**¿Qué es la Programación Orientada a Objetos?**

En el mundo de la programación, la POO es un paradigma que ha ganado una gran popularidad en los últimos años debido a su capacidad para crear aplicaciones más robustas, flexibles y fáciles de mantener. Esta metodología de desarrollo se basa en la idea de que los programas se pueden organizar como una colección de objetos interconectados, cada uno con su propio conjunto de datos y funcionalidades.

Con el paradigma de **Programación Orientado a Objetos** lo que buscamos es dejar de centrarnos en la lógica pura de los programas, para empezar a pensar en objetos, lo que constituye la base de este paradigma. Esto nos ayuda muchísimo en sistemas grandes, ya que, en vez de pensar en funciones, pensamos en las relaciones o interacciones de los diferentes componentes del sistema.

Un programador diseña un programa de software organizando piezas de información y comportamientos relacionados en una plantilla llamada clase. Luego, se crean objetos individuales a partir de la plantilla de clase. Todo el programa de software se ejecuta haciendo que varios objetos interactúen entre sí para crear un programa más grande.

**Claves de la Programación Orientada a Objetos**

La POO se inspira en la forma en que percibimos y entendemos el mundo que nos rodea. Imagina que estás construyendo un sistema de gestión de una biblioteca. En lugar de pensar en términos de algoritmos y estructuras de datos, la POO te invita a considerar las entidades que existen en el contexto de la biblioteca, como libros, bibliotecarios y usuarios.

En este enfoque, cada una de estas entidades se convierte en un**objeto**, con **propiedades (datos)**y **comportamientos (funcionalidades)**. Por ejemplo, un objeto «Libro» puede tener atributos como el título, el autor y el año de publicación, así como métodos para obtener información sobre el libro, prestarlo o devolverlo a la biblioteca.

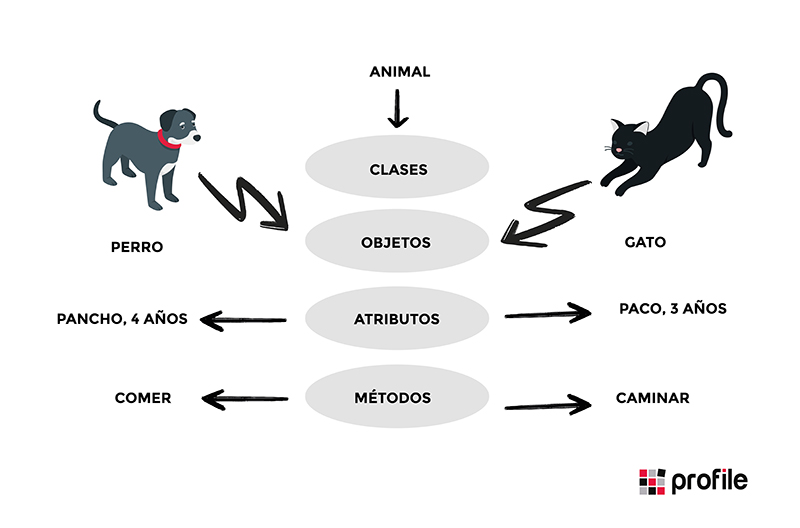
La clave de la POO radica en la interacción entre estos objetos. Pueden comunicarse entre sí enviándose mensajes y colaborando para lograr un objetivo común. Por ejemplo, un objeto «Usuario» podría enviar un mensaje al objeto «Libro» para solicitar su préstamo, y este último respondería actualizando su estado interno.

**Clases, objetos e instancias**

El primer y más importante concepto de la POO es la **distinción entre clase y objeto**.

Una **clase**es una plantilla. Define de manera genérica cómo van a ser los objetos de un determinado tipo. Por ejemplo, una clase para representar a animales puede llamarse ‘animal’ y tener una serie de **atributos**, como ‘nombre’ o ‘edad’ (que normalmente son propiedades), y una serie con los comportamientos que estos pueden tener, como caminar o comer, y que a su vez se implementan como métodos de la clase (funciones).

