“Conceptos Clave de Programación Orienta a Objetos (POO)”

Obed Ríos López

Instituto Universitario del Sureste
Ingeniería en Tecnologías de la Información y Comunicación
Análisis y Desarrollo de Sistemas
Ing. Nicolas Navarrete

Mérida Yucatán, México
26 de febrero de 2025

Programación Orientada a Objetos (POO)

¿Qué es?:

Es un paradigma de programación, en palabras simples, es un modelo o un estilo de programación. La POO se basa en clases y objetos y se utiliza para estructurar un programa en piezas simples y reutilizables de planos de código (clases) para crear instancias individuales de objetos.

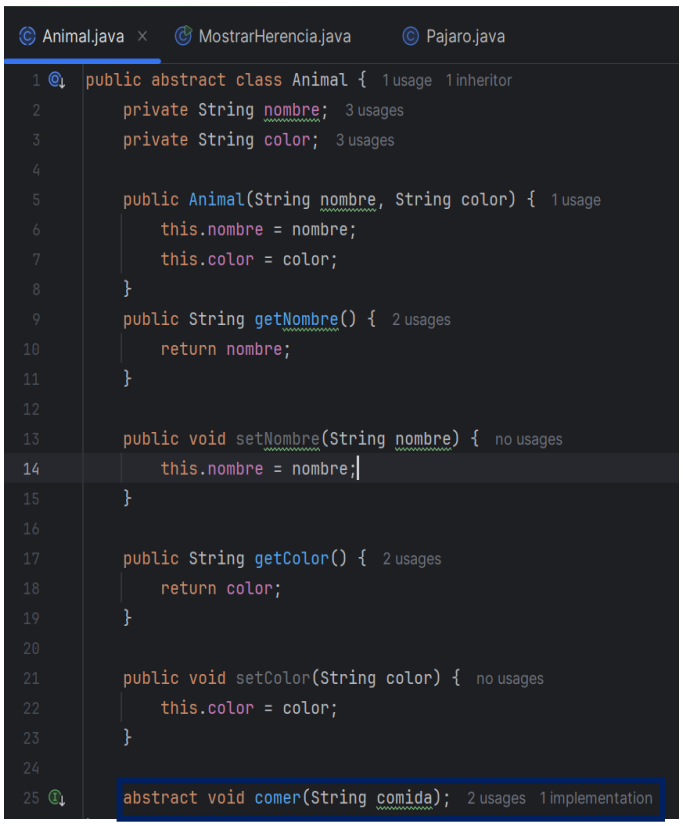
Dentro de los conceptos claves de la POO se encuentran ventajas como:

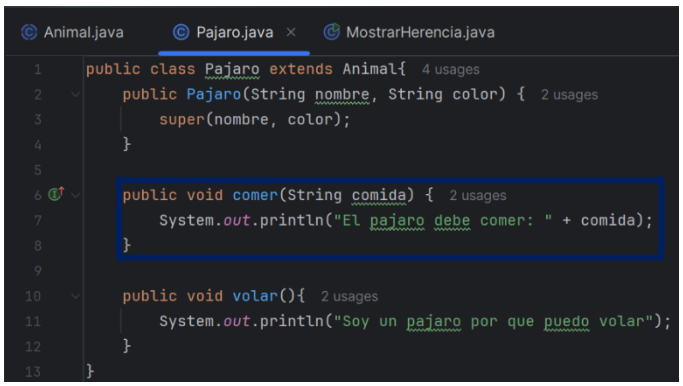
- Reutilización de código.
- Facilidad de extensibilidad.
- Promueve coherencia y consistencia.
- Abstracción y generalización.

