Introducción a PHP

por Matías Godoy
mattogodoy@gmail.com

Índice

Introducción

¿Qué es PHP?

PHP (acrónimo recursivo de PHP: Hypertext Preprocessor) es un lenguaje rápido pese a ser interpretado. Es multiplataforma y dispone de una gran cantidad de bibliotecas para el desarrollo de aplicaciones Web.

Está basado en herramientas de software libre (Apache, MySQL, etc.), es decir, no hay que pagarlas; además proporciona los mecanismos para poder trabajar con casi cualquier base de datos (sea software libre o no) y servidor web.

A diferencia de HTML, PHP se ejecuta en el servidor. Es decir, cuando pedimos a nuestro servidor web una página PHP, el código se ejecuta en el servidor y genera una página HTML que es enviada al navegador del cliente.

PHP nos permite embeber su pequeños fragmentos de código dentro de la página HTML y realizar determinadas acciones de una forma fácil y eficaz. Por otra parte, ofrece un sinfín de funciones para la explotación de bases de datos de una manera simple, sin complicaciones.

Lo que distingue a PHP de algo como Javascript del lado del cliente es que el código es ejecutado en el servidor, generando HTML y enviándolo al cliente. El cliente recibirá el resultado de ejecutar el script.

Lo mejor de usar PHP es que es extremadamente simple para el principiante, pero a su vez ofrece muchas características avanzadas para los programadores profesionales.

Sintaxis

La sintaxis de PHP

Sintaxis se refiere a las formas y estructura que debe de tener un lenguaje para su interpretación. Formado por un conjunto de reglas básicas que debemos de tener en cuenta a la hora de escribir. Un ejemplo podemos verlo a la hora de comunicarnos con otra persona, debemos de tener una base que nos haga explicarnos de una forma concreta y así conseguir que el receptor nos entienda.

Un script de PHP puede ser situado en cualquier parte de un documento HTML. Comienza con **<?php** y finaliza con **?>**

```
<html>
    <head>
        <title>Ejemplo</title>
    </head>
    <body>

        <?php
            echo "¡Hola, soy un script de PHP!";
        ?>

    </body>
</html>
```

El anterior ejemplo imprime el texto "¡Hola, soy un script de PHP!" en una página web.

Nótese que las sentencias PHP terminan con punto y coma (;). Esto indica al intérprete que esa línea de código finaliza y debe pasar a la próxima. Si nos olvidamos de cerrar una sentencia, el script nos dará un error.

Comentarios

Un comentario en PHP es una línea que no es ejecutada por el intérprete, y nos permite escribir información útil que pueda servir a otros desarrolladores que trabajen con el código, o a nosotros mismos en caso de que volvamos a trabajar con él un tiempo después.

PHP permite tres tipos de comentarios:

```
<?php
    // Este es un comentario de una sola línea

    # Este también es un comentario de una sola línea

    /*
    Este es un bloque de comentarios
    que puede contener mas de una
    línea.
    */
?>
```

Ninguno de estos comentarios será ejecutado por el intérprete.

Recomendamos que se utilicen comentarios siempre que sea posible para describir el funcionamiento del código que escribimos. De esta manera facilitamos en gran medida la lectura.

Mayúsculas y minúsculas

En PHP, todas las funciones, clases y palabras reservadas (if, else, while, echo, etc.) no se ven afectadas por mayúsculas y minúsculas.

En el siguiente ejemplo, todas las sentencias son válidas, y devuelven el mismo resultado:

```php
<?php
    ECHO "Hola Mundo!";
    echo "Hola Mundo!";
    EcHo "Hola Mundo!";
?>
```

Por el contrario, las variables en PHP sí son sensibles a mayúsculas y minúsculas:

```php
<?php
    $color = "rojo";
    echo "Mi coche es " . $color;
    echo "Mi casa es " . $COLOR;
    echo "Mi bote es " . $coLOR;
?>
```

En el ejemplo anterior, sólo la primer sentencia mostrará el valor de la variable *$color* porque *$color*, *$COLOR*, y *$coLOR* son tratados como 3 variables diferentes.

Variables

¿Qué son?