Un **ejemplo sencillo de un objeto,**como decíamos antes, podría ser un animal. Un animal tiene una edad, por lo que creamos un nuevo atributo de ‘edad’ y, además, puede envejecer, por lo que definimos un nuevo método. Datos y lógica. Esto es lo que se define en muchos programas como la definición de una clase, que es la definición global y genérica de muchos objetos.



**Con la clase se pueden crear instancias de un objeto**, cada uno de ellos con sus atributos definidos de forma independiente. Con esto podríamos crear un gato llamado *Paco*, con 3 años, y otro animal, este tipo perro y llamado *Pancho*, con una de edad de 4 años. Los dos están **definidos por la clase animal**, pero son dos instancias distintas. Por lo tanto, llamar a sus métodos puede tener resultados diferentes. Los dos comparten la lógica, pero cada uno tiene su estado de forma independiente.

**Pilares de la Programación Orientada a Objetos**

* **La encapsulación**

La encapsulación contiene**toda la información importante de un objeto dentro del mismo** y solo expone la información seleccionada al mundo exterior.   
Esta propiedad permite asegurar que la información de un objeto esté oculta para el mundo exterior, agrupando en una Clase las características o atributos que cuentan con un acceso privado, y los comportamientos o métodos que presentan un acceso público.

La encapsulación de cada objeto es responsable de su propia información y de su propio estado. La única forma en la que este se puede modificar es mediante los propios métodos del objeto. Por lo tanto, los atributos internos de un objeto deberían ser **inaccesibles desde fuera**, pudiéndose modificar sólo llamando a las funciones correspondientes. Con esto conseguimos mantener el estado a salvo de usos indebidos o que puedan resultar inesperados.

* **La abstracción**

La abstracción es cuando **el usuario interactúa solo con los atributos y métodos seleccionados de un objeto**, utilizando herramientas simplificadas de alto nivel para acceder a un objeto complejo.

En la programación orientada a objetos, los programas suelen ser muy grandes y los objetos se comunican mucho entre sí. El concepto de abstracción **facilita el mantenimiento de un código de gran tamaño**, donde a lo largo del tiempo pueden surgir diferentes cambios. Así, la abstracción se basa en usar **cosas simples para representar la complejidad**.

* **La herencia**

La herencia define **relaciones jerárquicas entre clases**, de forma que atributos y métodos comunes puedan ser reutilizados. Las clases principales extienden atributos y comportamientos a las clases secundarias. A través de la definición en una clase de los atributos y comportamientos básicos, se pueden crear clases secundarias, ampliando así la funcionalidad de la clase principal y agregando atributos y comportamientos adicionales.

Volviendo al ejemplo de los animales, se puede usar una sola clase de animal y agregar un atributo de tipo de animal que especifique el tipo de animal. Los diferentes tipos de animales necesitarán diferentes métodos, por ejemplo, las aves deben poder poner huevos y los peces, nadan. Incluso cuando los animales tienen un método en común, como moverse, la implementación necesitaría muchas declaraciones «si» para garantizar el comportamiento de movimiento correcto. Por ejemplo, las ranas saltan, mientras que las serpientes se deslizan. El principio de herencia nos permite solucionar este problema.

* **El polimorfismo**

El polimorfismo consiste en**diseñar objetos para compartir comportamientos**, lo que nos permite procesar objetos de diferentes maneras. Es la capacidad de presentar la misma interfaz para diferentes formas subyacentes o tipos de datos. Al utilizar la herencia, los objetos pueden anular los comportamientos principales compartidos, con comportamientos secundarios específicos. El polimorfismo permite que el mismo método ejecute diferentes comportamientos de dos formas: anulación de método y sobrecarga de método.

**Beneficios de Programación Orientada a Objetos**

* **Reutilización** del código.
* Convierte cosas complejas en **estructuras simples reproducibles**.
* Evita la **duplicación de código**.
* Permite **trabajar en equipo** gracias al encapsulamiento ya que minimiza la posibilidad de duplicar funciones cuando varias personas trabajan sobre un mismo objeto al mismo tiempo.
* Al estar la clase bien estructurada permite la **corrección de errores** en varios lugares del código.
* **Protege la información** a través de la encapsulación, ya que solo se puede acceder a los datos del objeto a través de propiedades y métodos privados.
* La abstracción nos permite **construir sistemas más complejos** y de una forma más sencilla y organizada.