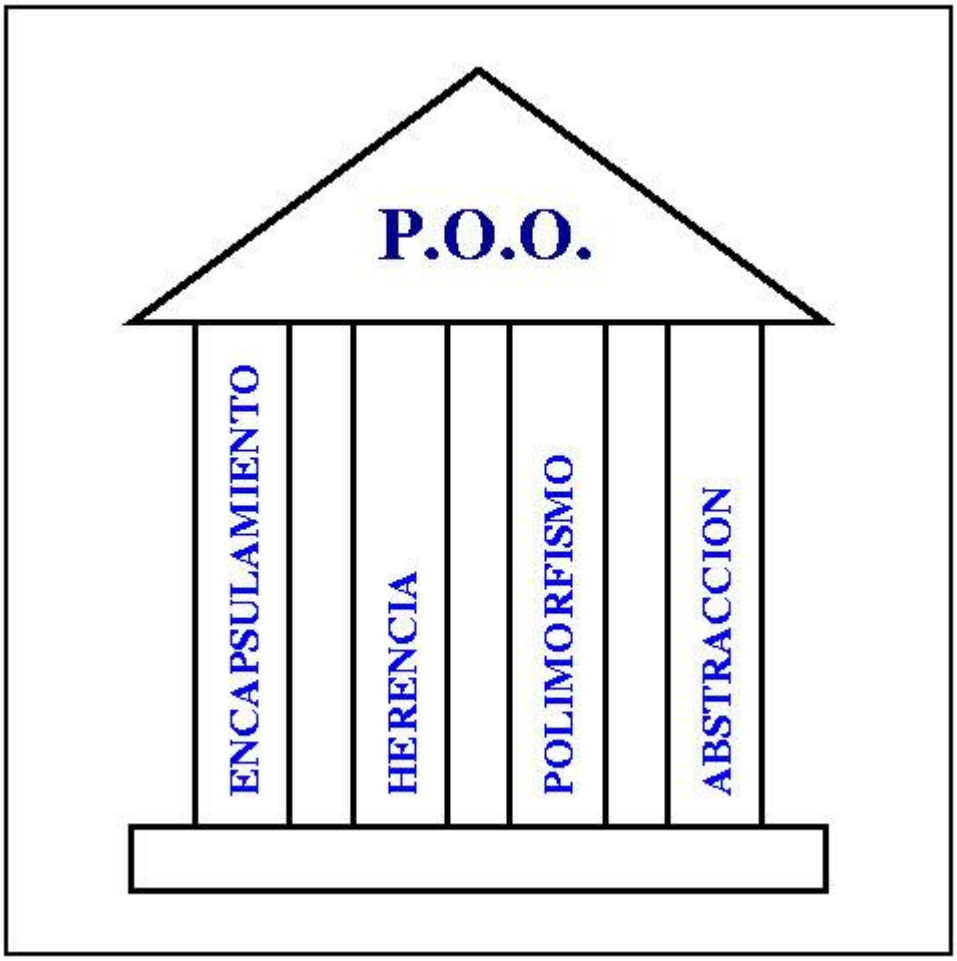
PROGRAMACIÓN ORIENTADA A OBJETOS EN PHP

La POO es un paradigma de programación (o técnica de programación) que utiliza objetos e interacciones en el diseño de un sistema.

Paradigma: teoría cuyo núcleo central [...] suministra la base y modelo para resolver problemas [...] (Definición de la Real Academia Española, vigésimo tercera edición)

Como tal, nos enseña un método -***probado y estudiado***- el cual se basa en las interacciones de objetos (todo lo descrito en el título anterior, Pensar en objetos) para resolver las necesidades de un sistema informático.

Básicamente, este paradigma se compone de 4 pilares y diferentes características que veremos a continuación.



CARACTERÍSTICAS CONCEPTUALES DE LA POO

La POO debe guardar ciertas características que la identifican y diferencian de otros paradigmas de programación. Dichas características se describen a continuación:

- **Abstracción**

Aislación de un elemento de su contexto. Define las características esenciales de un objeto.

- **Encapsulamiento**

Reúne al mismo nivel de abstracción, a todos los elementos que puedan considerarse pertenecientes a una misma entidad.

- **Modularidad**

Característica que permite dividir una aplicación en varias partes más pequeñas (denominadas módulos), independientes unas de otras.

- **Ocultación (aislamiento)**

Los objetos están aislados del exterior, protegiendo a sus propiedades para no ser modificadas por aquellos que no tengan derecho a acceder a las mismas.

- **Polimorfismo**

Es la capacidad que da a diferentes objetos, la posibilidad de contar con métodos, propiedades y atributos de igual nombre, sin que los de un objeto interfieran con el de otro.

- **Herencia**

Es la relación existente entre dos o más clases, donde una es la principal (padre) y otras son secundarias y dependen (heredan) de ellas (clases "hijas"), donde a la vez, los objetos heredan las características de los objetos de los cuales heredan.

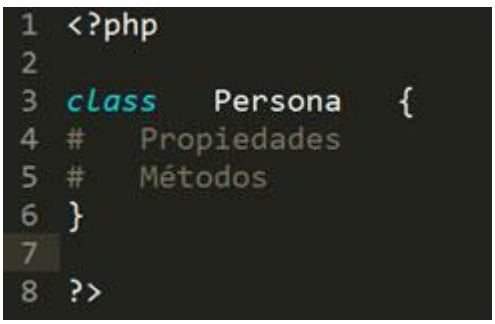
- **Recolección de basura**

Es la técnica que consiste en destruir aquellos objetos cuando ya no son necesarios, liberándolos de la memoria.

CLASES O CLASES CONCRETAS

Una clase es un modelo que se utiliza para crear objetos que comparten un mismo comportamiento, estado e identidad.