Las variables son "contenedores" para almacenar información a la que podremos acceder luego:

```
1  <?php
2      $x = 5;
3      $y = 6;
4      $z = $x + $y;
5      echo $z;
6  ?>
```

En las líneas 2 y 3 estamos declarando las variables *$x* e *$y*, y asignándoles los valores 5 y 6 respectivamente.

Reglas de las variables en PHP:

- Comienzan con el signo dólar ($), seguido del nombre de la variable.
- El nombre de una variable debe comenzar con una letra o con un guión bajo.
- El nombre no puede comenzar con números ni signos de puntuación.
- El nombre sólo puede contener caracteres alfanuméricos y guiones bajos (A-z, 0-9 y _).
- Los nombres de variables son sensibles a mayúsculas y minúsculas (no es lo mismo *$Nombre* que *$nombre*).

Declaración

A diferencia de otros lenguajes, PHP no tiene un comando para declarar variables.

Las variables son creadas en el momento en que les asignamos un valor por primera vez.

```
1  <?php
2      $txt = "Hola Mundo!";
3      $x = 5;
4      $y = 10.5;
5  ?>
```

Esto significa que la declaración de las variables se efectúa implícitamente al momento de la asignación.

En el anterior ejemplo, podemos ver que las variables no son de un tipo específico. PHP las convierte automáticamente al tipo correspondiente.

Ámbito

Las variables pueden ser declaradas en cualquier parte del programa.

El ámbito de una variable se refiere a la parte del programa en que esta puede ser referenciada o utilizada.

En PHP existen 3 tipos de ámbitos de variable:

- Global
- Local
- Estático

Ámbitos Local y Global

Una variable declarada fuera de una función, tiene un ámbito global, y sólo puede ser accedida fuera de cualquier función.

```php
<?php
    $x = 5; // Ámbito global

    function prueba(){
        $y = 10; // Ámbito local
        echo "El valor de x es: $x";
        echo "El valor de y es: $y";
    }

    prueba();

    echo "El valor de x es: $x";
    echo "El valor de y es: $y";
?>
```

El resultado de ejecutar este programa es:

```
El valor de x es:
El valor de y es: 10

El valor de x es: 5
El valor de y es:
```

En este ejemplo tenemos 2 variables; *$x* e *$y*, y una función llamada *prueba()*.
Dado que *$x* está declarada fuera de la función, es una variable **global**.
Por otra parte, *$y* está declarada dentro de la función, por lo que es visible sólo dentro del ámbito de la propia función. Es una variable **local**.

Al momento de imprimir los valores de las dos variables desde dentro de la función, se imprime sólo el valor de *$y*, dado que fue declarada dentro del mismo ámbito, pero no se imprimió el valor de *$x* ya que fue creada fuera de la función.

Si quisiéramos acceder desde una función al valor de las variables globales, tanto para leerlas como para modificarlas, habría que redeclararlas dentro de la propia función utilizando **global**.

```php
<?php
    $x = 5;
    $y = 10;

    function prueba() {
        global $x, $y;
        $y = $x + $y;
    }

    prueba();
    echo $y; // Devuelve 15
?>
```

Ámbito estático

Normalmente, cuando una función termina su ejecución, todas sus variables locales son eliminadas.
En algunas ocasiones, queremos que estas variables no se eliminen, sino que por el contrario, mantengan su valor. Para ello, utilizamos **static** a la hora de declararlas.

```php
<?php

    function prueba() {
        static $x = 0;
        echo $x;
        $x++;
    }

    prueba();
    prueba();
    prueba();

    /*
    En este punto, la variable local $x de la
    función prueba() valdrá 3.
    */

?>
```

Así, cada vez que la función es llamada, la variable *$x* mantendrá la información que contiene a lo largo del tiempo.

Nota: La variable sigue siendo local de la función.

Tipos de datos

En PHP encontramos diferentes tipos de datos, cada uno con sus características propias:

Strings

Son secuencias de caracteres, como “Hola, mundo!”. Un string puede contener cualquier texto entre las comillas. Estas comillas pueden ser dobles o simples.

```php
<?php

    $x = "mundo";
    echo "Hola, $x !";
    // Devuelve "Hola, mundo !"

    echo 'Hola, $x !';
    // Devuelve "Hola, $x !"
?>
```

Como se ve en el ejemplo, usando comillas dobles podemos agregar nombres de variables dentro del string, que a la hora de imprimirse serán reemplazados por su contenido.
Por otra parte, el uso de comillas simples implica una salida textual del string a la hora de imprimirlo.

