Aprenda Programación Orientada a Objetos (POO) en PHP

Por: Stefan Mischook - Septiembre 07 2007

www.killerphp.com - www.killersites.com - www.idea22.com

Preámbulo:

La cosa mas difícil para aprender (y enseñar dicho sea de paso) en PHP orientado a objetos es... lo básico. Pero una vez que obtengas el conocimiento, el resto vendrá mucho, mucho mas fácil.

Pero no te desanimes! Tu has encontrado el tutorial mas fácil de entender sobre POO y PHP.

Esto podría sonar como un reclamo jactancioso... lo se. Pero eso es lo que los nerd del zeitgeist están diciendo.

... O al menos es lo que he escuchado.

Vídeos:

Como un bonus extra, he creado unos tutoriales de vídeo para ti. Ellos cubren el mismo material como el articulo escrito, y son diseñados para reforzar el articulo.

- Introducción a PHP Orientado a Objetos (4:05)
- Porque Aprender PHP Orientado a Objetos (14:46)
- Objetos y Clases en PHP (5:26)
- Construir Objetos en PHP – Parte 1 (9:14)
- Construir Objetos en PHP - Parte 2 (9:41)
- Construir Objetos en PHP - Parte 3 (6:18)

Si tienes preguntas/comentarios, puedes contactarme en: stefan@killersites.com

Gracias,

Stefan Mischook

Programación Orientada a Objetos en PHP

La Programación Orientada a Objetos (POO) es un tipo de programación agregada a PHP 5 que hace de la construcción compleja, modular y reusable de aplicaciones web mucho mas fácil

Con el lanzamiento de PHP 5, los programadores en PHP finalmente tienen el poder de programar como los grandes. Asi como Java y C#, PHP finalmente tiene una completa infraestructura POO.

En este tutorial, seras guiado (paso a paso) a través del proceso de construir y trabajar con objetos usando las capacidades de POO. Al mismo tiempo aprenderás:

• La diferencia entre construir una aplicación al estilo antiguo (por procedimiento) versus la manera POO.

• Cuales son los principios básicos de la POO y como se usan en PHP.

• Cuando debes usar POO en tus scripts PHP.

La gente entra en confusión cuando programa por falta del entendimiento de lo básico Con esto en mente, vamos a pasar lentamente sobre los tema básicos de la POO mientras creamos nuestros propios objetos en PHP.

Con este conocimiento, podrás ser capaz de explorar POO aun mas. Para este tutorial, debes entender un poco lo básico de PHP: funciones, variables, condicionales y bucles o repeticiones.

Para hacer las cosas mas fácil, el tutorial se encuentra dividido en 23 pasos:

Paso 1:

Primera cosa que necesitamos es crear dos paginas PHP:

index.php
class_lib.php

La POO trata acerca de crear código modular, de manera que nuestro código PHP orientado a objetos sera contenido en archivos dedicados que serán insertados en nuestra pagina PHP usando "includes" de PHP. En este caso todo nuestro código PHP OO estará en el archivo PHP:

class_lib.php

OOP se revuelve alrededor de una estructura construida llamada 'clase' (class). Las clases son los encargados de definir las plantillas que posteriormente son usadas para la definición de objetos.

Paso 2:

Crea una clase en PHP

En vez de tener un montón de funciones, variables y código flotando de manera espontánea, para diseñar sus scripts PHP o librerías de códigos a la manera de POO, necesitaras definir/crear tus propias clases.

Defines tus propias clases comenzando con la palabra clave "class" (clase) seguida del nombre que le quiere dar a su nueva clase.

```
<?php
class person {

}
?>
```

Paso 3:

Agrega datos a tu clase

Las clases son los planos de los objetos PHP- mas sobre esto luego. Una de las grandes diferencias entre las funciones y clases es que la clase contiene ambos datos (variables) y funciones que forman un paquete llamado un "objeto". Cuando tu creas una variable dentro de una clase, es llamada una "propiedad".

```
<?php
class person {
var name;
}
?>
```

Nota: Los datos/variables dentro de una clase (ej: var name) son llamados "propiedades".

Paso 4:

Agrega funciones y métodos a tus clases.

De la misma manera que las variables obtienen un nombre diferente cuando son creadas dentro de una clase (llamadas: propiedades) las funciones también referidas como (por los nerds) con un nombre diferente cuando son creadas dentro de una clase – son llamadas 'métodos'.

Los métodos de las clases son usados para manipular su propios datos/propiedades.

```
<?php
class person {
var $name;
function set_name($new_name) {
$this->name = $new_name;
}
function get_name() {
return $this->name;
}
}
?>
```

Nota: No olvides que en una clase, las variables son llamadas "propiedades".

Paso 5:

Funciones que obtienen y que establecen.

Hemos creado dos funciones/métodos interesantes: get_name() y set_name(). Estos métodos siguen una convención de POO que tu podrás ver en muchos lenguajes (incluyendo Java y Ruby) donde creas métodos que establecen y obtienen propiedades en una clase.

