Clase 4: Programación Orientada a Objetos: Objetos Heredados 🚀

1. Conceptos Fundamentales

1.1 Clases y Objetos

- **Clase**: Es un plano o plantilla que define las características y comportamientos comunes de un grupo de objetos.
- **Objeto**: Es una instancia de una clase que tiene atributos y métodos.

1.2 Herencia

La **herencia** permite que una clase (hija) herede propiedades y métodos de otra clase (padre). Esto fomenta la reutilización de código y la creación de jerarquías en las clases.

1.3 Clases Base y Clases Derivadas

- **Clase Base**: Clase de la que se heredan las propiedades y métodos (por ejemplo, `Vehículo`).

- **Clase Derivada**: Clase que hereda de la clase base (por ejemplo, `Bicicleta` y `Camión`).

2. Ejemplo Práctico

A continuación, implementaremos un ejemplo que involucra la creación de un sistema para una concesionaria de vehículos, que incluye las clases `Vehículo`, `Bicicleta`, `Camión`, `Comprador`, y `Concesionaria`.

2.1 Clases Definidas

```
# Clase base
class Vehículo:
    def __init__(self, marca, modelo, precio):
        self.marca = marca
        self.modelo = modelo
        self.precio = precio
        self.disponible = True

    def mostrar_info(self):
        return f"Marca: {self.marca}, Modelo: {self.modelo},
Precio: {self.precio}, Disponible: {self.disponible}"

# Clase derivada Bicicleta
class Bicicleta(Vehículo):
    def __init__(self, marca, modelo, precio):
        super().__init__(marca, modelo, precio)

    def en_marcha(self):
        return "La bicicleta está en marcha."

# Clase derivada Camión
class Camión(Vehículo):
    def __init__(self, marca, modelo, precio):
        super().__init__(marca, modelo, precio)
```

```
    def motor_en_marcha(self):
        return "El motor del camión está en marcha."

# Clase Comprador
class Comprador:
    def __init__(self, nombre):
        self.nombre = nombre
        self.vehículos = []

    def comprar_vehículo(self, vehículo):
        if vehículo.disponible:
            vehículo.disponible = False
            self.vehículos.append(vehículo)
            print(f"{self.nombre} ha comprado el {vehículo.ma
rca}.")
        else:
            print("Lo siento, el vehículo no está disponibl
e.")

# Clase Concesionaria
class Concesionaria:
    def __init__(self):
        self.inventario = []
        self.clientes = []

    def añadir_vehículo(self, vehículo):
        self.inventario.append(vehículo)
        print(f"{vehículo.marca} ha sido añadido al inventari
o.")

    def registrar_cliente(self, comprador):
        self.clientes.append(comprador)
        print(f"El cliente {comprador.nombre} ha sido registr
ado.")

    def mostrar_vehículos_disponibles(self):
```

```
        print("Vehículos disponibles:")
        for vehículo in self.inventario:
            if vehículo.disponible:
                print(vehículo.mostrar_info())
```

2.2 Creando Instancias y Ejemplos

```
# Creación de la concesionaria
concesionaria = Concesionaria()

# Añadiendo vehículos
bicicleta1 = Bicicleta("Giant", "Escape 3", 500)
camión1 = Camión("Ford", "F-150", 30000)

concesionaria.añadir_vehículo(bicicleta1)
concesionaria.añadir_vehículo(camión1)

# Registrando un comprador
comprador1 = Comprador("Zeus")
concesionaria.registrar_cliente(comprador1)

# Mostrando vehículos disponibles
concesionaria.mostrar_vehículos_disponibles()

# Comprando un vehículo
comprador1.comprar_vehículo(bicicleta1)
concesionaria.mostrar_vehículos_disponibles()
```

3. Ejercicios Prácticos

Ejercicio 1: Crear una Clase `Auto`

Crea una clase `Auto` que herede de `Vehículo`. Asegúrate de incluir un método que indique si el auto tiene el motor encendido.

Solución:

```
class Auto(Vehículo):
    def __init__(self, marca, modelo, precio):
        super().__init__(marca, modelo, precio)

    def motor_encendido(self):
        return "El motor del auto está encendido."
```

Ejercicio 2: Añadir Métodos a Comprador

Modifica la clase `Comprador` para que incluya un método que muestre todos los vehículos que ha comprado.

Solución:

```
class Comprador:
    # Métodos existentes...

    def mostrar_vehículos_comprados(self):
        for vehículo in self.vehículos:
            print(vehículo.mostrar_info())
```

Ejercicio 3: Consultar Disponibilidad

Crea un método en la clase `Concesionaria` que consulte la disponibilidad de un vehículo específico.

Solución:

```
class Concesionaria:
    # Métodos existentes...

    def consultar_disponibilidad(self, vehículo):
        return "Disponible" if vehículo.disponible else "No d
isponible"
```

4. Conclusiones

La programación orientada a objetos, a través de la herencia, permite crear aplicaciones más organizadas y reutilizables. Entender y aplicar estos conceptos es fundamental para el desarrollo de software moderno. ¡Sigue practicando! 💻✨