Integers

Como su nombre lo indica, este tipo de datos se refiere a números enteros, sin decimales, tanto positivos como negativos.

```php
<?php
    $x = 5985;

    $x = -345; // número negativo

    $x = 0x8C; // número hexadecimal (comenzado en 0x)

    $x = 047; // número octal (comenzado en 0)
?>
```

Float

Un número de coma flotante es un número que contiene decimales, sea positivo o negativo.

```php
<?php
    $x = 10.365;

    $x = -3.141592;
?>
```

Booleans

Los datos booleanos solo pueden ser **verdaderos** o **falsos**.

```php
<?php
    $x = true;

    $y = false;
?>
```

Arrays

Un array puede almacenar múltiples valores en una sola variable.

```php
<?php
    $semana = array("Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo");

    echo $semana[0]; // Devuelve "Lunes"

    echo $semana[3]; // Devuelve "Jueves"
?>
```

Objetos

Un objeto es el resultado de la instanciación de una clase. Para ello, primero debemos declarar una clase como “molde” de ese objeto.

Una clase es una estructura que puede contener atributos y métodos.

```php
<?php
    class Moto{
        var $color;

        function Moto($color){
            $this->color = $color;
        }

        function getColor(){
            return $this->color;
        }
    }
?>
```

NULL

El tipo especial NULL, o nulo, representa a una variable que no tiene valor. El valor NULL identifica cuando una variable está vacía o no.

```php
<?php
    $x = "Hola mundo!";

    $x = null;
?>
```

Operadores

En PHP tenemos disponibles distintos operadores que nos permiten realizar cálculos, asignaciones, comparaciones y otras operaciones que veremos a continuación.

Operadores aritméticos

Nos permiten realizar cálculos matemáticos.

Operador	Nombre	Ejemplo	Resultado
+	Suma	$x + $y	Sumatoria de $x mas $y
-	Resta	$x - $y	Diferencia entre $x e $y
*	Multiplicación	$x * $y	Producto de $x por $y
/	División	$x / $y	Cociente de $x dividido $y
%	Módulo	$x % $y	Resto de $x dividido $y

En el siguiente ejemplo vemos los diferentes resultados de la aplicación de cada uno de los operadores aritméticos.

```php
<?php
    $x = 10;
    $y = 6;
    echo ($x + $y); // Imprime 16
    echo ($x - $y); // Imprime 4
    echo ($x * $y); // Imprime 60
    echo ($x / $y); // Imprime 1.6666666666667
    echo ($x % $y); // Imprime 4
?>
```

Operadores de asignación

En PHP utilizamos operadores de asignación para escribir un valor en una variable.

El operador básico es "= ", pero tiene algunas variaciones que simplifican algunas operaciones.

Asignación	Equivalente a	Descripción
x = y		Se asigna el valor de la expresión de la derecha al operando de la izquierda.
x += y	x = x + y	Adición
x -= y	x = x - y	Sustracción
x *= y	x = x * y	Multiplicación
x /= y	x = x / y	División
x %= y	x = x % y	Módulo

Ejemplos de asignaciones:

```php
<?php
    $x = 10;
    echo $x; // Imprime 10

    $y = 20;
    $y += 100;
    echo $y; // Imprime 120

    $z = 50;
    $z -= 25;
    echo $z; // Imprime 25

    $i = 5;
    $i *= 6;
    echo $i; // Imprime 30

    $j = 10;
    $j /= 5;
    echo $j; // Imprime 2

    $k = 15;
    $k %= 4;
    echo $k; // Imprime 3
?>
```

Operadores de cadenas

Para unir dos o mas cadenas, utilizamos el operador “.”

