POO/OOP

Programación Orientado a Objetos

Object-Oriented Programming

Vocabulario

- Clase, objeto
- Ejemplar de clase, instancia de clase, ejemplarizar una clase, instanciar una clase
- Modularización
- Encapsulamiento / encapsulación
- Herencia
- Poliformismo
- Sobrecarga del método constructor
  - Métodos mágicos

métodos static

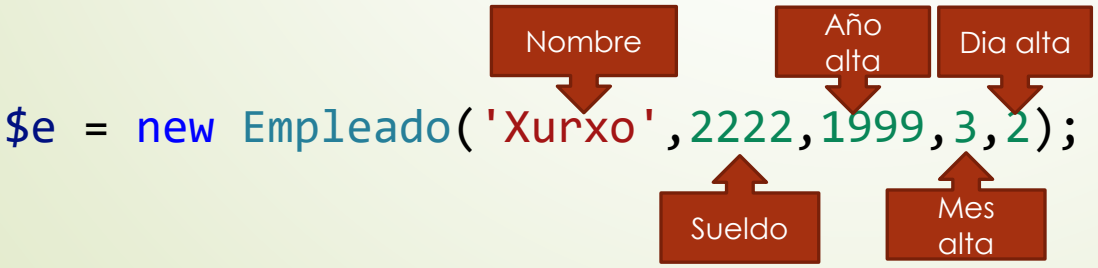
- Los métodos también pueden ser estáticos
- Un **método static** afecta a la clase no al objeto
- **No hace falta ser instanciada**
- Se accede a ellos a través del **nombre de la clase** o con la palabra reservada **self**
- Como el **método estático pertenece a la clase** y no al objeto para acceder a atributos/propiedades de la clase éstos han de ser estáticos también, es decir, no se puede acceder al campo de un objeto, pero sí al campo de una clase.

```
public static function funcionEstatica():void{

    echo  self::$ordenSiguiente;  //campo de CLASE

    echo $this->nombreEmpleado; // Error porque hace referencia al campo de un OBJETO

}
```

Sobrecarga del constructor

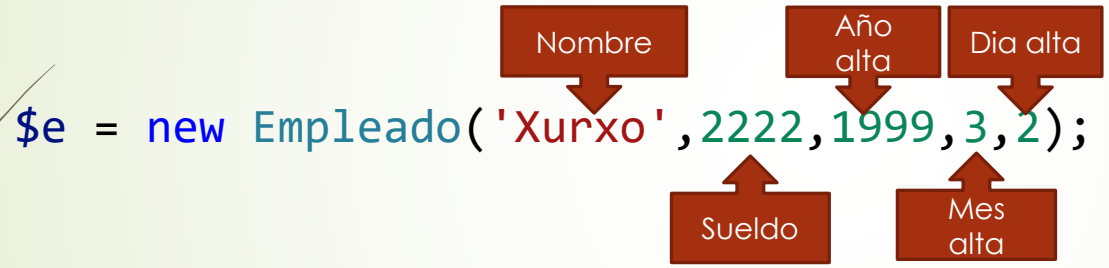
El **método constructor** es una función más de una clase, pero a diferencia de los demás métodos nos permite dar un **estado inicial** del objeto

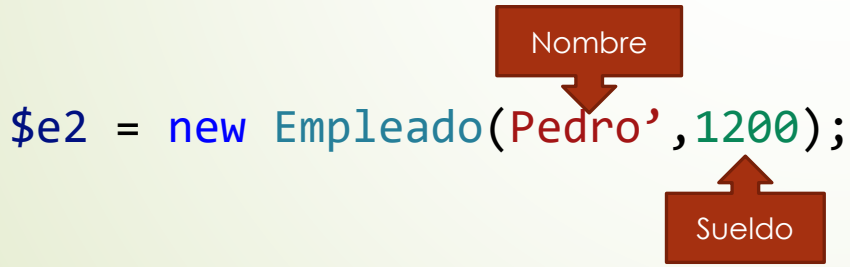
```php
/**
 * Método constructor
 *
 * @param string $paramNombre Nombre del empleado
 * @param float|null $paramSueldo Sueldo del Empleado. Si es nulo se establecerá el sueldo base
 * @param integer $ano Año de alta del empleado
 * @param integer $mes Mes de ata del empleado
 * @param integer $dia Día de alta del empleado
 */
 public function __construct(string $paramNombre,?float $paramSueldo, int $ano, int $mes, int $dia){
   $this->nombre = $paramNombre;
   $this->sueldo = ($paramSueldo)?$paramSueldo:self::SUELDO_BASE;
   ////DateTime::createFromFormat //método estático --> ahora da igual, ya se verá
   $this->altaContrato = DateTime::createFromFormat('Y-m-d',"$ano-$mes-$dia");
   //Establecemos orden
   $this->orden = Empleado::$ordenSiguiente++;
}
```



Sobrecarga del constructor

Imaginémonos un caso como el siguiente en que queramos instanciar dos objetos empleados pero con diferente número de parámetros **sin utilizar la sobrecarga de parámetros**.





En los lenguajes POO esto es perfectamente posible.

Véase la solución en POO de esta situación

Sobrecarga del constructor

Tendríamos tantos métodos constructores como
- Número de parámetros pasaramos al constructor
- El tipo de dato que pasáramos como parámetro

```php
public function __construct(string $nombre,float $sueldo,int $ano, int $mes, $dia){
    $this->nombre = $nombre;
    $this->sueldo = $sueldo;
    $this->altaContrato = DateTime::createFromFormat("d-m-Y","$dia-$mes-$ano");
}
```

```php
public function __construct(string $nombre,float $sueldo){
    $this->nombre = $nombre;
    $this->sueldo = $sueldo;
    $this->altaContrato = new DateTime('NOW');
}
```

Desgraciadamente PHP no nació como POO aunque con las nuevas versiones se va adaptando. Una de sus limitaciones es que no existe dicha sobrecarga.

Merda!!!Esto no lo puedo hacer en PHP

SOLUCIÓN
Métodos mágicos

Métodos mágicos

- func_num_args()
  Retorna el número de parámetros de una función
- func_get_args()
  Retorna en un array los argumentos
- call_user_func_array
  Permite llamar a un método y/o función pasándole los parámetros

Aunque con limitaciones los métodos mágicos nos permiten obtener el números de parámetros y los parámetros de un método cualquiera.
De esta manera podemos imitar el funcionamiento de un método constructor.
Para una explicación disponemos del enlace:

https://desarrolloweb.com/articulos/sobrecarga-constructores-php.html

Ejercicio

Nos piden realizar la gestión de una serie de productos.

Los productos tienen **seguros** los siguientes atributos:

1. Nombre del producto
2. Precio

Además disponemos de dos tipos de perecedero y no perecedero. Del perecedero hay que saber cuántos días faltan para que caduque.

Crea sus constructores, getters, setters y toString.

Tendremos una función llamada calcularPrecio() que que según tipo de producto (perecedero o no perecedero) puede devolver:

- En el perecedero el precio final se reducirá según los días a caducar:
  - Si le queda 1 día para caducar, se reducirá 4 veces su precio.
  - Si le quedan 2 días para caducar, se reducirá 3 veces su precio.
  - Si le quedan 3 días para caducar, se reducirá a la mitad de su precio.
- En el NO perecedero el precio final coincide con su precio.