Persona es la metáfora de una clase (la abstracción de Juan, Pedro, Ana y María), cuyo comportamiento puede ser caminar, correr, estudiar, leer, etc. Puede estar en estado despierto, dormido, etc. Sus características (propiedades) pueden ser el color de ojos, color de pelo, su estado civil, etc.



Objeto

Es una entidad provista de métodos o mensajes a los cuales responde (comportamiento); atributos con valores concretos (estado); y propiedades (identidad).

```
$persona = new Persona();
/*
El objeto, ahora, es $persona, que se ha
creado siguiendo el modelo de la clase Persona
*/
```

Método

Es el algoritmo asociado a un objeto que indica la capacidad de lo que éste puede hacer.

```
class Persona {
# Propiedades
# Métodos

    function   caminar()   {
        #...Código a ejecutar
    }

}
```

Propiedades y atributos

Las propiedades y atributos, son variables que contienen datos asociados a un objeto.

```
class Persona {
# Propiedades
# Métodos

    public $nombre;
    public $edad;
    public $altura;

    function   caminar()   {
        #...Código a ejecutar
    }

}
```

EJEMPLO DE CÓDIGO IMPERATIVO O ESPAGUETI VS POO

Muchas veces en PHP se busca encapsular, a través de una función, lógica propia a un solo bloque de código. Esto se le conoce como código espagueti.

```php
<?php

$automovil1 = array(
    'marca' => 'Toyota',
    'modelo' => 'Corolla'
    );
$automovil2 = array(
    'marca' => 'Volkswagen',
    'modelo' => 'Jetta'
    );

function imprimir($automovil){
    echo "<p>Hola! soy un $automovil[marca] modelo $automovil[modelo]</p>";
}

imprimir($automovil1);
imprimir($automovil2);

?>
```

Sin embargo, esta estrategia falla por varias razones. Al ver el código anterior pregúntese lo siguiente:

•Qué pasa cuando se empiezan a tener muchas propiedades? O si se comienzan a tener muchas funciones? Qué pasa si queremos otro conjunto de funciones y los nombres se repiten? Y es aquí donde la POO viene para salvarnos de la perdición.

```php
<?php
class Automovil
{
    public $marca;
    public $modelo;

    public function imprimir() {
        echo "<p>Hola! soy un $this->marca modelo $this->modelo</p>";
    }
}

$a = new Automovil();
$a->marca = "Toyota";
$a->modelo = "Corolla";
$a->imprimir();

$b = new Automovil();
$b->marca = 'Volkswagen';
$b->modelo = 'Jetta';
$b->imprimir();
```

El mismo código del ejemplo anterior, pero en este caso orientado a objetos. En el código vemos una clase llamada Automóvil que posee dos propiedades y un método.

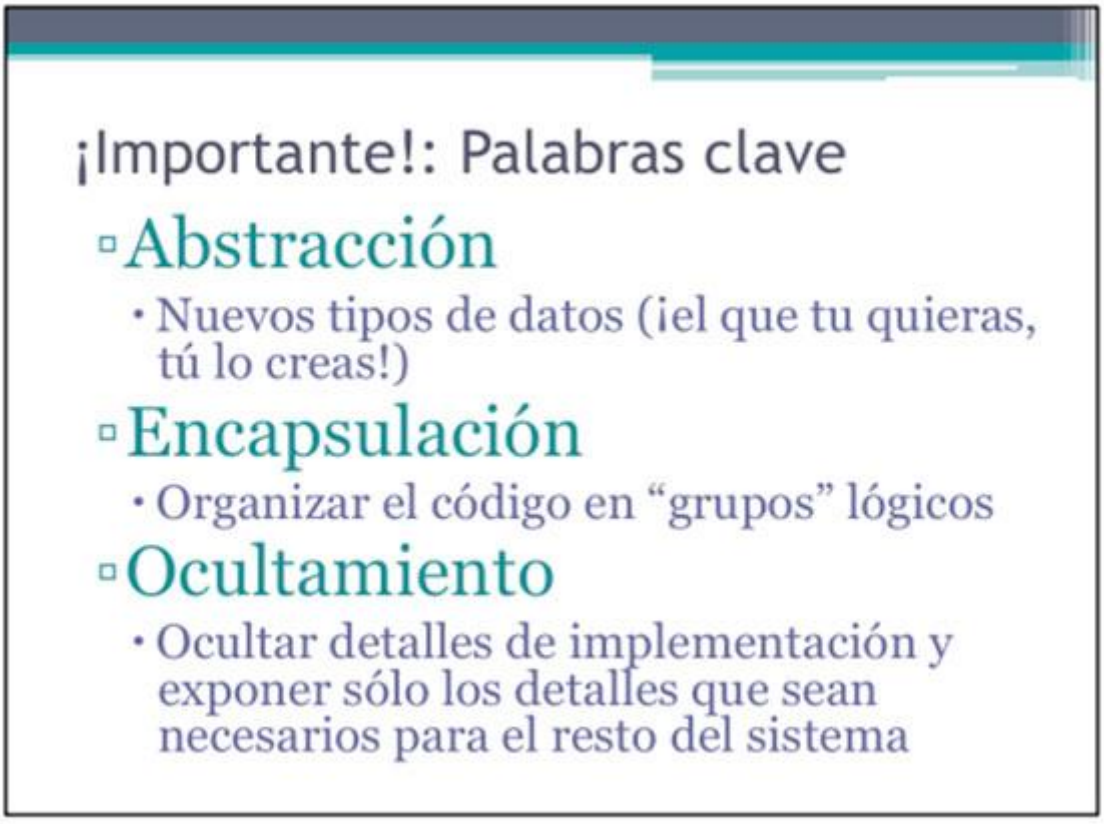
Más adelante explicaremos con mayor detalle qué significan cada una de estas cosas. Por el momento es interesante observar como en un sólo bloque de código (agrupado entre las palabras claves **class Automovil{…}** Tenemos la información que trabaja un objeto “Automóvil”.