Operador	Nombre
.	Concatenación
.=	Concatenación por asignación

Algunos ejemplos:

```php
<?php
    $a = "Hola";
    $b = $a . " mundo!";
    echo $b; // Imprime "Hola, mundo!"

    $x = "Hola";
    $x .= " mundo!";
    echo $x; // Imprime "Hola, mundo!"
?>
```

Operadores de incremento y decremento

En algunas ocasiones nos es útil sumar o restar 1 a una variable. Por ejemplo, para crear un contador de visitas, o una cuenta regresiva.

Operador	Nombre	Descripción
++$x	Pre-incremento	Incrementa $x en 1, y luego devuelve $x
$x++	Post-incremento	Devuelve $x, y luego incrementa $x en 1
--$x	Pre-decremento	Decrementa $x en 1, y luego devuelve $x
$x--	Post-decremento	Devuelve $x, y luego decrementa $x en 1

Veamos su aplicación:

```php
<?php
    $x = 10;
    echo ++$x; // Imprime 11

    $y = 10;
    echo $y++; // Imprime 10

    $z = 5;
    echo --$z; // Imprime 4

    $i = 5;
    echo $i--; // Imprime 5
?>
```

Operadores de comparación

Son utilizados para comparar dos valores de cualquier tipo.

Operador	Nombre	Ejemplo	Resultado
==	Igual	$x == $y	Es verdadero si $x es igual a $y
===	Idéntico	$x === $y	Es verdadero si $x es igual a $y, y ambos son del mismo tipo
!=	Distinto	$x != $y	Es verdadero si $x es distinto de $y
<>	Distinto	$x <> $y	Es verdadero si $x es distinto de $y
!==	No idéntico	$x !== $y	Es verdadero si $x es distinto de $y, o no son del mismo tipo.
>	Mayor	$x > $y	Es verdadero si $x es mayor que $y
<	Menor	$x < $y	Es verdadero si $x es menor que $y
>=	Mayor o igual	$x >= $y	Es verdadero si $x es mayor o igual a $y
<=	Menor o igual	$x <= $y	Es verdadero si $x es menor o igual a $y

Ejemplo:

```php
<?php
    $x = 100;
    $y = "100";

    $resultado = $x == $y; // $resultado = verdadero

    $resultado = $x === $y; // $resultado = falso

    $resultado = $x != $y; // $resultado = falso

    $resultado = $x !== $y; // $resultado = verdadero


    $a = 50;
    $b = 90;

    $resultado = $a > $b; // $resultado = falso

    $resultado = $a < $b; // $resultado = verdadero
?>
```

Operadores lógicos

Sirven para efectuar operaciones con valores booleanos (verdadero o falso).

Operador	Nombre	Ejemplo	Resultado
&&	And	$x && $y	Es verdadero si $x e $y son verdaderos
||	Or	$x || $y	Es verdadero si $x **ó** $y son verdaderos
!	Not	!$x	Es verdadero si $x no lo es

Ejemplo:

```php
<?php
    $x = true;
    $y = false;

    $resultado = $x && $y; // $resultado = falso

    $resultado = $x || $y; // $resultado = verdadero

    $resultado = !$x; // $resultado = falso
?>
```

Condicionales

A la hora de desarrollar, en varias ocasiones nos encontraremos con que tenemos que tomar decisiones basándonos en la información que tenemos disponible. Para ello, PHP nos ofrece algunas herramientas que nos serán de gran utilidad.

If -Else

El condicional **if** se utiliza para ejecutar un bloque de código únicamente si se cumple una condición específica.

```php
<?php
    $edad = 20;

    if($edad > 18){
        echo "Eres mayor de edad!";
    }
?>
```

Para algunos casos, tal vez necesitemos ejecutar un bloque de código si se cumple una condición, pero si no se cumple, ejecutar otro bloque distinto. Para esto utilizamos **else**.

```php
<?php
    $edad = 12;

    if($edad > 18){
        echo "Eres mayor de edad!";
    } else {
        echo "Eres menor";
    }
?>
```

Switch

Para el caso en que tengamos que ejecutar distintos bloques de código para varios casos diferentes, utilizamos **switch**, que ejecuta distintas acciones para varios casos posibles establecidos. En caso de que sea un caso no contemplado, se ejecuta una acción por defecto.

```php
<?php
    $color = "rojo";

    switch ($color){
        case "rojo":
            echo "Tu color favorito es rojo.";
            break;
        case "verde":
            echo "Tu color favorito es vede.";
            break;
        case "azul":
            echo "Tu color favorito es azul.";
            break;

        default:
        echo "Tu color favorito no es rojo, verde ni azul.";
    }
?>
```

Bucles

Es común que al desarrollar, necesitemos ejecutar un bloque de código más de una vez. Para no escribir varias veces el mismo código, utilizamos bucles.

