

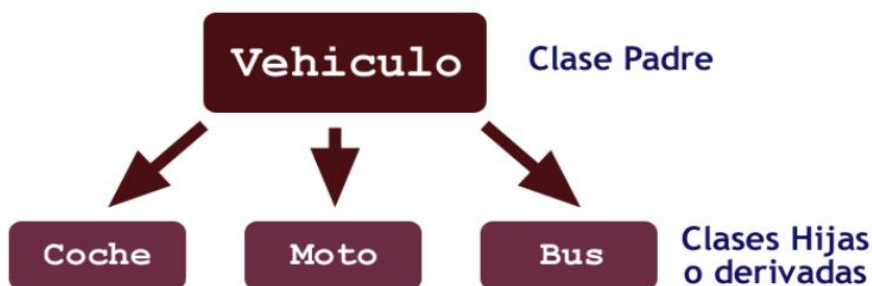
Qué es el polimorfismo en la Programación Orientada a Objetos, el motivo de su existencia y cómo implementar polimorfismo en clases y objetos.

Definición: El polimorfismo es una relajación del sistema de tipos, de tal manera que una referencia a una clase (atributo, parámetro o declaración local o elemento de un vector) acepta direcciones de objetos de dicha clase y de sus clases derivadas (hijas, nietas, ...).

Herencia y las clasificaciones en Programación Orientada a Objetos

Para poder entender este concepto de OOP necesitamos entender otras cosas previas, como es el caso de la herencia.

Veremos que el polimorfismo y la herencia son dos conceptos estrechamente ligados. Conseguimos implementar polimorfismo en jerarquías de clasificación que se dan a través de la herencia. Por ejemplo, tenemos una clase vehículo y de ella dependen varias clases hijas como coche, moto, autobús, etc.



Polimorfismo en objetos

Tal como funcionan los lenguajes fuertemente tipados, una variable podrá apuntar a un objeto de la clase que se indicó en el momento de su declaración. Un método cuyo parámetro se haya declarado de una clase, solo aceptará recibir objetos de esa clase. Un array que se ha declarado que es de elementos de una clase determinada, solo aceptará que rellenemos sus casillas con objetos de esa clase declarada.

```
Vehiculo[] misVehiculos = new Vehiculo[3];
```

Esa variable `misVehiculos` es un array y hemos declarado que el contenido de las casillas serán objetos de la clase "Vehiculo". Solo podrá contener objetos de la clase Vehiculo. Pues bien, **polimorfismo es el mecanismo por el cual podemos "relajar el sistema de tipos", de modo que nos acepte también objetos de las clases hijas o derivadas.**

Por tanto, la "relajación" del sistema de tipos no es total, sino que tiene que ver con las clasificaciones de herencia que tengamos en nuestros sistemas de clases:

- Si definimos un array con elementos de una determinada clase, el compilador también te aceptará que metas en esas casillas objetos de una clase hija de la que fue declarada.
- Si declaramos que una función recibe como parámetros objetos de una determinada clase, el compilador también aceptará que le enviemos objetos de una clase derivada de aquella que fue declarada cuando la invoquemos.

En concreto, en nuestro array de vehículos, gracias al polimorfismo podremos contener en los elementos del array no solo vehículos genéricos, sino también todos los objetos de clases hijas o derivadas de la clase "Vehiculo", es decir objetos de la clase "Coche", "Moto", "Bus" o cualquier hija que se haya definido.

Para qué nos sirve en la práctica el polimorfismo

Si tenemos la variable:

```
Largometraje miLargo = new Largometraje("Lo que el viento se llevó");
```

Esa variable "miLargo", por declaración puede tener una referencia a un objeto de la clase "Largometraje".

Supongamos una clase "Cine". En un cine se reproducen largometrajes. Podemos tener varios tipos de largometrajes, como películas o documentales, etc. Las películas y documentales tienen diferentes características, distintos horarios de audiencia, distintos precios para los espectadores y por ello decidimos que la clase "Largometraje" tenga clases hijas o derivadas como "Película" y "Documental".

