Clase 3: Herencia en Programación Orientada a Objetos (POO) con Python 🐍

Concepto de Herencia 🔗

La herencia es un principio fundamental de la programación orientada a objetos que permite crear nuevas clases (clases hijas) a partir de una clase existente (clase padre o superclase). Esto permite reutilizar el código y extender funcionalidades de manera más sencilla. Las clases hijas heredan atributos y métodos de la clase padre.

Ventajas de la Herencia 🌟

- **Reutilización de código**: Las clases hijas pueden usar el código de la clase padre.
- **Extensibilidad**: Se pueden agregar nuevas funcionalidades sin modificar la clase padre.
- **Organización**: Mejora la estructura del código al agrupar funcionalidades relacionadas.

Ejemplo Práctico de Herencia 🚗🚲🚚

Vamos a construir un sistema de gestión de vehículos en una concesionaria. Tendremos una clase padre llamada `Vehículo`, y las clases hijas `Auto`, `Bicicleta`, y `Camión`.

1. Clase Padre: Vehículo 🚗

```
class Vehículo:
    def __init__(self, marca, modelo, precio):
        self.marca = marca
        self.modelo = modelo
        self.precio = precio
        self.disponible = True  # Un atributo para indicar di
sponibilidad

    def vender(self):
        if self.disponible:
            self.disponible = False
            print(f"El {self.marca} ha sido vendido.")
        else:
            print(f"El {self.marca} no está disponible para l
a venta.")

    def esta_disponible(self):
        return self.disponible

    def obtener_precio(self):
        return self.precio

    def iniciar_funcionamiento(self):
        raise NotImplementedError("Este método debe ser imple
mentado por la subclase.")

    def detener_funcionamiento(self):
```

```
        raise NotImplementedError("Este método debe ser imple
mentado por la subclase.")
```

2. Clases Hijas: Auto, Bicicleta y Camión 🚲🚚

Clase Auto 🚗

```
class Auto(Vehículo):
    def iniciar_funcionamiento(self):
        if self.disponible:
            print(f"El motor del {self.marca} está en march
a.")
        else:
            print(f"El {self.marca} no está disponible para i
niciar.")

    def detener_funcionamiento(self):
        if self.disponible:
            print(f"El motor del {self.marca} se ha detenid
o.")
        else:
            print(f"El {self.marca} no está disponible para d
etener.")
```

Clase Bicicleta 🚲

```
class Bicicleta(Vehículo):
    def iniciar_funcionamiento(self):
        if self.disponible:
            print(f"El motor de la bicicleta {self.marca} est
á en marcha.")
        else:
            print(f"La bicicleta {self.marca} no está disponi
ble para iniciar.")
```

```
    def detener_funcionamiento(self):
        if self.disponible:
            print(f"El motor de la bicicleta {self.marca} se
ha detenido.")
        else:
            print(f"La bicicleta {self.marca} no está disponi
ble para detener.")
```

Clase Camión 🚚

```
class Camión(Vehículo):
    def iniciar_funcionamiento(self):
        if self.disponible:
            print(f"El motor del camión {self.marca} está en
marcha.")
        else:
            print(f"El camión {self.marca} no está disponible
para iniciar.")

    def detener_funcionamiento(self):
        if self.disponible:
            print(f"El motor del camión {self.marca} se ha de
tenido.")
        else:
            print(f"El camión {self.marca} no está disponible
para detener.")
```

---

3. Implementación en la Concesionaria 🏢

Vamos a crear una clase `Concesionaria` que gestionará el inventario de vehículos y la compra por parte de clientes.

```
class Concesionaria:
    def __init__(self):
        self.inventario = []
```

```
        self.clientes = []

    def agregar_auto(self, auto):
        self.inventario.append(auto)

    def registrar_cliente(self, cliente):
        self.clientes.append(cliente)

    def mostrar_autos_disponibles(self):
        for auto in self.inventario:
            if auto.esta_disponible():
                print(f"{auto.marca} {auto.modelo} está dispo
nible a ${auto.obtener_precio()}.")

    def vender_auto(self, cliente, auto):
        if auto.esta_disponible():
            auto.vender()
            print(f"{cliente} ha comprado el {auto.marca}.")
        else:
            print(f"Lo siento, {cliente}, el {auto.marca} no
está disponible.")
```

4. Ejercicio Práctico 📚

Ejercicio 1: Crear una Concesionaria y Vender Autos

Instrucciones:

1. Crea tres instancias de `Auto` y dos instancias de `Bicicleta`.
2. Agrega estos vehículos al inventario de la concesionaria.
3. Muestra los autos disponibles.
4. Un cliente intenta comprar un auto y un cliente intenta comprar una bicicleta.

Código de Solución:

```
# Crear autos
auto1 = Auto("Toyota", "Corolla", 20000)
auto2 = Auto("Honda", "Civic", 22000)
auto3 = Auto("Ford", "Mustang", 30000)

# Crear bicicletas
bici1 = Bicicleta("Giant", "Escape", 500)
bici2 = Bicicleta("Trek", "FX", 700)

# Crear concesionaria
concesionaria = Concesionaria()

# Agregar vehículos al inventario
concesionaria.agregar_auto(auto1)
concesionaria.agregar_auto(auto2)
concesionaria.agregar_auto(auto3)
concesionaria.agregar_auto(bici1)
concesionaria.agregar_auto(bici2)

# Mostrar autos disponibles
concesionaria.mostrar_autos_disponibles()

# Cliente intenta comprar un auto
concesionaria.vender_auto("Juan", auto1)

# Mostrar autos disponibles después de la compra
concesionaria.mostrar_autos_disponibles()

# Cliente intenta comprar una bicicleta
concesionaria.vender_auto("Ana", bici1)
```

5. Conclusiones 📌

La herencia en POO permite organizar y extender la funcionalidad de las clases de manera efectiva, facilitando la creación de sistemas más complejos con un código

más limpio y reutilizable. 🛠️ A través de este ejemplo, hemos aplicado la herencia para modelar una concesionaria de vehículos, mostrando cómo las clases hijas pueden personalizar el comportamiento de la clase padre.