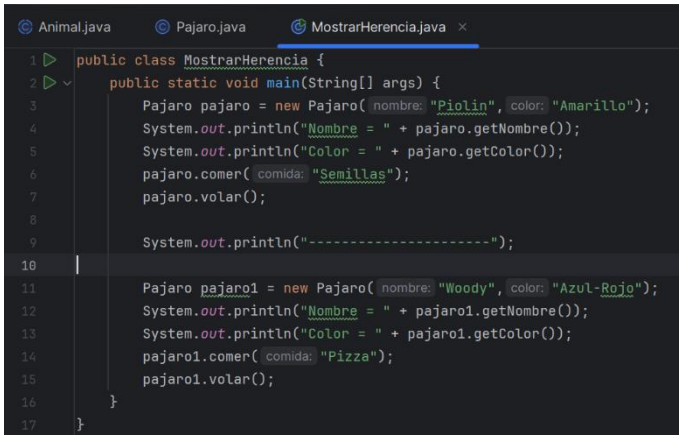
Ahora bien, programación orientada a objetos tiene 4 conceptos clave a entender: Herencia, polimorfismo, encapsulamiento y abstracción. En esta ocasión para ejemplificar cada principio usaremos el lenguaje de programación JAVA, ya que se presta perfectamente para ejemplificar la POO, mediante un programa hecho anteriormente basado en animales. Este programa incluye 4 clases, la clase padre “Animal”, las 2 clases hijas “Perro” y “Pájaro”, y una clase llamada “MostrarHerencia” que es la que incluye nuestro método “main” para correr el programa en consola. Un punto que resaltar es que en cada concepto se fue editando el programa para hacerlo más escalable, o bien para agregar datos para ejemplificar el concepto a explicar.

1.- Herencia: Esta es un propiedad de POO que permite crear clases derivadas a partir de una clase que llamamos padre. La clase hija hereda los atributos y métodos de su clase padre.

Ejemplo: Tenemos la clase padre que se llama Animal, la clase hija “Pájaro”, la clase padre tiene atributos como nombre, color y un método nombrado “Comer”. Ahora siguiendo el concepto de herencia, la clase hija (pájaro) hereda estos atributos y métodos, pero no todos los animales vuelan, entonces a esta clase también se le define su propio método especifico “volar”, para así después en la clase con el “main”, poder imprimir los valores de dichos métodos.







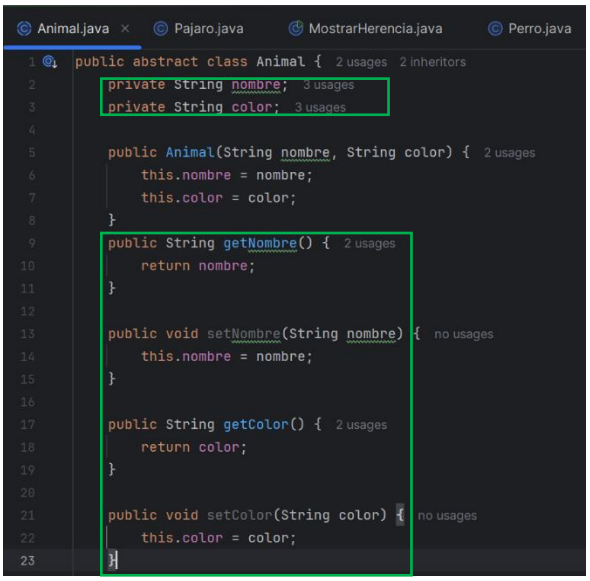
2.- Abstracción: La abstracción en POO es otro principio, su concepto teórico es que permite ocultar los detalles internos de una implementación y exponer solo las funcionalidades esenciales. Una de las ventajas principales mencionadas al inicio es que este principio facilita la reutilización del código, extensibilidad de este y mejora la organización del programa.