Más aún, **al momento de instanciar**, es decir, realizar copias de un automóvil, vemos como la POO nos hace la vida fácil. **En el código anterior $a y $b** son copias distintas, permitiendo así que **$a tenga una propiedad ‘modelo’** que es propia y **distinta de la propiedad ‘modelo’ de la variable $b.**

¿QUÉ ES UN OBJETO COPIA?

- Un objeto permite generar orden al encapsular las variables
- Las funciones no están agrupadas, no se sabe cuáles trabajan con qué variables. Los métodos sí.
- Dos clases distintas pueden tener métodos con los mismos nombres (las funciones globales son únicas)
- Los objetos tienen una estructura definida mientras que los arreglos son más volátiles

En ese orden de ideas, el código anterior cumple con los siguientes principios:



EMPECEMOS CON POO BÁSICA

Como ya se ha reflejado antes, toda clase consta de la palabra clave class seguido del nombre de la clase y un bloque de código entre llaves.

Dentro del bloque de código se pueden crear **tres tipos de bloques básicos**:

· Constantes

· Variables

· Métodos

Una vez creadas dentro de las llaves, tanto la constante, como la variable, como la función pertenecen a la clase, y para ser utilizadas hay que acceder a través de la clase.

Recuerda que…

Una analogía apropiada (pero muy básica) al momento de pensar en una clase es pensar en un molde del cual se van a extraer múltiples “copias” u “objetos” similares. A diferencia de un objeto físico, en este caso las copias serán dinámicas y pueden cambiar su comportamiento y estructura al momento de ejecutar un programa.

- Una clase es un agrupación de variables, funciones y constantes.
- Una variable en una clase se llama una "propiedad"
- Una función en una clase se llama un "método"
- Las clases son globales, son accesibles desde cualquier lugar sin usar "global"

Aquí 'Automovil' es la clase y la variable **$a es un objeto (instancia o copia personalizada)** sobre la clase Automóvil. La **palabra clave new** hace que Automóvil junto a todas sus propiedades y funciones se copien a $a.

```php
<?php
class Automovil
{
    public $marca = 'valor por defecto';
    public $modelo;
}

$a = new Automovil();
$a->marca = "Toyota";
$a->modelo = "Corolla";
```

Algunas especificaciones:

- Un objeto es una variable más (como los arreglos o enteros), sólo que actúa como una copia de la clase.
- Podemos tener todas las copias (objetos) que se deseen de la misma clase.
- Un objeto suele denominarse como una instancia de una clase

Según el Manual Oficial de PHP, para definir una clase se deben cumplir con los siguientes pasos:

[…] "La definición básica de clases comienza con la palabra clave class, seguido por un nombre de clase, continuado por un par de llaves que encierran las definiciones de las propiedades y métodos pertenecientes a la clase. El nombre de clase puede ser cualquier etiqueta válida que no sea una palabra reservada de PHP. Un nombre válido de clase comienza con una letra o un guion bajo, seguido de la cantidad de letras, números o guiones bajos que sea." […]

Reglas de Estilo sugeridas

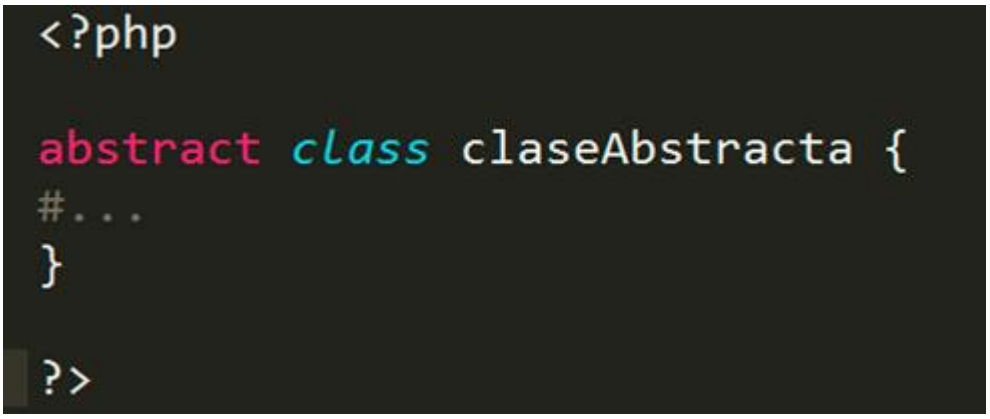
Utilizar CamelCase para el nombre de las clases. La llave de apertura en la misma línea que el nombre de la clase, permite una mejor legibilidad del código.

Herencia de Clases

Los objetos pueden heredar propiedades y métodos de otros objetos. Para ello, PHP permite la “extensión” (herencia) de clases, cuya característica representa la relación existente entre diferentes objetos. Para definir una clase como extensión de una clase “padre” se utiliza la palabra clave extends.

