



CICLO 2

[FORMACIÓN POR CICLOS]


Introducción a la Programación **ORIENTADA A OBJETOS**



Ingeni@
Soluciones TIC



**UNIVERSIDAD
DE ANTIOQUIA**
Facultad de Ingeniería



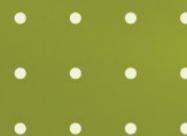
La programación no solo consiste en escribir algoritmos. Si bien ese conjunto de pasos que escribimos utilizando un lenguaje para resolver un problema es necesario, el desarrollo de aplicaciones de *software* no se limita solo a eso. Al momento de escribir un programa, a menudo es necesario modelar una parte del mundo que nos rodea (por modelar entendemos el representar de alguna manera esa parte del mundo, teniendo en cuenta solo los elementos que nos interesan en ese momento, y dejando por fuera todo lo demás).


Un ejemplo de lo anterior es el plano de un edificio. Un plano es una representación gráfica de un edificio o máquina, que tiene en cuenta ciertos elementos relevantes para el proceso constructivo y deja por fuera muchos otros. Por esto, un plano cuenta con medidas o dimensiones (a escala), ubicación de puertas y ventanas, grosor de los muros, ubicación de tomacorrientes y suiches, pero no incluye muchos detalles innecesarios, como el color de las ventanas, el diseño del piso o el material de las puertas.

Si nos fijamos bien, la representación de un edificio que incluya absolutamente todos los detalles puede resultar increíblemente compleja. Se recurre entonces a los planos para lidiar con dicha complejidad. En general, en este y en otros casos se lleva a cabo un proceso de abstracción, el cual permite simplificar un objeto del mundo real (el edificio) usando un modelo (el plano) que sea útil en un momento particular.

A esto debemos sumar que el desarrollo de *software* es una actividad compleja de por sí. Las razones son varias:

- En ocasiones, el dominio del problema es complejo. Por ejemplo, desarrollar un sistema de información para la gestión financiera de una empresa exige tener en cuenta una enorme cantidad de conceptos, reglas de negocio y restricciones.
- La gestión de proyectos grandes de desarrollo suele ser difícil y compleja.
- Al tratarse de algo que no podemos tocar, el *software* aparenta (para el cliente) ser algo infinitamente flexible y con un escaso costo a la hora de crearlo, modificarlo o destruirlo.





Para lidiar con las complejidades anteriores, tanto del mundo real como del mundo del desarrollo de *software*, se usa la programación orientada a objetos.

La **programación orientada a objetos** es un paradigma de programación en el cual los programas se organizan como colecciones de objetos que colaboran entre sí. Normalmente, los objetos son instancias de una clase preexistente, y tanto entre objetos como entre clases se dan relaciones de diversa índole. La programación orientada a objetos se suele apoyar en el **diseño orientado a objetos**, en el cual el dominio de la aplicación o programa en construcción se descompone en objetos usando una notación gráfica destinada a dicho fin.


Conceptos básicos de la programación orientada a objetos

La programación orientada a objetos cuenta con varios conceptos que son comunes en todos los lenguajes de programación que soportan dicho paradigma, y en todos ellos significan más o menos lo mismo; estos conceptos son: clase, objeto, atributo, método y relación (habiendo relaciones de diferentes tipos).

En programación orientada a objetos, una **clase** corresponde a la descripción abstracta de un conjunto de objetos con propiedades similares, con un comportamiento común y relaciones similares con otras clases. Una clase también se puede entender como una especie de molde, modelo o plantilla a partir de la cual se crean objetos individuales.

Un ejemplo de lo que podríamos ver como una clase en el mundo real es el concepto de *Jugador de Fútbol*. Un jugador de fútbol tiene propiedades comunes a todo jugador (nombre, edad, estatura, peso, velocidad, habilidad para gambetear, potencia, etc.), aun cuando entre todos los jugadores de fútbol los “valores” de dichas propiedades sean diferentes. Esto es, todo jugador tiene una edad, pero la edad puede ser diferente entre jugador y jugador.





Otro ejemplo de lo que es una clase es el concepto de Bicicleta. Todas las bicicletas tienen unas propiedades comunes (marca, peso, material del marco, número de platos, número de piñones, etc.), si bien, para cada bicicleta individual, los “valores” de dichas propiedades pueden ser diferentes entre bicicleta y bicicleta. Por ejemplo, el peso de mi bicicleta puede ser diferente al peso de la suya. Además, todas las bicicletas tienen un comportamiento común: si pedaleamos, aumenta su velocidad; si frenamos, esta disminuye; y si accionamos una palanca de cambios, un cambio sube o baja. Este comportamiento determina cómo interactuamos con los objetos de dicha clase y cómo estos responden a dichas interacciones.

Bicicleta

- **Atributos**

- marca
- referencia
- color
- velocidad

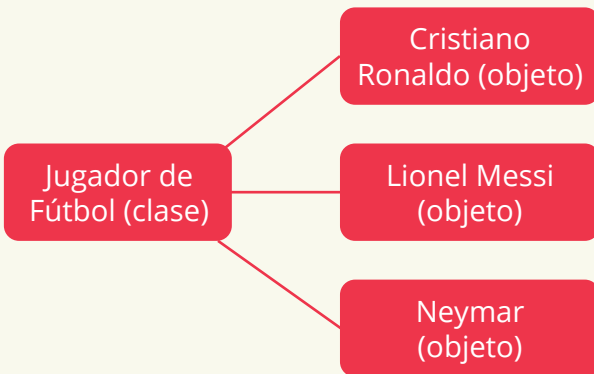
- **Métodos**

- pedalear
- frenar

Luego tenemos los objetos. Un **objeto** es, en su definición más rigurosa, una instancia de una clase, y cuenta con un comportamiento y un estado. El “comportamiento” viene dictado por los métodos de la clase, y su “estado”, por los valores que toman los atributos definidos en la clase. También podemos ver los objetos como aquello en lo que se materializan las clases, pudiendo una clase dar lugar a uno, dos o muchos objetos diferentes.