Supongamos que en la clase "Cine" creamos un método que se llama "reproducir()". Este método podrá recibir como parámetro aquello que queremos proyectar en una sala de cine y podrán llegar a veces objetos de la clase "Película" y otras veces objetos de la clase "Documental". Debido a que los métodos declaran los tipos de los parámetros que reciben, habrá que hacer algo como esto:

```
reproducir(Pelicula peliculaParaReproducir)
```

Pero si luego tienes que reproducir documentales, tendrás que declarar:

```
reproducir(Documental documentaParaReproducir)
```

Probablemente, el código de ambos métodos sea exactamente el mismo. Poner la película en el proyector, darle al play, crear un registro con el número de entradas vendidas, parar la cinta cuando llega al final, etc. ¿Realmente es necesario hacer dos métodos? Si más adelante nos surge una tercera opción para reproducir ¿Tendremos que crear un nuevo método reproducir() sobre la clase "Cine" que nos acepte ese tipo de emisión? ¿es posible ahorrarnos todo ese mantenimiento?

Aquí es donde el polimorfismo nos ayuda. Podremos crear perfectamente un método "reproducir()" que recibe un largometraje y donde podremos recibir todo tipo de elementos, películas, documentales y cualquier otra cosa similar que sea creada en el futuro.

Entonces, lo que nos permiten hacer los lenguajes con orientación a objetos, es declarar el método "reproducir()" indicando que el parámetro que vamos a recibir es un objeto de la clase padre "Largometraje", pero donde realmente el se aceptará cualquier objeto de la clase hija o derivada, "Película", "Documental", etc.

```
reproducir(Largometraje elementoParaReproducir)
```

Podremos crear películas y reproducirlas, también crear documentales para luego reproducirlos y todos estos objetos son aceptados por el método "reproducir()", gracias a la relajación del sistema de tipos. Incluso, si queremos reproducir otro tipo de cinta, no habrá que tocar la clase "Cine" y el método "reproducir()". Siempre que aquello que queramos reproducir sea de la clase "Largometraje" o una clase hija, el método lo aceptará.

Otro ejemplo: volviendo de nuevo a la clase Vehículo.

Tenemos la clase Parking. Dentro de esta tenemos un método estacionar(). Puede que en un parking tengamos que estacionar coches, motos o autobuses. Sin polimorfismo tendríamos que crear un método que permitiese estacionar objetos de la clase "Coche", otro método que acepte objetos de la clase "Moto" para estacionarlos, etc. Pero estacionar un coche, una moto o un bus es bastante similar: "entrar en el parking, recoger el ticket de entrada, buscar una plaza, situar el vehículo dentro de esa plaza...".

Lo ideal sería que nuestro método permitiera recibir todo tipo de vehículos para estacionarlos, primero por reutilización del código, ya que es muy parecido estacionar uno u otro vehículo, pero además porque así, si mañana el mercado trae otro tipo de vehículos,

como un todoterreno híbrido, o una furgoneta, nuestro software sea capaz de aceptarlos sin tener que modificar la clase Parking.

En resumen, gracias al polimorfismo, cuando declaramos la función estacionar() podemos decir que recibe como parámetro un objeto de la clase "Vehículo" y el compilador me aceptará no solamente vehículos genéricos, sino todos aquellos objetos que hayamos creado que hereden de la clase Vehículo, es decir, coches, motos, buses, etc. Esa relajación del sistema de tipos para aceptar una gama de objetos diferente es lo que llamamos **polimorfismo**.

Declaro la función:

```
function estacionar( Vehiculo ) { }
```

Invoco la función: (soporto polimorfismo)

```
estacionar( Coche ) ;  
estacionar( Moto ) ;  
estacionar( Bus ) ;
```

No puedo invocar la función: (no lo permitiría, porque no ser clasificación de herencia de vehículos)

```
estacionar( Mono ) ;  
estacionar( INT ) ;
```

En el futuro si podría: (Si creo las clases "Van" o "Nave especial" y heredan de Vehículo)

```
estacionar( Van ) ;  
estacionar( Nave espacial ) ;
```