IBM SkillsBuild

IBM

Conceptos básicos de programación

Qué es la Programación Orientada a Objetos (POO)

Índice

Introducción

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.

¿Por qué POO?

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, de la que hablaremos en detalle más adelante.

Clases, objetos e instancias

Pensar en términos de objetos es muy parecido a cómo lo haríamos en la vida real. Por ejemplo, vamos a pensar en un coche para tratar de modelizarlo en un esquema de POO.

Diríamos que el coche es el elemento principal que tiene una serie de características, como podrían ser el color, el modelo o la marca. Además, tiene una serie de funcionalidades asociadas, como pueden ser ponerse en marcha, parar o aparcar.

Por tanto, pensar en objetos requiere analizar qué elementos vas a manejar en tus programas, tratando de identificar sus características y funcionalidades. Una vez tengamos un ecosistema de objetos, éstos colaborarán entre sí para resolver los objetivos de las aplicaciones.

Quizás al principio puede ser un poco complejo dar ese salto, para pensar en objetos, pero con el tiempo será una tarea que realizarás automáticamente.

Manejando el concepto de clase

En un esquema de programación orientada a objetos "el coche" sería lo que se conoce como "Clase".
La clase contiene la definición de las características de un modelo (el coche), como el color o la marca, junto con la implantación de sus funcionalidades, como arrancar o parar.

Las características definidas en la clase las llamamos propiedades y las funcionalidades asociadas las llamamos métodos.

Para entender este concepto tan importante dentro de la Programación Orientada a Objetos, podemos pensar que la clase es como un libro, que describe como son todos los objetos de un mismo tipo. La clase coche describe cómo son todos los coches del mundo, es decir, qué propiedades tienen y qué funcionalidades deben poder realizar y, por supuesto, cómo se realizan.

Manejando el concepto de objetos

A partir de una clase podemos crear cualquier número de objetos de esa clase. Por ejemplo, a partir de la clase "el coche" podemos crear un coche rojo que es de la marca Ford y modelo Fiesta, otro verde que es de la marca Seat y modelo Ibiza.

Por tanto, los objetos son ejemplares de una clase, o elementos concretos creados a partir de una clase. Puedes entender a la clase como el molde y a los objetos como concreciones creadas a partir del molde de clase.

Otros ejemplos concretos de clases y objetos

Por poner otro ejemplo vamos a ver cómo modelizaríamos en un esquema POO una fracción, es decir, esa estructura matemática que tiene un numerador y un denominador que divide al numerador, por ejemplo 3/2.

La fracción será la clase y tendrá dos propiedades, el numerador y el denominador. Luego podría tener varios métodos como simplificarse, sumarse con otra fracción o número, restarse con otra fracción, etc.

A partir de la definición de una fracción (la clase) podremos construir un número indeterminado de objetos de tipo fracción. Por ejemplo podemos tener el objeto fracción 2/5 o 3/9, 4/3, etc. Todos esos son objetos de la clase fracción de números enteros.

Nuestra clase fracción la podríamos utilizar en cualquier número de programas, por ejemplo en un programa de matemáticas hará uso de la clase fracción y construirá muchos objetos de tipo fracción para hacer cuentas diversas. Pero podrías usar esa misma clase fracción en un programa de contabilidad o facturación.

Otro ejemplo podría ser la clase coordenada, que tiene dos propiedades, el valor x e y. Podrías tener otra clase "coordenada 3D" que necesitaría 3 propiedades x, y y z. Las coordenadas se podrían sumar a otra coordenada, mostrarse en un gráfico, encontrar el camino más corto entre dos coordenadas, etc.

Estos son simplemente ejemplos simples de clases, que también nos sirven para destacar la reutilización que podemos conseguir con los objetos. La clase coordenada la podremos usar en infinidad de programas de gráficos, mapas, etc. Por su parte, la clase coche la podrás utilizar en un programa de gestión de un taller de coches o en un programa de gestión de un parking. A partir de clase coche y crearán diversos objetos de tipo coche para hacer las operativas tanto en la aplicación del taller como en la del parking.

Los objetos colaboran entre sí

En los lenguajes puramente orientados a objetos, tendremos únicamente clases y objetos. Las clases permitirán definir un número indeterminado de objetos, que colaboran entre ellos para resolver los problemas.

Con muchos objetos de diferentes clases conseguiremos realizar las acciones que se desean implementar en la funcionalidad de la aplicación.

Además, las propias aplicaciones como un todo, también serán definidas por medio de clases. Es decir, el taller de coches será una clase, de la que podremos crear el objeto taller de coches, que utilizará objetos coche, objetos de clase herramienta, objetos de clase mecánico, objetos de clase recambio, etc.

Para continuar vamos a definir de manera más formal los conceptos que acabamos de introducir anteriormente.

