ChatGPT

Paradigma de Orientación a Objetos

La **programación orientada a objetos (POO)** es un paradigma que modela el software en términos de "objetos" con datos y funcionalidades propias [1]. A diferencia de la programación estructurada, que se centra en subrutinas y secuencias de instrucciones, la POO agrupa datos (atributos) y comportamientos (métodos) en unidades llamadas *clases* y *objetos*. Este enfoque imita el mundo real: cada entidad relevante (por ejemplo, un "libro" o un "usuario" en un sistema bibliotecario) se convierte en un objeto con sus propiedades y operaciones [2]. De este modo se facilita la reutilización y el mantenimiento del código, pues cada clase actúa como un plano reutilizable y los objetos interactúan entre sí mediante mensajes.

Diferencias con la programación estructurada: A modo de comparación, en la programación estructurada los programas se dividen principalmente en funciones o procedimientos separados de los datos. En cambio, en POO los datos y funciones se combinan en objetos. Por ejemplo, «La POO se centra en clases y objetos» mientras que la programación estructurada utiliza procedimientos independientes [3]. Además, la POO incorpora herencia y polimorfismo (heredar métodos entre clases), conceptos ausentes en el enfoque estructurado [4]. Otro rasgo clave es el *encapsulamiento*: en POO los datos internos de un objeto quedan ocultos y sólo pueden modificarse mediante sus métodos públicos, formando una "cápsula" protectora, algo que la programación estructurada no ofrece tan explícitamente [5] [6]. En resumen, POO privilegia la organización modular basada en objetos mientras que la estructurada se basa en lógica secuencial y llamadas a procedimientos.

Ejemplo cotidiano: La POO se inspira en cómo percibimos el mundo real [2]. Por ejemplo, en un sistema de gestión bibliotecaria podríamos tener una clase `Libro` con atributos como `título`, `autor` y métodos como `prestar()` o `devolver()`. Cada libro concreto ("*Don Quijote*" o "*Cien años de soledad*") sería un **objeto** (o instancia) de esa clase, con valores concretos para sus atributos. Así, el paradigma facilita pensar en términos de entidades reconocibles (libros, usuarios, transacciones) en lugar de solo funciones sueltas.

Clases, Objetos e Instancias

- **Clase:** Es un *prototipo* o *plantilla genérica* que define un tipo de objeto. Una clase especifica qué atributos (datos) y métodos (funciones) comparten todos sus objetos [7] [8]. En otras palabras, la clase describe el *estado* (qué datos puede tener) y el *comportamiento* (qué puede hacer) de los objetos que crea [7] [8]. Por ejemplo, una clase `Animal` podría declarar atributos como `nombre` y `edad`, y métodos como `comer()` o `dormir()` para todos los animales.

- **Objeto (instancia):** Es cada realización concreta de una clase. Cuando se crea un objeto, se asignan valores específicos a los atributos definidos por su clase. En POO, «un objeto es el resultado de la instanciación de una clase» [9] [10]. Así, un objeto hereda la estructura de su clase pero tiene su propia identidad y sus datos particulares. Siguiendo el ejemplo anterior, `Animal("Paco", 3)` puede ser un objeto de la clase `Animal`. Cada objeto tiene *identidad*, *estado* (valores concretos de atributos) y *comportamiento* (métodos) [10].

- **Instancia:** Es sinónimo de objeto. Se dice que crear o instanciar un objeto es "sacar del molde" (clase) una entidad concreta. En terminología: *instancia* de una clase = objeto de esa clase [11].

Por ejemplo, en Python podemos definir y usar una clase así:

```
class Animal:
    def __init__(self, nombre, edad):
        self.nombre = nombre   # Atributo de la clase
        self.edad = edad       # Atributo de la clase

    def describir(self):
        # Método que muestra el estado actual del objeto
        print(f"Animal {self.nombre}, Edad: {self.edad} años")

    def envejecer(self):
        # Método que modifica el estado (edad) del objeto
        self.edad += 1

# Creamos dos objetos (instancias) de la clase Animal
gato = Animal("Paco", 3)
perro = Animal("Pancho", 4)
gato.describir()   # Comportamiento: muestra "Animal Paco, Edad: 3 años"
perro.describir()  # "Animal Pancho, Edad: 4 años"
```

En este ejemplo, `Animal` es la clase, y `gato` y `perro` son objetos (instancias) de esa clase. Ambos objetos comparten los mismos atributos (`nombre`, `edad`) y métodos (`describir`, `envejecer`), pero cada uno mantiene su **estado** independiente (por ejemplo, el nombre y la edad propios).