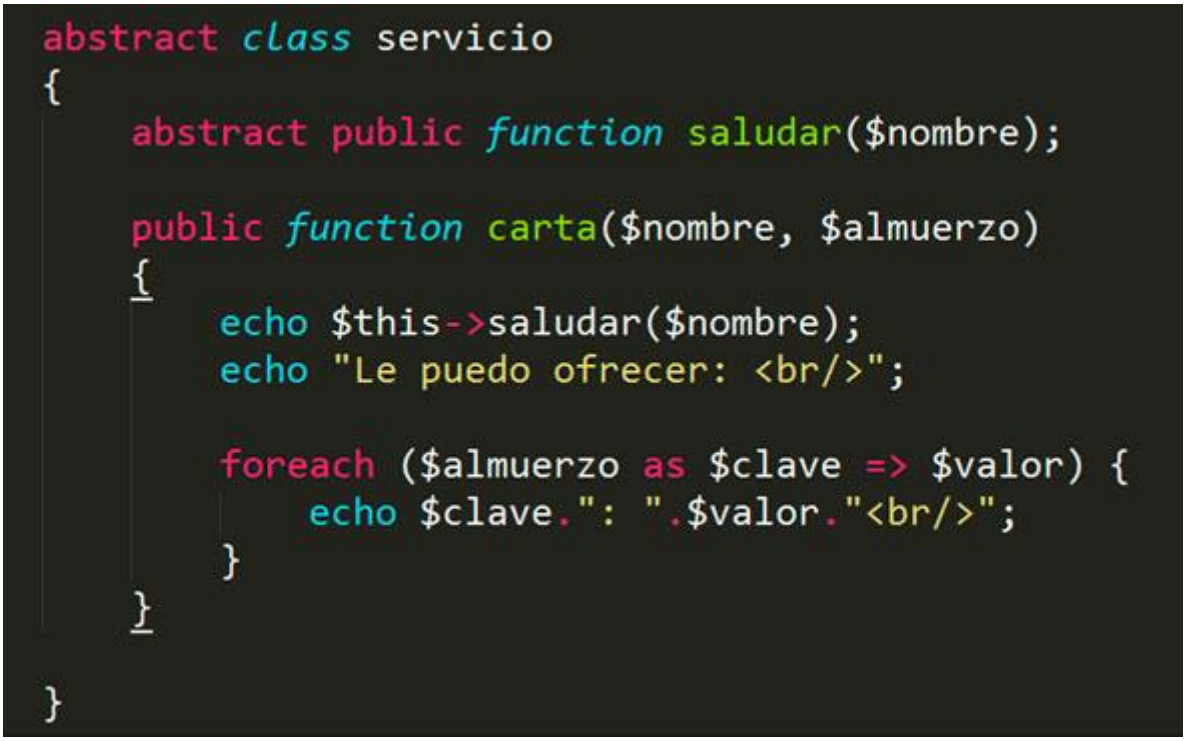
Declaración de clases abstractas

Las clases abstractas son aquellas que no necesitan ser instanciadas pero sin embargo, serán heredadas en algún momento. Se definen anteponiendo la palabra clave abstract:



Este tipo de clases, será la que contenga métodos abstractos (que veremos más adelante) y generalmente, su finalidad, es la de declarar clases “genéricas” que necesitan ser declaradas pero a las cuales, no se puede otorgar una definición precisa **(No se pueden instanciar),** de eso, **se encargarán las clases que la hereden).**

Un ejemplo más avanzado seria:



```php
abstract class servicio
{
    abstract public function saludar($nombre);

    public function carta($nombre, $almuerzo)
    {
        echo $this->saludar($nombre);
        echo "Le puedo ofrecer: <br/>";

        foreach ($almuerzo as $clave => $valor) {
            echo $clave.": ".$valor."<br/>";
        }
    }

}
```

Se crea una clase abstracta llamada servicio la cual contendrá 2 métodos:

- carta($nombre, $almuerzo) la cual recibirá el nombre de la persona y lo enviara al método abstracto saludar, el cual se debe sobrescribir en la clase que lo herede. También recibirá un array, el cual se recorrerá y mostrara su clave y valor, haciendo similitud a una carta de restaurante.

- Se crea una función abstracta **saludar($nombre)** la cual recibe el nombre de la persona y deberá ser **obligatoriamente sobrescrita** en la clase que extienda de servicio.

Lo que realizaremos ahora, será una clase que herede los métodos de la clase abstracta y sobrescriba el método de saludar.

```php
class mesero extends servicio
{
    public function saludar($nombre)
    {
        return "Buenas tardes señor(a): ".$nombre."<br/>";
    }

}

$mesero1 = new mesero();
$nombrePersona = "Yhoan Galeano";
$almuerzo = array("Sopa" => "Fideos",
                  "Seco" => "Carne Res, Papitas, Ensalada",
                  "Jugo" => "Mora"
                 );
$mesero1->carta($nombrePersona, $almuerzo);
```

Como se puede visualizar en el anterior código se extiende de la clase servicio y sobrescribe la función saludar. Además se realizan las pruebas al instanciar el objeto mesero.

Declaración de Clases finales En PHP

PHP desde su versión 5.1 incorpora clases finales que no pueden ser heredadas por otra. Se definen anteponiendo la palabra clave **final**.

```php
<?php

final class claseFinal {
#esta clase no podrá ser heredada
}

?>
```

Esto pasaría si se hace una herencia de una clase final:

```php
<?php

final class Automovil{

}

//Fatal error:
//Class Camion may not inherit from final class (Automovil)
class Camion extends Automovil{

}
```

¿QUÉ TIPO DE CLASE DECLARAR?

Hasta aquí, han quedado en claro, cuatro tipos de clases diferentes: clases instanciables, abstractas, heredadas y finales. ¿Cómo saber qué tipo de clase declarar? Todo dependerá, de lo que necesitemos hacer. Este cuadro, puede servirnos como guía básica:

Necesito...	Clases instanciables	Abstracta	Heredada
Crear una clase que pueda ser instanciada y/o heredada	X		
Crear una clase cuyo objeto guarda relación con los métodos y propiedades de otra clase			X
Crear una clase que solo sirva de modelo para otra clase(clase padre), sin que pueda ser instanciada		X	
Crear una clase que pueda instanciarse pero que no pueda ser heredada por ninguna otra clase			

OBJETOS E INSTANCIAS

Una vez que las clases han sido declaradas, será necesario crear los objetos y utilizarlos, aunque hemos visto que algunas clases, como las clases abstractas son solo modelos para otras, y por lo tanto no necesitan instanciar al objeto.