En PHP existen 3 tipos de bucles.

While

Ejecuta un bloque de código mientras se cumpla una condición.

```php
<?php
    $x = 1;

    while($x <= 5){
        echo "El número es: $x";
        $x++;
    }
?>
```

Otra forma de hacerlo es utilizando **do**:

```php
<?php
    $x = 1;

    do {
        echo "El número es: $x";
        $x++;
    } while($x <= 5)
?>
```

La diferencia es que de esta manera, se controla que se cumpla la condición al final de la ejecución, por lo tanto, el bloque de código se ejecuta siempre al menos una vez.

For

Lo utilizamos cuando ya sabemos el número de veces que queremos ejecutar el bloque de código.
Este tipo de bucle utiliza 3 parámetros:

- Desde : Indica desde que valor empezar
- Hasta: Indica cuando detenerse
- Paso: Indica el valor del incremento en cada vuelta

```php
<?php
    for($i = 0; $i <= 10; $i++){
        echo "El número es: $i";
    }

    /*
    Imprime:

    El numero es: 0
    El numero es: 1
    El numero es: 2
    El numero es: 3
    El numero es: 4
    El numero es: 5
    El numero es: 6
    El numero es: 7
    El numero es: 8
    El numero es: 9
    El numero es: 10
    */
?>
```

En el siguiente ejemplo, le diremos que incremente el paso de 2 en 2:

```php
<?php
    for($i = 0; $i <= 10; $i += 2){
        echo "El número es: $i";
    }

    /*
    Imprime:

    El numero es: 0
    El numero es: 2
    El numero es: 4
    El numero es: 6
    El numero es: 8
    El numero es: 10
    */
?>
```

Foreach

Es aplicable sólo a arrays. Ejecuta el bloque de código tantas veces como elementos contenga el array, es decir, una vez por cada elemento.

```php
<?php
    $semana = array("Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo");

    foreach ($semana as $dia) {
        echo "El día es " . $dia;
    }

    /*
    Imprime

    El día es Lunes
    El día es Martes
    El día es Miércoles
    El día es Jueves
    El día es Viernes
    El día es Sábado
    El día es Domingo
    */
?>
```

Funciones

Para reutilizar un bloque de código que será ejecutado mas de una vez, lo recomendable es crear una función que contenga este bloque, y llamar a la función cada vez que necesitemos utilizarlo.

Definición

PHP nos permite definir nuestras propias funciones, aparte de las que ya tiene incluidas.

```php
<?php
    function escribirMensaje(){
        echo "Hola mundo!";
    }

    escribirMensaje(); // Llamada a la función
?>
```

Parámetros

A la hora de llamar a una función, podemos pasarle información por medio de parámetros.
Los parámetros deben ser especificados en la declaración de la función, y podemos agregar tantos como nos sea necesario, separándolos por comas.

```php
<?php
    function escribirNombre($nombre){
        echo "Mi nombre es $nombre";
    }

    escribirNombre("Matías"); // Imprime "Mi nombre es Matías"
?>
```

Ejemplo con varios parámetros:

```php
<?php
    function escribirMensaje($nombre, $edad, $ciudad){
        echo "Mi nombre es $nombre, tengo $edad años y vivo en $ciudad";
    }

    escribirMensaje("Matías", 27, "Madrid");
    // Imprime "Mi nombre es Matías, tengo 27 años y vivo en Madrid"
?>
```

Return

En algunos casos necesitaremos que una función nos devuelva algún valor. Para ello utilizamos **return**.

```php
<?php
    function sumar($x, $y){
        $resultado = $x + $y;

        return $resultado;
    }

    $valor = sumar(14, 6);
    echo $valor; // Imprime 18
?>
```

Programación Orientada a Objetos en PHP

¿Qué es la POO?