Atributos y Estado

- **Atributo:** Es una variable definida dentro de una clase que representa una *característica* o propiedad común de los objetos de ese tipo. Cada atributo tiene un nombre y un tipo (en lenguajes tipados) y define qué valores puede tomar. El conjunto de atributos de una clase define la estructura básica de sus objetos [12]. Por ejemplo, la clase `Animal` anterior define los atributos `nombre` y `edad`.

- **Estado de un objeto:** Se refiere a los valores actuales de los atributos de ese objeto en un momento dado. Según la teoría, "el conjunto de valores de los campos define el estado del objeto" [12]. Es decir, aunque `Animal` define qué atributos existen, cada objeto tiene su estado particular según cómo se hayan inicializado o modificado esos atributos. En el ejemplo anterior, el objeto `gato` tiene estado `{nombre: "Paco", edad: 3}`, mientras que `perro` tiene `{nombre: "Pancho", edad: 4}` [13] [14].

Diferencia atributo–estado: Un *atributo* es la definición de la propiedad (por ejemplo, "edad"), mientras que el *estado* es el valor concreto que ese atributo tiene en un objeto específico (por ejemplo, `edad = 3` en el objeto `gato`). Podemos ilustrarlo con el código:

```python
# Atributos definidos en la clase:
class Animal:
    def __init__(self, nombre, edad):
        self.nombre = nombre  # atributo 'nombre'
        self.edad = edad      # atributo 'edad'

gato = Animal("Paco", 3)
print(gato.nombre, gato.edad)  # Muestra el estado actual: Paco 3

# Modificamos el estado del objeto gato
gato.envejecer()
print(gato.edad)  # Ahora edad es 4
```

En este fragmento, `nombre` y `edad` son atributos de la clase `Animal`. El objeto `gato` posee un *estado* inicial donde `nombre="Paco"` y `edad=3`. Al llamar a `gato.envejecer()`, el método modifica su atributo `edad`, cambiando así el estado interno del objeto. Cada objeto mantiene sus propios valores, aún cuando comparten la definición de atributos.

Métodos y Comportamiento

- **Método:** Es una función o procedimiento definido dentro de una clase, que describe un *comportamiento* común a sus objetos. Los métodos representan las operaciones que los objetos pueden realizar o que pueden realizar sobre ellos. Por ejemplo, en `Animal` definimos el método `describir()` y `envejecer()`, que describen cómo actúa cualquier objeto Animal. Formalmente, "un método es un comportamiento o conducta de un objeto" [15]. Se llaman a través de objetos (por ejemplo, `gato.describir()`) para ejecutar esa operación en particular.

- **Comportamiento de un objeto:** Es el resultado de ejecutar sus métodos. En la práctica, el comportamiento es cómo el objeto "responde" o actúa ante una operación. Cada vez que se invoca un método sobre un objeto, éste realiza una acción (por ejemplo, mostrar datos o modificar su estado). En otras palabras, el *comportamiento* de un objeto se determina por los métodos definidos en su clase, pero se ejecuta en el contexto de la instancia concreta. Por tanto, **método** es la definición (en la clase) y **comportamiento** es la ejecución de ese método en el objeto.

Siguiendo con nuestro ejemplo, los métodos `describir()` y `envejecer()` dan comportamientos al objeto `gato`. Cuando hacemos `gato.describir()`, el objeto `gato` utiliza el código de `describir` para imprimir su estado actual. Cuando hacemos `gato.envejecer()`, el objeto `gato` ejecuta ese método y su **estado** (`edad`) cambia. Así, el mismo método produce distintos comportamientos dependiendo del objeto (por ejemplo, si lo llamamos en `gato` o en `perro`).

```python
# Ejemplo de métodos y comportamiento:
gato.describir()   # Comportamiento: el gato imprime su estado actual
# Salida: "Animal Paco, Edad: 3 años"

gato.envejecer()   # Comportamiento: el gato cambia su estado interno
gato.describir()   # Ahora imprime "Animal Paco, Edad: 4 años"
```

En resumen, un **método** es el procedimiento definido en la clase, y el **comportamiento** es lo que ocurre en tiempo de ejecución cuando ese método actúa sobre un objeto concreto.