Por ejemplo, para la clase *Jugador de Fútbol* que describimos arriba, los objetos serían los jugadores de fútbol que conocemos: Cristiano Ronaldo, Lionel Messi, Neymar, y muchísimos otros más. Todos los jugadores de fútbol pueden hacer más o menos lo mismo (si bien algunos harán ciertas cosas mejor que otros), pero por lo general diferirán en sus propiedades. En el caso de los objetos de esta clase, se diferencian normalmente en el nombre, la habilidad y el peso, entre otras propiedades.





En el caso de la clase *Bicicleta*, los objetos son bicicletas individuales: cada una de las que vemos en la calle o de las que tenemos en nuestros patios son objetos. Para cada bicicleta (u objeto de la clase *Bicicleta*), las propiedades suelen cambiar: el peso, la referencia, el marco, el color, entre otras cosas, pueden ser diferentes. Pero para cada caso, su comportamiento es similar: podemos pedalear en ellas y frenarlas, entre otras operaciones.

Por otro lado, las clases y, por tanto, los objetos cuentan con atributos. Un **atributo** no es más que una propiedad que comparten todos los objetos de una clase. Por lo general los atributos de una clase tienen un nombre y cuentan con un rango de valores que pueden tomar. Finalmente, varios objetos diferentes de una misma clase podrán tener valores diferentes en un mismo atributo.

Por ejemplo, en el caso de la clase *Jugador de Fútbol* vista anteriormente, los atributos serían el nombre del jugador, su altura, peso, habilidad, potencia, entre muchas otras que consideremos necesarias en su modelado. Todos los jugadores de fútbol (objetos) tendrán todos esos atributos, pero para muchos de ellos el valor de cada atributo podrá ser diferente entre sí.



Jugador (clase)	Messi (objeto)	Mbappé (objeto)
<ul style="list-style-type: none"> • Atributos • nombre • peso • velocidad • Métodos • patear • cabecear 	<ul style="list-style-type: none"> • Atributos • nombre: "Lionel" • peso: 72 • velocidad: 40 • Métodos • patear • cabecear 	<ul style="list-style-type: none"> • Atributos • nombre: "Kylian" • peso: 73 • velocidad : 30 • Métodos • patear • cabecear

Lo mismo sucede en el caso de la clase *Bicicleta*, en el que los atributos serían la referencia, el peso, el material del marco y el color, entre otros. Todas las bicicletas tienen los mismos atributos, pero sabemos que en cada bicicleta individual (la mía o la suya) el valor de los atributos puede variar, ya sea porque tienen un color diferente, están hechas con otro material, o cualquiera de las demás posibilidades.

Arriba hablábamos de que los objetos de una determinada clase tienen un comportamiento común. Dicho comportamiento lo comprende un conjunto de operaciones, que a la hora de programar se implementan en forma de métodos. Un **método** es por lo general una función (en un lenguaje de programación particular) asociada a una clase, mediante la cual podemos interactuar con objetos de dicha clase. También, los métodos de un objeto son el medio por el cual podemos cambiar el estado (valor de los atributos) del mismo.

Por ejemplo, en el caso de la clase *Bicicleta*, los métodos podrían ser pedalear, accionar el freno, virar a la derecha o a la izquierda. Son, normalmente, las formas en las que interactuamos con nuestra bicicleta, y, en muchos casos, cambian atributos de la bicicleta específica (el objeto). Un ejemplo es que, al pedalear o frenar (métodos), cambiamos la velocidad (atributo) de la bicicleta.



Por último, cabe mencionar que entre los objetos y clases se pueden dar diferentes tipos de relaciones. Uno de los tipos de relaciones más importantes que se dan entre clases es la relación de herencia. En programación orientada a objetos, la **herencia** consiste en que un conjunto de clases herede los atributos y métodos de una clase padre. Es decir, dichas clases van a contar con los mismos atributos y métodos de la clase padre, pero, además, cada una va a contar con sus propios métodos y/o atributos adicionales, constituyendo tipos nuevos.

Por ejemplo, se podría dar una relación de herencia entre las clases *Bicicleta* y *Bicicleta de Montaña*, *Bicicleta de Ruta* y *Bicicleta Eléctrica*. Si se fijan, las tres últimas cuentan con todos los métodos y atributos de una bicicleta normal (la primera), pero cada una de ellas agrega métodos y atributos nuevos. Por ejemplo, una *Bicicleta de Ruta* va a tener llantas diferentes a las de los demás tipos de bicicleta. También, una *Bicicleta de Montaña* va a tener uno o dos amortiguadores. Finalmente, la *Bicicleta Eléctrica* va a tener un motor, así como un método de encendido que no tendrán los otros tipos. La herencia permite entonces crear nuevas clases reutilizando los métodos y atributos de una clase a menudo denominada clase padre.