Instanciar una clase

Para instanciar una clase, solo es necesario utilizar la palabra clave **new**. El objeto será creado, asignando esta instancia a una variable (la cual, adoptará la forma de objeto).
Lógicamente, la clase debe haber sido declarada antes de ser instanciada, como se muestra a continuación:

```php
<?php

# declaro la clase
class Persona {
#...
}
# creo el objeto instanciando la clase
$persona = new Persona();

|
?>
```

Reglas para la instanciación de los objetos

Para una mejor legibilidad y manejo de las clases, se recomienda utilizar nombres de variables (objetos) descriptivos, siempre con **guion bajo** al comenzar, la primera letra debe ser en **minúscula**, y la siguiente palabra en **mayúscula**. Por ejemplo si el nombre de la clase es **nombreClase** como variable utilizar **$_nombreClase**. Esto permitirá una mayor legibilidad del código.

Definición de atributos o propiedades en PHP

Las propiedades representan ciertas características del objeto en sí mismo. Se definen anteponiendo la palabra clave **var** al nombre de la variable (propiedad). No es necesario utilizar la palabra reservada **var** para la definición de la variable, pues PHP la reconoce por defecto:

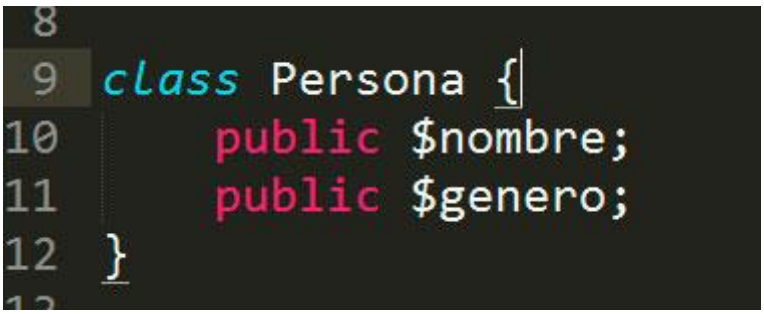
```php
<?php

class Persona {
    var $nombre;
    var $edad;
    var $genero;
}


?>
```

Las propiedades pueden gozar de diferentes características, como por ejemplo, la visibilidad: pueden ser públicas, privadas o protegidas. Como veremos más adelante, la visibilidad de las propiedades, es aplicable también a la visibilidad de los métodos.

Niveles de acceso

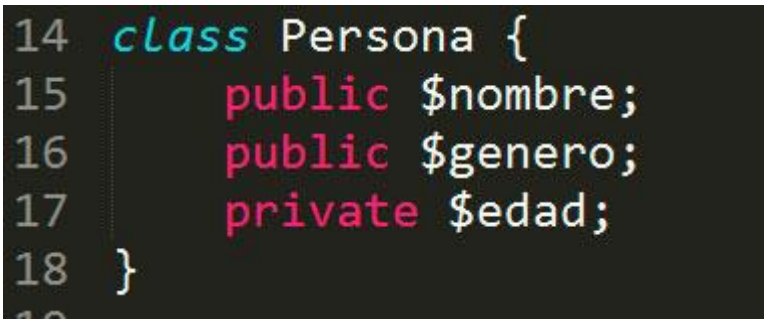
1. **Propiedades públicas**

Las **propiedades públicas** se definen anteponiendo la palabra clave **public** al nombre de la variable. Éstas, pueden ser accedidas desde cualquier parte de la aplicación, sin restricción.



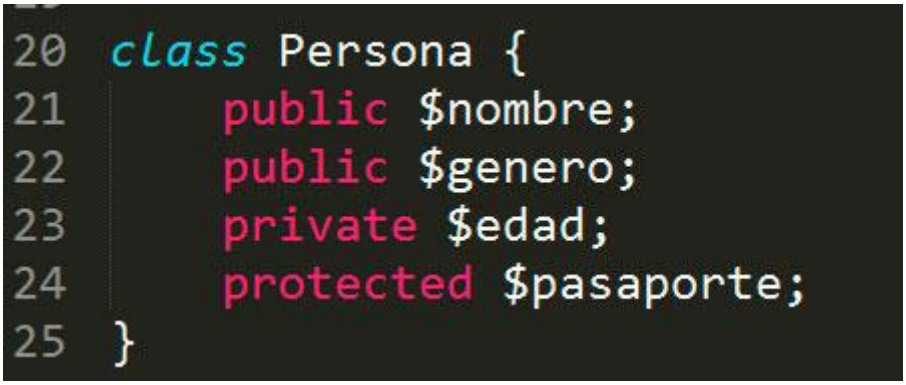
2. **Propiedades privadas**

Las **propiedades privadas** se definen anteponiendo la palabra clave **private** al nombre de la variable. Éstas solo pueden ser accedidas por la clase que las definió.



3. **Propiedades protegidas**

Las **propiedades protegidas** pueden ser accedidas por la propia clase que la definió, así como por las clases que la heredan, pero no, desde otras partes de la aplicación. Éstas, se definen anteponiendo la palabra clave **protected** al nombre de la variable:



4. Propiedades estáticas

Las **propiedades estáticas** representan una característica de "variabilidad" de sus datos, de gran importancia en PHP. Una propiedad declarada como estática, puede ser accedida sin necesidad de instanciar un objeto y su valor es estático (**es decir, no puede ser modificada para cada objeto, es como una variable global para todas las instancias que se crean de ese objeto**). Ésta, se define anteponiendo la palabra clave **static** al nombre de la variable:

```php
<?php
class Persona{
    public $nombre = "Yhoan";//Valor por defecto Yhoan
    public $apellido;
    public static $tipoSangre = 'A+';
}

//De esta manera puedo acceder a un atributo static
//No hay necesidad de crear el objeto, en cambio para los
//otros atributos es necesario crear la instancia
echo Persona::$tipoSangre."<br>";

//Lo modificamos y queda global para todos los elementos
Persona::$tipoSangre = "O+";
echo Persona::$tipoSangre."<br>";

//Necesito crear el objeto para poder acceder a los elementos
$p = new Persona();
$p->nombre = "Felipe";
echo $p->nombre."<br>";

//-----------Otro objeto--------
$p2 = new Persona();
$p2->nombre = "Andres";
echo $p2->nombre."<br>";

?>
```

ACCEDIENDO A LAS PROPIEDADES DE UN OBJETO:

Para acceder a la propiedad de un objeto, existen varias maneras de hacerlo. Todas ellas, dependerán del ámbito desde el cual se las invoque así como de su condición y visibilidad.

