Curso de OOP

Programación orientada a objetos

Es un paradigma de la programación que busca identificar todos objetos relacionados en la resolución de un problema para trabajar en torno a ellos.

Los componentes más importantes de este paradigma son:

- **Los objetos**
- **Las clases**
- **Las propiedades de las clases.**
- **Los métodos de las clases.**

Basados en cuatro pilares:

- **Abstracción**
- **Herencia**
- **Polimorfismo**
- **Encapsulamiento**

Objetos

En términos generales podemos definir que un objeto es aquel elemento físico o conceptual que posee **propiedades** y **comportamientos.** Además, se identifica siempre con un **sustantivo** desde una perspectiva gramatical.

En el ámbito físico, por ejemplo, podemos decir que un perro es un objeto. (Pero ojo, **no la idea de perro en general, sino un perro en particular**. Por ejemplo, un perro llamado Charlie sería el objeto). El caso de Charlie cumple al pie de la letra la definición: Se refiere a el con el uso de un sustantivo, tiene propiedades (su color es negro, y su tamaño grande...) y comportamientos (camina, ladra…).

En el ámbito conceptual podríamos hablar de, por ejemplo, una cuenta bancaria. (De nuevo, no el concepto de cuenta bancaria, sino la cuenta bancaria de alguien en particular: La cuenta bancaria de María). También cumple con la definición descrita anteriormente: Es un sustantivo, tiene propiedades (sin comisiones, límites de transferencia…) y comportamientos (retirar balance, añadir balance…).

Clases

Cuando te puse ejemplos sobre lo que era un objeto, recalqué varias veces que se trataban de elementos particulares y no de ideas generales (Charlie, en vez de perro; La cuenta de María, en vez de cuenta bancaria). Pues bien, cuando nos referimos a clases, la idea es precisamente inversa. Es decir, ya no **nos referimos a elementos particulares, sino a los conceptos generales**.

Por ejemplo, hablaríamos de la clase “perro”, o de la clase “cuenta bancaria” y estas agruparán a Charlie y a la cuenta de María respectivamente.

En términos de programación, las clases son una potente manera de entender todos los elementos que intervienen en el flujo de una aplicación (objetos) para poder tratar con ellos de una manera más dinámica.

Al igual que sucedía con los objetos, puedes entender a una clase como un **conjunto** de elementos **físicos o conceptuales que poseen propiedades y comportamientos**. Por ejemplo, la clase perro puede tener las propiedades (**propiedades de las clases**): color, tamaño… y comportamientos (**métodos)**: ladrar, caminar, saltar. De esta manera, a través del uso de la clase perro, podré crear a Charlie, a Mike y a todos los perros que quiera sin tener que recrear independientemente las propiedades y comportamientos de cada uno.

Abstracción

Habiendo entendido lo anterior, al proceso de encontrar una serie de objetos y definirle una clase generadora le denominamos **abstracción.** Por ejemplo, si tenemos una aplicación que hace uso de apartamentos/casas y hacemos la creación de una clase vivienda para gestionar el funcionamiento de ambos, estaríamos realizando **abstracción**.

Herencia

Nos referimos a herencia cuando creamos una clase que obtiene acceso a todas las propiedades y métodos de otra. En el ejemplo anterior, podríamos crear una clase “apartamento” que obtenga herencia de la clase vivienda. De esta manera, no haría falta recrear todas las propiedades y métodos comunes a apartamentos y casas, sino únicamente las particulares a apartamentos.

Polimorfismo

En ese sentido, el polimorfismo es una propiedad de la herencia. Imaginémonos que tenemos la clase “felino”, que dentro de sí tiene el método “aullar”. Pues bien, decidimos crear una clase “león” que toma herencia de la clase “felino”. Es decir, la clase “león” ahora también cuenta con el método “aullar” la cuestión es que el método “aullar” de la clase felinos está pensado para gatos únicamente y el aullido del león es distinto.

Pues bien, la idea del polimorfismo (poli: múltiples, morphe: formas) es que podemos tomar cualquier propiedad o método y transformarlo como más nos convenga. De esta manera, podríamos transformar el método aullido de la clase “felino” para que se corresponda al del león.

Encapsulamiento

El encapsulamiento se refiere a la protección de los datos de una clase, de manera que la información sea inviolable. Esto se consigue con el uso de un modificador de acceso, el cual modifica el alcance y visibilidad del miembro en cuestión.

Los modificadores de acceso son los siguientes:

- Public: Permite un acceso de carácter global.
- Protected: Permite el acceso a clases, paquetes y subclases
- Default: Permite el acceso a clases y paquetes.
- Private: permite únicamente el acceso desde la propia clase.

Diagramas de modelado

Cuando nos referimos a la forma en la que se plasman las soluciones informáticas con una orientación a objetos, nos referimos a diagramas de modelado. En la actualidad contamos con **OMT** (en desuso) y **UML** (basado en OMT).

- **OMT (Técnicas de modelado).**
- **UML (Lenguaje de modelado unificado).**

UML

UML referencia un lenguaje estándar en el que se respetan las soluciones basadas en orientación a objetos.

Está constituida por clases, las cuales se representan de la siguiente manera:

A los atributos y propiedades se les puede establecer una visibilidad que responderán a diferentes símbolos. Estos son:

- Private
+ Public
# Protected
~ Default

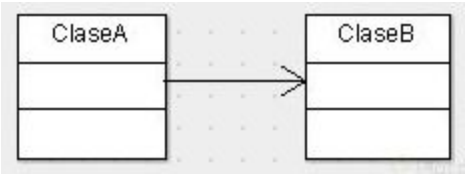
Podemos representar las relaciones entre los diferentes elementos haciendo uso de:

Asociación.

La asociación se representa con esta flecha:

Y representa dependencia.

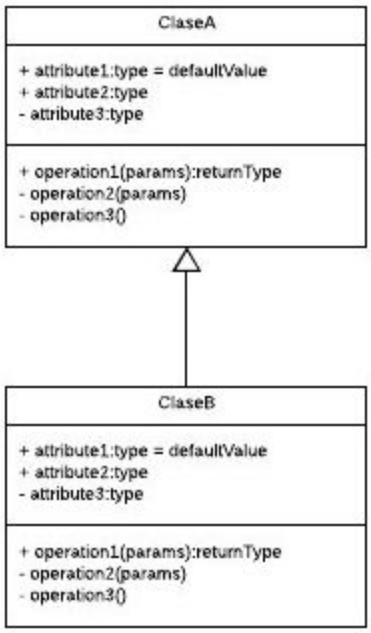
Por ejemplo, en este caso la ClaseA es por definición, dependiente de la clase B.



Herencia

Podemos indicar que una clase hereda atributos de otra con el uso de esta flecha en vertical:

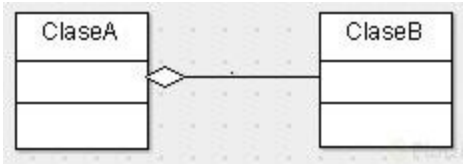
Por ejemplo, en este caso, la ClaseB hereda de la ClaseA.



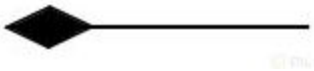
Agregación:

Representado con esta flecha:

Indica que una Clase dependerá de varios elementos de otra. Por ejemplo:



Composición:



Hace referencia a una relación compenetrada entre ambas clases. Es decir, una clase no podría existir sin la otra.