Clases en Programación Orientada a Objetos

Como habrás podido entender, las clases son declaraciones de objetos, también se podrían definir como abstracciones de objetos. Esto quiere decir que la definición de un objeto es la clase. Cuando programamos un objeto y definimos sus características y funcionalidades en realidad lo que estamos haciendo es programar una clase. En los ejemplos anteriores en realidad hablábamos de las clases coche o fracción porque sólo estuvimos definiendo, aunque por encima, sus formas.

Propiedades en clases

Las propiedades o atributos son las características de los objetos o ejemplares de una clase. Cuando definimos una propiedad normalmente especificamos su nombre y su tipo.

Aunque no es una manera muy correcta de hablar, nos podemos hacer a la idea de que las propiedades son algo así como variables donde almacenaremos los datos relacionados con los objetos.

Métodos en las clases

Son las funcionalidades asociadas a los objetos, que son implantadas o programadas dentro de las clases. Es decir, cuando estamos programando las clases, las funciones que creamos dentro asociadas a esas clases las llamamos métodos.

Aunque los métodos son como funciones, es importante que los llames métodos para que quede claro que son funciones que existen dentro del contexto de una clase, funciones que podremos invocar sobre todos los objetos creados a partir de una clase.

Objetos en Programación Orientada a Objetos

Los objetos son ejemplares de una clase. A partir de una clase puedo crear ejemplares (objetos) de esa clase, que tendrán por tanto las características y funcionalidades definidas en esa clase.

Cuando creamos un ejemplar tenemos que especificar la clase a partir de la cual se creará. Esta acción de crear un objeto a partir de una clase se llama instanciar (que viene de una mala traducción de la palabra instace que en inglés significa ejemplar). Por ejemplo, un objeto de la clase fracción es por ejemplo 3/5. El concepto o definición de fracción sería la clase, pero cuando ya estamos hablando de una fracción en concreto 4/7, 8/1000, o cualquier otra, la llamamos objeto.

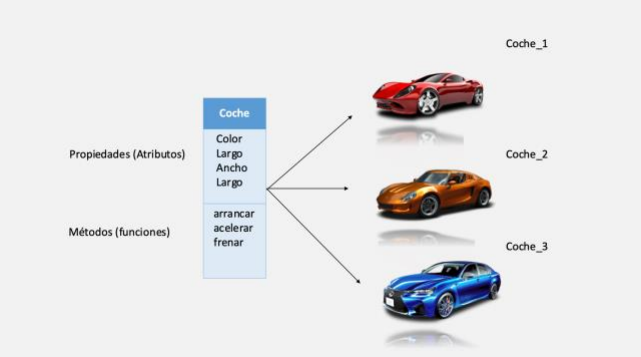
Para crear un objeto se tiene que escribir una instrucción especial que puede ser distinta dependiendo el lenguaje de programación que se emplee, pero será algo parecido a esto.

miCoche = new Coche()

Con la palabra "new" especificamos que se tiene que crear una instancia de la clase que sigue a continuación.

En este caso "Coche" sería el nombre de la clase. Con la palabra "new" se crea una nueva instancia de coche, un objeto concreto de la clase coche y ese objeto se almacena en la variable "miCoche".

Dentro de los paréntesis podríamos colocar parámetros con los que inicializar el objeto de la clase coche, como podría ser su color, o su marca.



Estados en objetos

Cuando tenemos un objeto sus propiedades toman valores. Por ejemplo, cuando tenemos un coche la propiedad color tomará un valor en concreto, como por ejemplo rojo o gris metalizado. El valor concreto de una propiedad de un objeto se llama estado.

Para acceder a un estado de un objeto para ver su valor o cambiarlo se utiliza el operador punto.

miCoche.color = "rojo"

El objeto es miCoche, luego colocamos el operador punto y por último el nombre de la propiedad a la que deseamos acceder. En este ejemplo estamos cambiando el estado de la propiedad "color" del objeto al valor "rojo". Esto lo hacemos con una simple asignación.

Mensajes en objetos

Un mensaje en un objeto es la acción de efectuar una llamada a un método. Por ejemplo, cuando le decimos a un objeto coche que se ponga en marcha estamos pasándole el mensaje "ponte en marcha".

Para mandar mensajes a los objetos utilizamos el operador punto, seguido del método que deseamos invocar y los paréntesis, como en las llamadas a las funciones.

miCoche.ponteEnMarcha()

En este ejemplo pasamos el mensaje "ponteEnMarcha" al objeto "miCoche".

Después del nombre del método que queremos invocar hay que colocar paréntesis, igual que cuando invocamos una función. Dentro de los paréntesis irían los parámetros, si es que el método los requiere.

4 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.

Alrededor de estos principios de la programación orientada a objetos se construyen muchas cosas. Por ejemplo, los principios SOLID, o los patrones de diseño, que son recetas que se aplican a problemas recurrentes que se han encontrado y se repiten en varios proyectos.

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.