· **Acceso a variables desde el ámbito de la clase**

Se accede a una propiedad no estática dentro de la clase, utilizando la pseudo-**variable $this** siendo esta pseudo-variable una referencia al objeto mismo, se debe tener en cuenta que la variable que se llamara **no llevara adelante el $**:

```php
class Persona{
    public $nombre = "Yhoan";//Valor por defecto Yhoan
    public $apellido;
    public static $tipoSangre = 'A+';

    public function hola()
    {
        echo $this->nombre."<br>";
    }
}
```

Cuando la variable es estática, se accede a ella mediante el operador de resolución de ámbito, **doble dos-puntos ::** anteponiendo la palabra clave **self** o **parent** según si trata de una variable de la misma clase o de otra de la cual se ha heredado, respectivamente:

```php
class Persona{
    public $nombre = "Yhoan";//Valor por defecto Yhoan
    public $apellido;
    public static $tipoSangre = 'A+';

    public function hola()
    {
        echo $this->nombre."<br>";
        echo self::$tipoSangre."<br>";

        //Esto seria para la clase que extiende
        // echo parent::$tipoSangre."<br>";

    }
}
```

- **Acceso a variables desde el exterior de la clase**

Se accede a una propiedad **no estática** con la siguiente sintaxis:

$objeto->variable

Nótese además, que este acceso dependerá de la visibilidad de la variable. Por lo tanto, solo variables públicas pueden ser accedidas desde cualquier ámbito fuera de la clase o clases heredadas.

```php
// creo el objeto instanciando la clase
$p2 = new Persona();
// accedo a la variable NO estática
$p2->nombre = "Andres";
echo $p2->nombre."<br>";
```

Para acceder a una propiedad pública y estática el objeto no necesita ser instanciado, permitiendo así, el acceso a dicha variable mediante la siguiente sintaxis:

Clase::$variable_estática

```php
//De esta manera puedo acceder a un atributo static
//No hay necesidad de crear el objeto, en cambio para los
//otros atributos es necesario crear la instancia
echo Persona::$tipoSangre."<br>";
```

CONSTANTES:

Otro tipo de "propiedad" de una clase, son las constantes, aquellas que mantienen su valor de forma permanente y sin cambios. A diferencia de las propiedades estáticas que pueden ser declaradas dentro de una clase, las constantes solo pueden tener una visibilidad pública y no deben ser creadas dentro de las clases. El acceso a constantes es exactamente igual que al de otras propiedades.

```php
<?php

//Estos valores no puedes ser modificados

define('MENSAJE','<h1>Bienvenidos al Curso</h1>');
echo MENSAJE;

const MI_CONSTANTE = 'Este es el valor estatico de mi constante';
echo MI_CONSTANTE;

?>
```

MÉTODOS PHP

Cabe recordar, para quienes vienen de la programación estructurada, que el método de una clase, es un algoritmo igual al de una función. La única diferencia entre método y función, es que llamamos método a las funciones de una clase (en la POO), mientras que llamamos funciones, a los algoritmos de la programación estructurada.

La forma de declarar un método es anteponiendo la palabra clave function al nombre del método, seguido por un par paréntesis de apertura y cierre y llaves que encierren el algoritmo:

```php
<?php

# declaro la clase

class Persona {

    #propiedades
    #métodos
    function donar_sangre() {
        #...
    }
}


?>
```