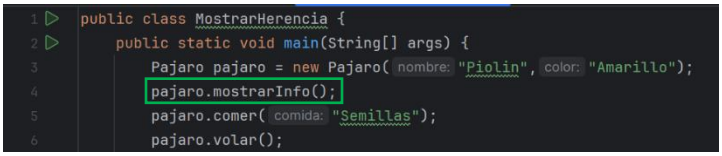
Ejemplo: Podemos usar el mismo ejemplo de arriba, pues usamos abstracción, ya que la clase “Animal” es una clase abstracta con el método “comer” pues no se implementa en la clase padre si no que cualquier clase hija que herede de “Animal”, está obligada a definir su propia

versión del método “comer”. Básicamente se define la propiedad y método común entre todos los animales (comer), pero sin especificar detalles de cómo comen. (Recuadro azul)

3.- Encapsulamiento: La función de este principio es impedir, ocultar, o “encapsular” el acceso a los datos por cualquier medio que no sean los servicios propuestos, es decir, protege los datos de una clase al restringir el acceso directo a ellos. Para acceder directamente a los atributos usamos los métodos getters y setters. La encapsulación garantiza la protección de los datos del objeto, esto evita modificaciones accidentales permitiendo controlar el como se accede y se cambian los datos dentro del objeto.

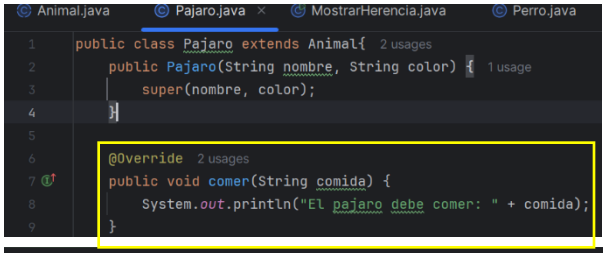
Ejemplo: Usando el mismo ejemplo, pero con algunos cambios observamos que tenemos en la clase padre el nombre de las 2 variables declaradas en tipo de dato “private”, esto permite que no se pueda modificar directamente desde fuera de la clase; más abajo usamos los métodos de getters y setters para poder acceder desde fuera. Para este caso agregamos un método en la clase padre que contenga el “getNombre” y “getColor”, así no reescribimos código en las demás clases, y en el método “main”, solo invocamos el método “mostrarInfo”. Al final el encapsulamiento se hace presente mediante los getters y setters, y también mediante los atributos “private”. (Recuadros color verde)

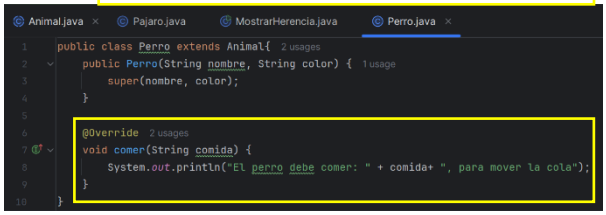




4.- Polimorfismo: El polimorfismo en POO es la capacidad que tienen los objetos de una clase en ofrecer respuesta distinta e independiente en función de los parámetros (diferentes implementaciones) utilizados durante su invocación. Básicamente es la capacidad de que un objeto adquiera distintas formas y tenga diferentes comportamientos según la clase que lo implemente. Existen dos tipos principales de polimorfismo en Java:

- Polimorfismo en tiempo de compilación (Sobrecarga de métodos): Permite implementar múltiples métodos dentro de la misma clase que usan el mismo nombre pero diferentes parámetros. Un ejemplo es una clase "calculadora" que tenga 3 métodos con el mismo nombre "suma", pero cada método tiene distintas formas de sumar (decimal o entera).
- Polimorfismo en tiempo de ejecución (Sobreescritura de métodos): Una subclase puede sobrescribir y redefinir el método de una superclase con una implementación especifica; esta opción ofrece una de las ventajas mencionadas al principio, escribir código más genérico y reutilizable y ocurre cuando se escribe "@Override".
- **Ejemplo:** Usando el ejemplo anterior, el polimorfismo que se aplica es una sobreescritura de métodos, ya que la clase "pájaro", sobrescribe el método "comer" de la clase padre, pero para verlo mejor tenemos que agregar más clases, por eso agregamos "perro", así cada clase lo implementara de forma distinta. El polimorfismo no está en la variable comida si no en la implementación de "comer", pues aunque cada subclase está obligada a usarlo, lo implementa de distinta manera, el pájaro dice una cosa mientras que el perro otra, así el método "comer" tiene distintos comportamientos dependiendo el animal. (Recuadro amarillo)





Al imprimir los valores: