Proyecto Final POO: Sistema de Concesionaria

Guía de Requerimientos y Arquitectura

Curso: Python de 0 a Interfaces Gráficas

1. Introducción

El objetivo de este proyecto es desarrollar el sistema backend para una agencia de ventas de automóviles. Deberás aplicar los cuatro pilares de la Programación Orientada a Objetos (POO) y un manejo robusto de errores para garantizar que el sistema sea seguro y escalable.

2. Requerimientos del Sistema

El programa debe cumplir con las siguientes reglas de negocio:

1. **Inventario Diverso:** La agencia vende **Autos** y **Camionetas**. Ambos son vehículos, pero tienen impuestos y comisiones diferentes.
2. **Control de Precios:**
   - El precio base de un vehículo no puede ser negativo.
   - El precio final de venta se calcula automáticamente sumando comisiones e impuestos al precio base.
3. **Estados del Vehículo:** Un vehículo puede estar "Disponible.º "Vendido". No se puede vender un vehículo que ya ha sido vendido.
4. **Identificación:** Cada vehículo tiene un VIN (Número de Identificación Vehicular) único.

3. Arquitectura de Clases (El Diseño)

Deberás implementar la siguiente estructura de clases. Sigue este diseño al pie de la letra.

3.1. 1. Manejo de Errores

Clase: `ConcesionariaError`

- **Herencia:** Debe heredar de `Exception`.
- **Uso:** Se lanzará esta excepción cuando:

- Se intente asignar un precio negativo.
- Se intente vender un auto que ya no está disponible.
- Se intente registrar un VIN duplicado.

3.2. 2. Clase Abstracta: `Vehiculo`

Esta es la plantilla base. No se pueden crear objetos directos de esta clase.

- **Atributos:**
  - `vin` (texto).
  - `marca` (texto).
  - `modelo` (texto).
  - `_precio_base` (flotante, encapsulado/protegido).
  - `estado` (texto, inicia siempre en "Disponible").
- **Métodos:**
  - `precio_base (Getter y Setter)`: El setter debe validar que el valor no sea negativo. Si es negativo, lanza `ConcesionariaError`.
  - `calcular_precio_final()`: Método **abstracto**. Obliga a las clases hijas a definir su propia fórmula.
  - `__str__()`: Devuelve la descripción del vehículo y su estado.

3.3. 3. Clases Concretas (Hijas)

Clase: `Auto` (Hereda de Vehiculo)

- Implementa `calcular_precio_final()`:
- **Fórmula:** Precio Base + 10 % de Comisión + 5 % de Impuestos.

Clase: `Camioneta` (Hereda de Vehiculo)

- Implementa `calcular_precio_final()`:
- **Fórmula:** Precio Base + 15 % de Comisión + 10 % de Impuestos.

3.4. 4. Clase Controladora: `Concesionaria`

Esta clase administra la lógica del negocio.

- **Atributo:** `inventario` (una lista vacía al inicio).
- **Método** `agregar_vehiculo(self, vehiculo)`:
  - Recibe un objeto (Auto o Camioneta).
  - Valida que el VIN no exista ya en la lista.
  - Lo agrega a la lista `inventario`.

- **Método** `vender_vehiculo(self, vin)`:
  - Busca el vehículo por VIN.
  - Si lo encuentra y está "Disponible": Cambia su estado a "Vendido.eiimprime el precio final calculado.
  - Si ya estaba "Vendido": Lanza `ConcesionariaError`.
  - Si no lo encuentra: Imprime un mensaje de error.

4. Instrucciones Paso a Paso

4.1. Paso 1: Estructura Base

Copia el siguiente esqueleto en tu editor. Tu tarea es rellenar todos los espacios que dicen `pass` o `TODO` con la lógica correcta descrita arriba.

```
from abc import ABC, abstractmethod

# --- EXCEPCION PERSONALIZADA ---
class ConcesionariaError(Exception):
    pass

# --- CLASE PADRE (ABSTRACTA) ---
class Vehiculo(ABC):
    def __init__(self, vin, marca, modelo, precio):
        self.vin = vin
        self.marca = marca
        self.modelo = modelo
        # TODO: Asignar el precio usando el setter (self.
   precio_base = precio)
        self._precio_base = precio
        self.estado = "Disponible"

    @property
    def precio_base(self):
        return self._precio_base

    @precio_base.setter
    def precio_base(self, valor):
        # TODO: Validar si 'valor' < 0. Si si, raise
   ConcesionariaError.
        # Si no, asignar a self._precio_base
        pass

    @abstractmethod
    def calcular_precio_final(self):
        pass

    def __str__(self):
        return f"[{self.estado}] {self.marca} {self.modelo} ({
   self.vin})"

# --- CLASES HIJAS ---
class Auto(Vehiculo):
    def calcular_precio_final(self):
        # TODO: Retornar precio_base + 10% + 5%
        pass

class Camioneta(Vehiculo):
    def calcular_precio_final(self):
        # TODO: Retornar precio_base + 15% + 10%
```

```
        pass

# --- CONTROLADOR ---
class Concesionaria:
    def __init__(self):
        self.inventario = []

    def agregar_vehiculo(self, nuevo_vehiculo):
        # TODO: Verificar si el VIN ya existe en self.inventario
        # TODO: Si no existe, hacer append a la lista
        pass

    def vender_vehiculo(self, vin_a_vender):
        # TODO: Recorrer la lista buscando el VIN
        # TODO: Si lo encuentra y estado == "Disponible":
        #       1. Cambiar estado a "Vendido"
        #       2. Calcular precio final (usando el metodo del
   objeto)
        #       3. Imprimir ticket de venta
        # TODO: Si estado == "Vendido", raise ConcesionariaError
        pass
```

4.2. Paso 2: El Menú Principal (Main)

Fuera de las clases, crea una función `main()` que contenga un bucle infinito (`while True`) con las siguientes opciones:

1. **Registrar Auto:** Pide datos (usa `try-except` para validar números y tu error de precio negativo) y llama a `agregar_vehiculo`.
2. **Registrar Camioneta:** Similar al anterior, pero crea un objeto Camioneta.
3. **Ver Inventario:** Recorre la lista de la concesionaria e imprime cada objeto.
4. **Vender Vehículo:** Pide un VIN y llama a `vender_vehiculo` (usa `try-except` para capturar errores de venta).
5. **Salir.**