La Programación Orientada a Objetos es un paradigma de programación que se basa en objetos y sus interacciones. Se trata de una metodología de desarrollo, y no de un lenguaje en particular.

Conceptos fundamentales

Clase

Es el "molde" o plantilla que se utiliza para crear objetos de un tipo. Define sus propiedades y su comportamiento.
La clase define los atributos y métodos comunes a los objetos de ese tipo, pero luego, cada objeto tendrá sus propios valores y compartirán las mismas funciones.
Un objeto creado a partir de una determinada clase se denomina una **instancia** de esa clase.

Las clases en PHP se definen de la siguiente manera:

```php
<?php

    class Persona {
        private $nombre;
        private $apellido;
        private $edad;

        public function __construct($n, $e){
            $this->nombre = $n;
            $this->edad = $e;
        }

        public function esMayor(){
            if($this->edad >= 18){
                return true;
            } else {
                return false;
            }
        }

        public function getNombre(){
            return $this->nombre;
        }

        public function setApellido($a){
            $this->apellido = $a;
        }
    }

?>
```

Como podemos observar, una clase consta de:

- Identificador: Es el nombre de la clase. En este caso “Persona”. Los nombres de las clases comienzan siempre con mayúsculas.
- Constructor: Es un método especial que se ejecuta automáticamente cada vez que una clase es instanciada y no puede volver a ser llamado. Su objetivo fundamental es inicializar los atributos de un objeto creado. Es opcional y puede tener parámetros como en nuestro ejemplo.

- Atributos: Son propiedades de la clase. Contienen información sobre el objeto y determinan el estado del mismo. Varios objetos de una clase tendrán los mismos atributos, pero con valores diferentes. Se recomienda que sean privados para que solo los métodos de la clase puedan modificarlos.
- Métodos: Son funciones como las que vimos anteriormente, pero están definidas dentro de la clase y operan sobre los atributos de dicha clase. Al igual que los atributos, los métodos pueden ser públicos o privados. De ser privados, sólo pueden ser llamados desde dentro de la propia clase. Los métodos públicos pueden ser llamados desde fuera. Al conjunto de estos últimos lo llamamos la **interfaz** de la clase. Es lo que nos permite interactuar con ella.
- Getters y Setters: Son métodos dedicados a leer y escribir los atributos de la clase. Dado que los atributos deberían ser privados, los getters y setters serán la única manera de poder leerlos o modificarlos desde el exterior.

Objetos

Son instancias de una clase, y por tanto, heredan de ella los mismos atributos y métodos.

La POO intenta simular el mundo real a través del significado de objetos que contienen características y funciones. Estos objetos tienen un determinado estado (atributos), comportamiento (métodos) e identidad:

- El estado está compuesto de uno o varios atributos a los que se habrán asignado unos valores concretos.
- El comportamiento está definido por los métodos o mensajes a los que sabe responder dicho objeto, es decir, qué operaciones se pueden realizar con él.
- La identidad es una propiedad de un objeto que lo diferencia del resto.

Podemos pensar en un objeto como en “un tipo de”. Por ejemplo, los objetos barco, tren y camión son un tipo de la clase Transporte.

Siguiendo con el ejemplo anterior de la clase Persona, instanciaremos un objeto:

```php
<?php

    class Persona {
        private $nombre;
        private $apellido;
        private $edad;

        public function __construct($n, $e){
            $this->nombre = $n;
            $this->edad = $e;
        }

        public function esMayor(){
            if($this->edad >= 18){
                return true;
            } else {
                return false;
            }
        }

        public function getNombre(){
            return $this->nombre;
        }

        public function setApellido($a){
            $this->apellido = $a;
        }
    }


    // Instanciamos un nuevo objeto de la clase Persona
    $usuario = new Persona("Juan", 24);

    // Establecemos su apellido (setter)
    $usuario->setApellido("Perez");

    // Obtenemos su nombre (getter)
    $nombre = $usuario->getNombre();

    // Ejecutamos uno de sus métodos
    if($usuario->esMayor() == true){
        echo "$nombre es mayor";
    } else {
        echo "$nombre es menor";
    }


?>
```

Características de la POO

Encapsulamiento

Se denomina encapsulamiento al ocultamiento del estado, es decir, de los atributos de un objeto de manera que sólo se puedan modificar mediante las operaciones definidas para ese objeto (getters y setters).

