mostrar total a pagar.
5. **Confirmar compra**:
   - Si el carrito está vacío, avisar y no permitir la compra.
   - Registrar la compra en un archivo de texto simple (ej: ordenes.txt) con fecha/hora, productos y total.
   - Vaciar el carrito tras confirmar.

3.3. Manejo de errores (excepciones)

- Deben manejarse casos como:
  - ID de producto inexistente.
  - Cantidad menor o igual a 0.
  - Archivos que no se pueden abrir/escribir.
- Debe utilizarse al menos una **excepción personalizada** (por ejemplo, ProductoNoEncontradoError o CantidadInvalidaError) además de excepciones estándar.

4) Requisitos técnicos

La solución debe:

1. Estar implementada en **Python**, ejecutable en consola.
2. Incluir **clases** que reflejen la estructura del problema. Ejemplo (puedes adaptarlo):
    - Producto
    - Catalogo (contiene muchos Producto)
    - Carrito (contiene ítems de producto + cantidad)
    - Usuario (clase base)
    - Admin y Cliente (heredan de Usuario)
    - Aplicacion o Tienda (coordina la ejecución y menús)
3. Aplicar:
    - **Composición**: por ejemplo, Carrito tiene una colección de productos.
    - **Herencia**: Admin y Cliente extienden a Usuario, con comportamientos distintos.
4. Usar **métodos** claros para cada acción importante:
    - agregar/eliminar/actualizar producto,
    - agregar al carrito,
    - calcular total,
    - guardar/leer de archivo, etc.
5. Usar **excepciones** con bloques try/except y, cuando corresponda, finally.
6. Usar **archivos de texto** para:
    - Guardar catálogo (opcional pero recomendado).
    - Registrar compras (obligatorio).
7. Mantener buena **legibilidad**:
    - Nombres en snake_case.
    - Identación correcta.
    - Comentarios breves donde sea necesario.

No se debe usar bases de datos, librerías externas, frameworks web ni temas que no se hayan pasado en el módulo.

5) Entregables

- Código fuente (por ejemplo):
    - main.py
    - Otros archivos .py si se separa en módulos (opcional, pero recomendable).
- (Opcional) Un archivo README.md o .txt con:
    - Descripción breve de la app.
    - Cómo ejecutar el programa.

Rúbrica de evaluación

Criterio	Excelente (3 pts)	Adecuado (2 pts)	Básico (1 pt)	Insuficiente (0 pts)
Diseño OO (clases, herencia, composición)	Clases bien definidas; herencia y composición usadas de forma coherente; responsabilidades claras.	Clases correctas en general; herencia/composición usadas pero con algunas mezclas de responsabilidad.	Diseño OO muy básico; pocas clases o casi sin herencia/composición útil.	No se aplica realmente POO; todo queda en funciones/procedural.
Implementación de roles y funcionalidades	ADMIN y CLIENTE con menús claros; todas las operaciones clave (catalogar, buscar, carrito, compra) funcionan correctamente.	Ambos roles funcionan con pequeños errores o funciones faltantes menores.	Solo parte de las funciones de uno o ambos roles está implementada; flujo confuso.	Roles mal implementados o inexistentes; funcionalidades centrales incompletas.
Excepciones y archivos	Manejo de errores robusto con try/except y al menos una excepción personalizada; lectura/escritura de archivos funciona bien.	Manejo de errores parcial pero funcional; archivos usados con algún detalle menor.	Pocas excepciones o mal usadas; archivos manejados de forma muy limitada.	Sin manejo real de excepciones; no usa archivos o fallan.
Organización y legibilidad del código	Código modular, limpio, con nombres claros e identación adecuada; fácil de leer.	Organización buena en general; algunos detalles de estilo o claridad.	Código algo desordenado; difícil de seguir en algunas partes.	Código caótico o muy difícil de entender.