**Programación Orientada a objetos**

La Programación Orientada a Objetos (POO) es un paradigma de programación, es decir, un modelo o un estilo de programación que nos da unas guías sobre cómo trabajar con él. Se basa en el concepto de clases y objetos. Este tipo de programación se utiliza para estructurar un programa de software en piezas simples y reutilizables de planos de código (clases) para crear instancias individuales de objetos.

A lo largo de la historia, han ido apareciendo diferentes paradigmas de programación. Lenguajes secuenciales como COBOL o procedimentales como Basic o C, se centraban más en la lógica que en los datos. Otros más modernos como Java, C# y Python, utilizan paradigmas para definir los programas, siendo la Programación Orientada a Objetos la más popular.

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.

La Programación Orientada a objetos permite que el código sea reutilizable, organizado y fácil de mantener. Sigue el principio de desarrollo de software utilizado por muchos programadores DRY (Don’t Repeat Yourself), para evitar duplicar el código y crear de esta manera programas eficientes. Además, evita el acceso no deseado a los datos o la exposición de código propietario mediante la encapsulación y la abstracción.

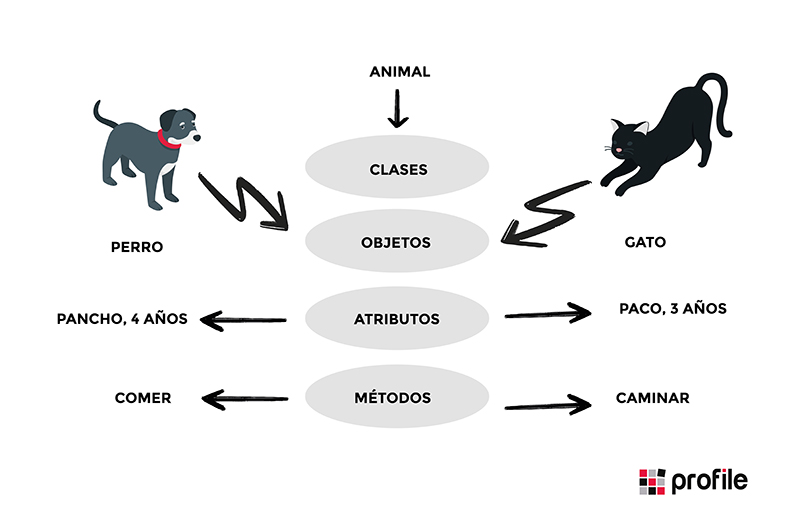
**Clases, objetos e instancias**

Las clases forman el modelo a partir del que se estructuran los datos y los comportamientos. 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 de edad, 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.



Principios 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.

Usamos de ejemplo un coche para explicar la encapsulación. El coche comparte información pública a través de las luces de freno o intermitentes para indicar los giros (interfaz pública). Por el contrario, tenemos la interfaz interna, que sería el mecanismo propulsor del coche, que está oculto bajo el capó. Cuando se conduce un automóvil es necesario indicar a otros conductores tus movimientos, pero no exponer datos privados sobre el tipo de carburante o la temperatura del motor, ya que son muchos datos, lo que confundiría al resto de conductores.

**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. Los objetos y las clases representan código subyacente, ocultando los detalles complejos al usuario. Por consiguiente, supone una extensión de la encapsulación. Siguiendo con el ejemplo del coche, no es necesario que conozcas todos los detalles de cómo funciona el motor para poder conducirlo.

**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.