Cada objeto está aislado del exterior, y cada tipo de objeto expone una interfaz a otros objetos que especifica cómo pueden interactuar con los objetos de la clase. El aislamiento protege a las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ellas; solamente los propios métodos internos del objeto pueden acceder a su estado. Esto asegura que otros objetos no puedan cambiar el estado interno de un objeto de manera inesperada, eliminando efectos secundarios e interacciones inesperadas

De esta forma el usuario de la clase puede obviar la implementación de los métodos y propiedades para concentrarse sólo en cómo usarlos. Por otro lado se evita que el usuario pueda cambiar su estado de maneras imprevistas e incontroladas.

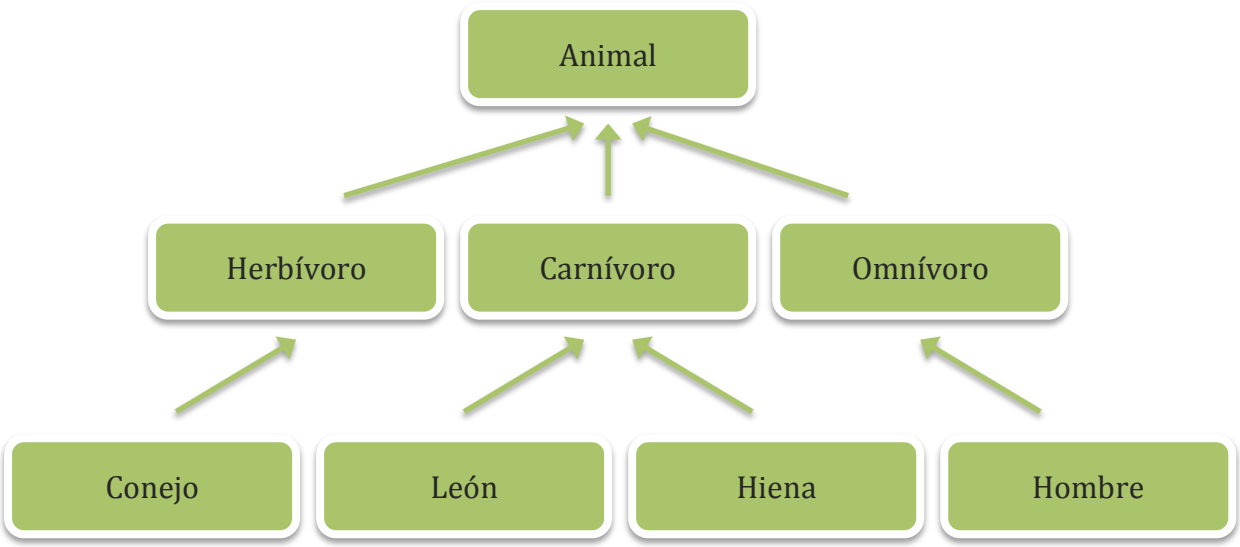
Herencia

Se da cuando una clase nueva se crea a partir de otra clase existente. La herencia proviene del hecho de que la subclase (la nueva clase creada) incluye los atributos y métodos de la clase primaria o padre, creando una jerarquía de clases.
La principal ventaja de la herencia es la capacidad para definir atributos y métodos nuevos para la subclase, que luego se aplican a los atributos y métodos heredados.

Continuando con el ejemplo anterior:

```php
<?php

    class Persona {
        private $nombre;
        private $apellido;
        private $edad;

        public function __construct($n, $e){
            $this->nombre = $n;
            $this->edad = $e;
        }

        public function esMayor(){
            if($this->edad >= 18){
                return true;
            } else {
                return false;
            }
        }

        public function getNombre(){
            return $this->nombre;
        }

        public function setApellido($a){
            $this->apellido = $a;
        }
    }

    class Hombre extends Persona {
        private $genero = "Masculino";

        public function getGenero(){
            return $this->genero;
        }
    }

    class Mujer extends Persona {
        private $genero = "Femenino";

        public function getGenero(){
            return $this->genero;
        }
    }



    $ella = new Mujer();
    $genero = $ella->getGenero();

    $el = new Hombre();
    $nombre = $el->getNombre();

?>
```

Para representar esto de una manera mas gráfica podemos ver el siguiente ejemplo:



De esto deducimos que el León es un tipo de Carnívoro, que a su vez es un tipo de Animal, o en POO, la clase León es una subclase de la clase Carnívoro, la cual es a su vez subclase de la clase Animal.

