La Programación Orientada a Objetos (POO)

Es un paradigma de programación que se basa en el concepto de "objetos", los cuales pueden contener tanto datos como funciones que operan en esos datos. Las POO manejan unos conceptos fundamentales los cuales son los siguientes:

- Clases y Objetos: En la POO, una clase es una plantilla para crear objetos.
- La herencia es un mecanismo que permite que una clase (objeto) adquiera las propiedades y comportamientos de otra clase.
- Abstracción: La POO permite modelar entidades del mundo real como objetos, abstrayendo los detalles irrelevantes y centrándose en las características esenciales de esos objetos.
- Encapsulación: Este concepto se refiere a la combinación de datos y funciones en un solo componente, el objeto.

Ejemplos:

Imagina que estás diseñando un sistema para gestionar una flota de vehículos. Podrías tener una clase Vehículo con atributos como marca, modelo, año, etc. Luego, podrías tener subclases como Coche, Camión, Motocicleta, cada una con sus propios atributos y métodos específicos. Esto te permitiría gestionar diferentes tipos de vehículos de manera eficiente y modular.

```python
class Vehiculo:
    def __init__(self, marca, modelo, year):
        self.marca = marca
        self.modelo = modelo
        self.year = year

class Coche(Vehiculo):
    def __init__(self, marca, modelo, year, tipo_motor):
        super().__init__(marca, modelo, year)
        self.tipo_motor = tipo_motor

class Camion(Vehiculo):
    def __init__(self, marca, modelo, year, capacidad_carga):
        super().__init__(marca, modelo, year)
        self.capacidad_carga = capacidad_carga

class Motocicleta(Vehiculo):
    def __init__(self, marca, modelo, year, cilindrada):
        super().__init__(marca, modelo, year)
        self.cilindrada = cilindra
```

En un sistema bancario, podrías tener clases como `Cliente`, `CuentaBancaria`, `Transacción`, etc. Cada cliente sería un objeto de la clase `Cliente`, con atributos como nombre, dirección, número de identificación, etc. Las cuentas bancarias también podrían ser objetos con métodos para depositar, retirar dinero, etc.

```python
class Cliente:
    def __init__(self, nombre, direccion, id_cliente):
        self.nombre = nombre
        self.direccion = direccion
        self.id_cliente = id_cliente

class CuentaBancaria:
    def __init__(self, cliente, saldo):
        self.cliente = cliente
        self.saldo = saldo

    def depositar(self, cantidad):
        self.saldo += cantidad

    def retirar(self, cantidad):
        if cantidad <= self.saldo:
            self.saldo -= cantidad
        else:
            print("Fondos insuficientes")

class Transaccion:
    def __init__(self, cuenta_origen, cuenta_destino, cantidad):
        self.cuenta_origen = cuenta_origen
        self.cuenta_destino = cuenta_destino
        self.cantidad = cantidad
```

En un juego de video, podrías tener clases como `Jugador`, `Enemigo`, `Arma`, etc. Cada jugador y enemigo serían objetos con atributos como salud, velocidad, puntos de ataque, etc. Las armas también podrían ser objetos con métodos para disparar, recargar, etc.

```python
class Jugador:
    def __init__(self, nombre, salud, velocidad):
        self.nombre = nombre
        self.salud = salud
        self.velocidad = velocidad

    def recibir_danio(self, cantidad):
        self.salud -= cantidad

class Enemigo:
    def __init__(self, tipo, salud, puntos_ataque):
        self.tipo = tipo
        self.salud = salud
        self.puntos_ataque = puntos_ataque

    def atacar_jugador(self, jugador):
        jugador.recibir_danio(self.puntos_ataque)

class Arma:
    def __init__(self, nombre, danio):
        self.nombre = nombre
        self.danio = danio

    def disparar(self, objetivo):
        objetivo.recibir_danio(self.danio)
```