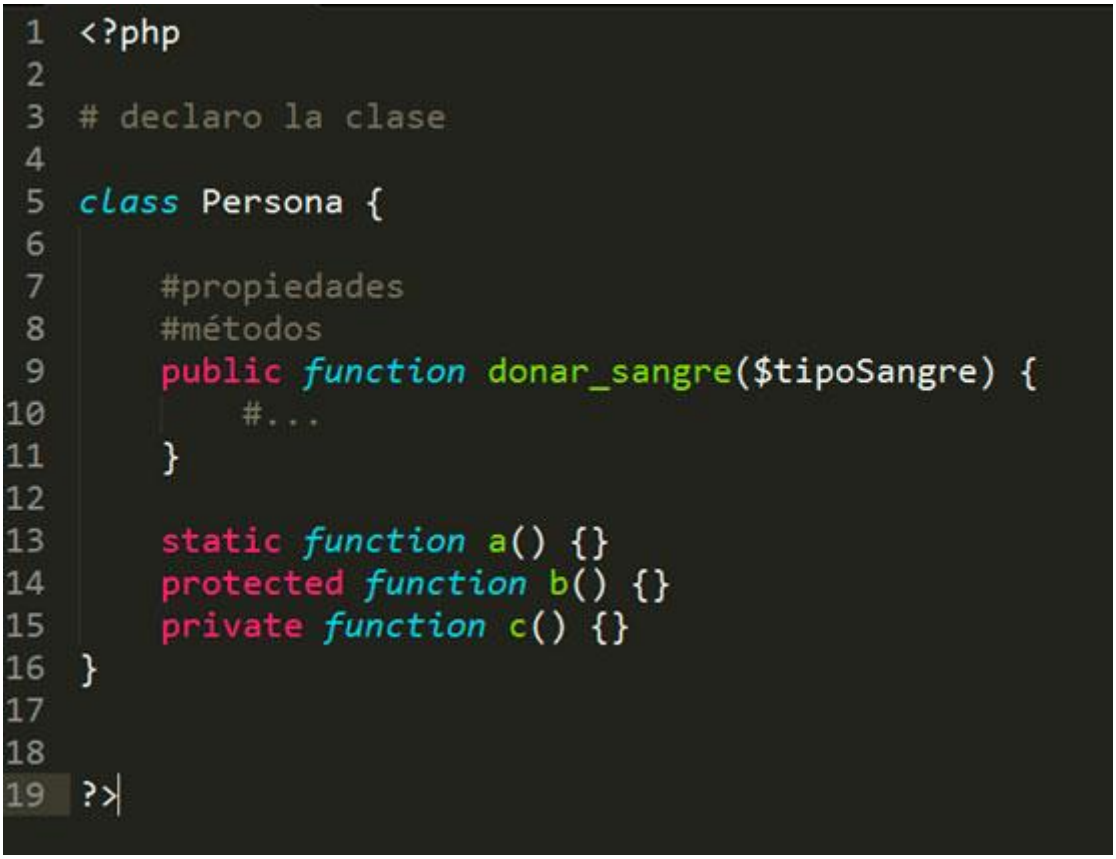
Al igual que cualquier otra función en PHP, los métodos recibirán los parámetros necesarios indicando aquellos requeridos, dentro de los paréntesis:

```php
<?php

# declaro la clase

class Persona {

    #propiedades
    #métodos
    function donar_sangre($tipoSangre) {
        #...
    }
}


?>
```

Métodos públicos, privados, protegidos y estáticos

Los métodos, al igual que las propiedades, pueden ser públicos, privados, protegidos o estáticos. La forma de declarar su visibilidad tanto como las características de ésta, es exactamente la misma que para las propiedades.

Los métodos abstractos por el contrario, solo pueden ser creados en las clases cuya declaración haya sido abstracta. De lo contrario mostrará un error al ejecutar el código.



```php
<?php

# declaro la clase

class Persona {

    #propiedades
    #métodos
    public function donar_sangre($tipoSangre) {
        #...
    }

    static function a() {}
    protected function b() {}
    private function c() {}
}


?>
```

Ver todos los ejemplos de los métodos para observar cómo se pueden acceder según su ámbito.

MÉTODOS ABSTRACTOS

A diferencia de las propiedades, los métodos, pueden ser abstractos como sucede con las clases.

El Manual Oficial de PHP, se refiere a los métodos abstractos, describiéndolos de la siguiente forma:

[…] "Los métodos definidos como abstractos simplemente declaran la estructura del método, pero no pueden definir la implementación. Cuando se hereda de una clase abstracta, todos los métodos definidos como abstract en la definición de la clase parent deben ser redefinidos en la clase child; adicionalmente, estos métodos deben ser definidos con la misma visibilidad (o con una menos restrictiva). Por ejemplo, si el método abstracto está definido como protected, la implementación de la función puede ser redefinida como protected o public, pero nunca como private." […]

Para entender mejor los métodos abstractos, podríamos decir que a grandes rasgos, los métodos abstractos son aquellos que se declaran inicialmente en una clase abstracta, sin especificar el algoritmo que implementarán, es decir, que solo son declarados pero no contienen un "código" que especifique qué harán y cómo lo harán.

Ejemplo:

Clase padre

```php
abstract class servicio
{
    abstract public function saludar($nombre);

    public function carta($nombre, $almuerzo)
    {
        echo $this->saludar($nombre);
        echo "Le puedo ofrecer: <br/>";

        foreach ($almuerzo as $clave => $valor) {
            echo $clave.": ".$valor."<br/>";
        }
    }

}
```

Clase hija:

```php
class mesero extends servicio
{
    public function saludar($nombre)
    {
        return "Buenas tardes señor(a): ".$nombre."<br/>";
    }

}
```

Como se puede visualizar la clase hija debe obligatoriamente sobrescribir el método saludar ya que fue declarado como abstracto en la clase padre.

Métodos mágicos en PHP

PHP, nos trae una gran cantidad de auto-denominados “métodos mágicos”. Estos métodos, otorgan una funcionalidad pre-definida por PHP, que pueden aportar valor a nuestras clases y ahorrarnos grandes cantidades de código. Lo que muchos programadores consideramos, ayuda a convertir a PHP en un lenguaje orientado a objetos, cada vez más robusto.

Entre los métodos mágicos, podemos encontrar los siguientes:

El Método Mágico __construct()

El método __construct() es aquel que será invocado de manera automática, al instanciar un objeto. Su función es la de ejecutar cualquier inicialización que el objeto necesite antes de ser utilizado.

```php
<?php
# declaro la clase

class Producto {
    #defino algunas propiedades
    public $nombre;
    public $precio;
    protected $estado;

    #defino el método set_estado_producto()
    protected function setEstado($estado) {
        $this->estado = $estado;
    }

    public function ver()
    {
        echo "Hola soy el producto: ".$this->nombre."<br/>";
        echo "Tengo un precio de: ".$this->precio."<br/>";
        echo "Me encuentro con un estado de: ".$this->estado."<br/>";
    }

    # constructor de la clase
    function __construct() {
        $this->setEstado('en uso');
    }
}

$_producto = new Producto();
$_producto->nombre = "Leche Colanta";
$_producto->precio = 2000;
$_producto->ver();

?>
```

En el ejemplo anterior, el constructor de la clase se encarga de definir el estado del producto como “en uso”, antes de que el objeto (Producto) comience a utilizarse. Si se agregaran otros métodos, éstos, podrán hacer referencia al estado del producto, para determinar si ejecutar o no determinada función. Por ejemplo, no podría mostrarse a la venta un producto “en uso por el sistema”, ya que a éste, se le podría estar modificando el precio.