La herencia permite que existan clases que nunca serán instanciadas directamente. En el ejemplo anterior, una clase "Hiena" heredaría los atributos y métodos de la clase "Carnívoro", así como también "León" o cualquier otra subclase; pero, en ejecución, no habrá ningún objeto "Carnívoro" que no pertenezca a alguna de las subclases. En ese caso, a una clase así se la conocería como **Clase Abstracta**. La ausencia de instancias específicas es su única particularidad, para todo lo demás es como cualquier otra clase.

Polimorfismo

Comportamientos diferentes, asociados a objetos distintos, pueden compartir el mismo nombre; al llamarlos por ese nombre se utilizará el comportamiento correspondiente al objeto que se esté usando.

Siguiendo con el ejemplo anterior, podemos decir que todas las subclases de Animal, tienen un método conseguirComida(), pero cada una lo hará de una manera diferente.

Esto nos permite llamar al método conseguirComida() para cualquier subclase de Animal, sin preocuparnos por como lo hace. De esto se ocupa cada clase.

Ejercicios

1- Definir una variable que contenga su nombre e imprimir su contenido en pantalla.

2- Definir una variable que contenga su edad. Si el valor es mayor o igual que 25, imprimir en pantalla "Mayor de 25". De lo contrario, imprimir "Menor de 25". En ambos casos, agregar al final del mensaje "Mi edad es: " y el valor de la variable que contiene su edad.

3- Crear un array con los nombres de los días de la semana e imprimir el mensaje "Hoy es " mas el valor del array correspodiente al día de hoy.

4- Crear un bucle "for" que imprima los números del 1 al 10 separados por comas.

5- Crear un array con los siguientes colores: Rojo, Verde, Azul, Amarillo, Negro, Blanco. Utilizar el bucle "foreach" para imprimir todos los colores del array uno debajo del otro.

6- Crear una función llamada "calcularArea" que calculará el área de un triángulo. Para ello deberá recibir 2 parámetros (base y altura), efectuar el cálculo y devolver el resultado de la operación. Hacer una llamada a esta función, asignar su valor a una variable, e imprimir la variable por pantalla. Nota: La fórmula para calcular el área de un triángulo rectángulo es:

$$A = \frac{b \cdot h}{2}$$

7- Crear una clase llamada “Semaforo” que contenga:

- Un atributo “color” que defina si el semáforo está en rojo, amarillo o verde. Inicialmente estará vacío.
- Un atributo booleano “activo” que definirá si el semáforo estará funcionando o no. Inicialmente deberá ser verdadero.
- Un constructor que defina el color con que se creará el semáforo.
- Un método “cambiarColor” que se comporte de la siguiente manera:
    - Si color es rojo, cambiarlo a verde.
    - Si color es verde, cambiarlo a amarillo.
    - Si color es amarillo, cambiarlo a rojo.
- Getters y setters para cambiar directamente el color por uno específico. Sólo se debe permitir que los colores sean “rojo”, “amarillo” y “verde. De lo contrario, no cambiar el valor.
- Getters y setters para conocer y cambiar el estado de la variable “activo” del semáforo. Sólo se debe permitir que los valores sean “true” o “false”. De lo contrario, no cambiar el estado.

8- Instanciar un objeto de la clase Semaforo del ejercicio 7 asignando como color inicial “rojo”:

- Imprimir en pantalla el color actual.
- Pasar al siguiente color llamando al método “cambiarColor” del objeto.
- Imprimir en pantalla el color actual.
- Cambiar directamente el color a “amarillo” utilizando el setter correspondiente.
- Imprimir en pantalla el color actual.