🧿 Proyecto POO – Sistema de Gestión de Vehículos y Empleados

TRABAJO REALIZADO POR:

Juan Pablo Gonzalez Trujillo

Descripción general del trabajo

Este proyecto implementa la gestión empresarial de activos empleados y vehiculos en Python aplicando los cuatro pilares de la Programación Orientada a Objetos (POO): abstracción, herencia, encapsulamiento y polimorfismo. El programa permite registrar vehículos (carros, motos y camiones) y empleados (conductores y mecánicos), además de interactuar entre ellos como asignando un conductor a un vehículo y registrar mantenimientos realizados por los mecánicos.

Estructura del Proyecto

```
src/
├─ Codigo.py            #Código completo para su ejecución por si aparecen fallos
├─ main.py
├─ vehiculos/
│  ├─ __init__.py       # Clase base abstracta Vehiculos
│  ├─ carros.py         # Subclase Carro
│  ├─ motos.py          # Subclase Moto
│  └─ camiones.py       # Subclase Camion
└─ empleados/
   ├─ __init__.py       # Clase base abstracta Empleados
   ├─ conductores.py    # Subclase Conductor
   └─ mecanicos.py      # Subclase Mecanico + registro de mantenimientos.


El programa realiza los siguientes pasos:

1. Crea empleados (conductores y mecánicos).
2. Crea vehículos (carros, motos y camiones).
3. Asigna conductores a algunos vehículos (los no asignados aparecen como "sin
asignar").
4. Muestra descripciones y costos de mantenimiento (polimorfismo).
5. Permite conducir y registrar mantenimientos.
6. Verifica el estado de los empleados (activar/desactivar).



## ¿Donde se encuentran los pilares de POO?

## Abstracción:
- Se aplica en las clases "Vehiculos" y Empleados", que son clases abstractas.
- Contiene métodos con "@abstractmethod" ("tipo()", "costo_mantenimiento()",
```

```
"rol()").

### Herencia
- "Carro", "Moto", "Camion" heredan de la clase padre "Vehiculos".
- "Conductor" y "Mecanico" heredan de la clase padre "Empleados".
- Gracias a esto sus subclases comparten atributos y métodos.

### Encapsulamiento
- Los atributos internos "__placa", "__kilometraje", "__activo" son de caracter
privados.
- Solo se accede a ellos a través de métodos públicos como "get_" y "add_km()"
evitando modificar los datos directamente.

### Polimorfismo
- El método "costo_mantenimiento()" se usa de forma diferente dependiendo de cada
subclase de "Vehiculo".

-- Interacción entre Objetos --

El proyecto demuestra la relación entre clases a través de la interacción:

- Un conductor puede ser asignado a un vehículo y conducirlo
("asignar_conductor()", "conducir()").
- Un mecánico puede registrar mantenimientos sobre los vehículos
(`registrar_mantenimiento()`).
- Todos los mantenimientos se establecen en la lista "_REGISTROS" y se pueden
consultar por placa.

----------------------------------------

----Ejemplo de Ejecución-----

Salida resumida del programa:
```

=== LISTA DE EMPLEADOS === Soy Jesus Ariel González, Conductor Soy Juan Gonzalez, Mecánico ...

=== LISTA DE VEHÍCULOS === Carro OMG650 | km=23500 | Jesus Ariel González Moto PMU79S | km=4420 | Brayan Esteban Rojas Camion AWH430 | km=75656 | Marianita Lopez

=== COSTOS DE MANTENIMIENTO === Carro -> 1500000 Moto -> 400000 Camion -> 3000000

=== ESTADO DE EMPLEADOS === Juan Diego Trujillo activo? True Jhon Erick Puentes activo? True ...

```
## Algunos problemas durante el desarrollo del programa:

Primeramente lo más tardado y dificil fue la lógica del ejercicio, acomodar todos
los datos requeridos y pensar como programar
este ejercicio fue el reto principal.
Durante el desarrollo tuve dificultades para saber usar y determinar correctamente
las clases abstractas y cómo aplicar el encapsulamiento con sus atributos
```

privados, en este punto me apoyé bastante de la IA para aclarar mis propias ideas, pues
la herencia y poliformismo las tenia medianamente claras.
También aparecieron errores de ejecución por los modulos del proyecto puesto que este es el segundo código que realizo de esta forma, sin embargo, es mucho más organizado y me permite encontrar datos o errores facilmente.
Otro de los problemas que tuve fueron en los atributos privados pues me salian bastantes errores porque no los instanciaba de forma correcta, y por último, un error tonto pero que solucioné rápido fue que no asignaba a los metodos abstractos su respectiva salida (si es string o float), salia un error que resolví rápidamente con ayuda de la IA.