El método mágico __destruct()

El método __destruct() es el encargado de liberar de la memoria, al objeto cuando ya no es referenciado. Se puede aprovechar este método, para realizar otras tareas que se estimen necesarias al momento de destruir un objeto.

```php
<?php
# declaro la clase

class Producto {
    #defino algunas propiedades
    public $nombre;
    public $precio;
    protected $estado;

    #defino el método set_estado_producto()
    protected function setEstado($estado) {
        $this->estado = $estado;
    }

    public function ver()
    {
        echo "Hola soy el producto: ".$this->nombre."<br/>";
        echo "Tengo un precio de: ".$this->precio."<br/>";
        echo "Me encuentro con un estado de: ".$this->estado."<br/>";
    }

    # constructor de la clase
    function __construct() {
        $this->setEstado('en uso');
    }

    # destructor de la clase
    function __destruct() {
        $this->setEstado('liberado');
        echo "El objeto ha sido destruido <br/>";
        echo "El objeto ha sido: ".$this->estado."<br/>";
    }
}

$_producto = new Producto();
$_producto->nombre = "Leche Colanta";
$_producto->precio = 2000;
$_producto->ver();

?>
```

Otros métodos mágicos

PHP nos ofrece otros métodos mágicos tales como __call, __callStatic, __get, __set, __isset, __unset, __sleep, __wakeup, __toString, __invoke, __set_state y __clone.

Puede verse una descripción y ejemplo de su uso, en el sitio Web oficial de PHP: http://www.php.net/manual/es/language.oop5.magic.php

INTERFACES

Una interfaz es un conjunto de métodos abstractos y de constantes cuya funcionalidad es la de determinar el funcionamiento de una clase, es decir, funciona como un molde o como una plantilla. Al ser sus métodos abstractos estos no tiene funcionalidad alguna, sólo se definen su tipo, argumento y tipo de retorno.

Para implementar una interface es necesario que la clase que quiera hacer uso de sus métodos utilice la palabra reservada **implements**. La clase que la implemente, de igual modo debe sobrescribir los métodos y añadir su funcionalidad.

La construcción de una Interfaz es la siguiente:

```php
<?php

interface Filtro{
  public function filtrar($elem);
}

class Filtro_Vocales implements Filtro{
  public function filtrar($cadena){
    return preg_replace('/[aeiou]/', '', $cadena);
  }
}

$a = new Filtro_Vocales();
echo $a->filtrar('Dinosaurio');

?>
```

AGREGACIÓN Y COMPOSICIÓN EN PHP

En PHP también podemos mapear las diferentes relaciones que realizamos en los diagramas UML, como lo son la agregación y composición. Para este tema, contaremos con una breve explicación e implementación.

Agregación:

Según lo estudiado en UML, sabemos que una agregación es un tipo de relación dinámica, en donde **el tiempo de vida del objeto incluido es independiente** del que lo incluye (el objeto base utiliza al incluido para su funcionamiento).

Para empezar, realizaremos una clase con nombre **Clase1**, la cual contiene dentro de ella un método llamado saludar.

```php
class Clase1
{
    function saludar()
    {
        echo "Hola <br/>Soy una agregación <br/><br/>";
    }
}
```

A continuación, realizaremos una clase que recibe como parámetro la instancia de la **Clase1, sin importar donde se halla realizado la instancia de este.** La idea es que la instancia de la Clase1 no dependa de la instancia de la Clase2, pues en esta, solamente se utilizara.

```php
class Clase2
{
    function saludar($InstanciaC1)
    {
        $InstanciaC1->saludar();
    }
}
```

Su implementación seria de la siguiente manera:

```php
$InstanciaC1 = new Clase1();
$InstanciaC2 = new Clase2();

$InstanciaC2->saludar($InstanciaC1);
$InstanciaC1->saludar();
```

En el código anterior, creamos una instancia para la Clase1 y otra instancia para la Clase2. Al llamar el método **saludar()** , que se encuentra en la $InstanciaC2, pasamos como parámetro la instancia de Clase1, pero, en la siguiente línea podemos volver a llamar a $InstanciaC1 y a su método saludar. **Esto demuestra la independencia entre las dos clases, puesto que Clase1 no muere al realizar una instancia de Clase2.**

Composición:

Según UML, una composición es un tipo de relación estática, en donde el tiempo de vida del objeto incluido está condicionado por el tiempo de vida del que lo incluye (el Objeto base se construye a partir del objeto incluido, es decir, es "parte/todo").

Para ver su funcionamiento e implementación, trabajaremos con las clases anteriores.

```php
class Clase1
{
    function saludar()
    {
        echo "Hola <br/>Soy una composición";
    }
}
```

Creamos la misma clase llamada Clase1 y en su método saludar nos especifica que es una composición.

```php
class Clase3
{
    private $InstanciaC1;
    function saludar()
    {
        $InstanciaC1 = new Clase1();
        $InstanciaC1->saludar();
    }
}
```

En el código anterior, observamos que se creó otra clase llamada Clase3, en la cual se tiene una propiedad con tipo de visualización private, lo cual indica que solo podrá ser accedida desde esta clase. A continuación creamos el método saludar y dentro de este, realizamos la instancia de la Clase1. Para este caso, esta instancia solo vivirá mientras esté viva la instancia de la Clase3.

Su implementación sería de la siguiente manera:

```php
$InstanciaC3 = new Clase3();
$InstanciaC3->saludar();
```

Como vemos, la Clase1 es dependiente de la Clase2.