Principios Básicos: Abstracción y Encapsulamiento

En la POO existen varios principios fundamentales, entre ellos la **abstracción** y el **encapsulamiento**:

- **Abstracción:** Consiste en identificar y modelar sólo las características esenciales de un objeto, ignorando los detalles irrelevantes. Según la teoría, "la abstracción es un proceso de... reconocer y enfocarse en las características importantes..., y filtrar o ignorar las particularidades no esenciales" [16]. En la práctica, al definir una clase nos abstraemos de los detalles internos de su implementación; solo exponemos los atributos y métodos relevantes para quien use la clase. Por ejemplo, al usar un objeto `Coche`, no necesitamos saber cómo funciona el motor internamente, basta con métodos de alto nivel como `arrancar()` o `frenar()`. La abstracción en POO permite manejar sistemas complejos mediante interfaces sencillas: cada clase es una abstracción de un concepto del problema, mostrando sólo lo necesario.

- **Encapsulamiento:** Es la propiedad que restringe el acceso directo al estado interno de un objeto, obligando a usar sus métodos para leer o modificar sus datos. Se suele decir que los datos quedan "ocultos" dentro del objeto. Formalmente, "el encapsulamiento consiste en agrupar en una clase las características (atributos) con acceso privado y los comportamientos (métodos) con acceso público" [6]. De este modo, el estado interno sólo puede cambiarse a través de los métodos definidos, protegiendo los datos de usos indebidos.

Por ejemplo, consideremos una clase `CuentaBancaria` con encapsulamiento en Python:

```
class CuentaBancaria:
    def __init__(self, saldo):
        self.__saldo = saldo   # Atributo privado (nombre con __)

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

    def retirar(self, cantidad):
        if cantidad <= self.__saldo:
            self.__saldo -= cantidad

    def mostrar_saldo(self):
        print(f"Saldo: {self.__saldo}")

cuenta = CuentaBancaria(100)
cuenta.depositar(50)
cuenta.retirar(30)
cuenta.mostrar_saldo()  # Salida: "Saldo: 120"
```

Aquí, `__saldo` es un atributo privado (por convención de Python), accesible sólo dentro de la clase. Los métodos públicos `depositar` y `retirar` permiten modificar el saldo de forma controlada, y `mostrar_saldo` permite consultarlo. Nadie puede cambiar el saldo arbitrariamente sin pasar por

estos métodos. Este ejemplo ilustra cómo el encapsulamiento protege la *integridad de los datos* y oculta los detalles internos del objeto, exponiendo únicamente la interfaz pública necesaria [6].

En conjunto, la abstracción y el encapsulamiento son pilares que ayudan a escribir código más limpio, modular y robusto en POO. La abstracción simplifica el diseño concentrándose en el *qué* en lugar del *cómo* [16], y el encapsulamiento garantiza que los objetos gestionen su propio estado de forma segura [6].

En síntesis: La programación orientada a objetos organiza el software en clases y objetos, diferenciando claramente el concepto de plantilla (clase) del objeto concreto (instancia). Cada objeto tiene atributos (definidos en la clase) que conforman su estado, y métodos que implementan sus comportamientos [7] [14]. A diferencia de la programación estructurada, la POO agrupa datos y funciones en unidades coherentes (objetos), favorece la abstracción de detalles y protege la información interna mediante el encapsulamiento [3] [6]. Estos principios permiten crear sistemas más modulares, reutilizables y fáciles de mantener, reflejando de forma intuitiva la realidad en el diseño del software [1] [5].

Fuentes: Conceptos extraídos de literatura especializada en POO [1] [8] [16] [7] [6] [3] [9] [10], junto con ejemplos propios. Cada afirmación técnica está respaldada por las referencias citadas.

---

[1] [2] [8] [13] ¿Qué es la Programación Orientada a Objetos?
https://profile.es/blog/que-es-la-programacion-orientada-a-objetos/

[3] [4] [5] Programación Orientada a Objetos vs Programación Estructurada - EducaciónIT
https://blog.educacionit.com/programacion-orientada-a-objetos-vs-programacion-estructurada/

[6] [7] [10] [11] [12] [15] [16] Paradigma de la programación orientada a objetos
https://ferestrepoca.github.io/paradigmas-de-programacion/poo/poo_teoria/concepts.html

[9] [14] Objeto (programación) - Wikipedia, la enciclopedia libre
https://es.wikipedia.org/wiki/Objeto_(programaci%C3%B3n)