Otra convención (una convención para nombrar) es que los nombres para obtener (get_name) y establecer (set_name) deberían ser similares a los nombres de las propiedades (name).

```
<?php
class person {
var $name;
function set_name($new_name) {
$this->name = $new_name;
}
function get_name() {
return $this->name;
}
}
?>
```

Nota: Notese que los nombres que obtienen y establecen (set_name y get_name) hacen juego con el nombre de propiedad asociada (name). De esta manera, si otros programadores PHP quieren usar sus objetos, ellos sabrán que tienes un método/función llamado "set_name()" (establecer nombre), el cual se relaciona con la propiedad/variable llamada "name" (nombre).

Paso 6:

La variable '$this'

Ya probablemente has notado esta linea de código:

$this->name = $new_name.

El $**this** es una variable incluida (construida automáticamente dentro de todos los objetos) que apunta al objeto actual. O en otras palabras, $this es una variable especial de auto-referencia. Puedes usar $this para accesar a las propiedades y llamar a otros métodos de la clase actual donde te encuentras.

```
function get_name() {
return $this->name;
}
```

Nota: Esto podría ser un poco confuso para ti... esto es porque lo estas viendo por primera vez, es una de aquellas capacidades del OO (construidas dentro del mismo PHP 5) que hace las cosas automáticamente para nosotros. Por ahora, solo piense de $this como una palabra clave OO. Cuando PHP se encuentra con esta variable, el motor PHP sabe que hacer.
... Muy pronto tu también sabrás que hacer con ella.

Paso 7:

Incluye tu clase en la pagina principal de PHP.

Tu nunca deberías crear tus clases PHP directamente dentro de las paginas principales de PHP– esto eliminaría el propósito o esencia del uso de PHP orientado a objetos.

En vez de esto, la mejor practica es crear paginas de PHP separadas que solo contengan las clases. De esta forma accesarias a sus objetos y clases PHP incluyéndolas en tu pagina PHP principal con un "include" o "require".

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>OOP in PHP</title>
<?php include("class_lib.php"); ?>
</head>
<body>
</body>
</html>
```

Nota: Nota como no hemos hecho nada con nuestra clase aun. Lo haremos en el siguiente paso.

Paso 8:

Instancia/crea tu objeto

Las clases son planos/plantillas de objetos en PHP. Las clases no se hacen objetos hasta que realizas algo llamado: instanciamiento.

Cuando tu instancias una clase, creas un instancia de ella (valga la redundancia), creando así el objeto.

En otras palabras, el instanciamiento es el proceso de crear de una clase, una objeto en memoria. Que memoria? La memoria del servidor por supuesto.

```
<?php include("class_lib.php"); ?>
</head>
<body>
$stefan = new person();
</body>
</html>
```

Nota: La variable $stefan viene a ser la referencia/manejador de nuestro nuevo objeto de persona creado. Yo llamo a $stefan un "manejador", porque usaremos a $stefan para controlar y usar el objeto persona.

Si ejecutas el código PHP ahora, no podrás ver nada mostrado en la pagina. La razón para esto, es que no le hemos dicho a PHP que haga algo con el objeto que acabamos de crear.

Paso 9:

La palabra clave "new" (nuevo)

Para crear un objeto fuera de una clase debe usarse la palabra clave "new".

Cuando creas/instancias una clase, puedes opcionalmente agregar llaves al nombre de la clase, como yo hice en el ejemplo de abajo. Para estar claro, tu puedes ver en el código de abajo como yo pude crear múltiples objetos de la misma clase.

Desde el punto de vista del motor PHP, cada objeto es su propia entidad. Tiene sentido?

```
<?php include("class_lib.php"); ?>
</head>
<body>
$stefan = new person();
$jimmy = new person;
</body>
</html>
```

Nota: Cuando estés creando un objeto, asegúrate de no encerrar el nombre de la clase en comillas simples o dobles. Realizar esto ocasionaría un error ya que en vez de ser una clase seria una cadena de caracteres

Por ejemplo:

```
$stefan = new 'person';
```

... te dará un error.

Paso 10:

Estableciendo las propiedades del objeto.

Ahora que hemos creado/instanciado nuestros dos objetos separados "persona", podemos establecer sus propiedades usando los métodos (set_name y get_name) que nosotros hemos creado.

Tenga en mente que aunque ambos de nuestros objetos personas ($stefan y $nick) son basados en la misma clase "person", a lo que concierne a PHP, son objetos totalmente diferentes.

```
<?php include("class_lib.php"); ?>
</head>
<body>
<?php
$stefan = new person();
$jimmy = new person;
$stefan->set_name("Stefan Mischook");
$jimmy->set_name("Nick Waddles");
?>
</body>
</html>
```

Paso 11:

Accesando los datos del objeto.

Ahora nosotros usamos el método get_name para obtener acceso a los datos mantenidos en nuestros objetos... son los mismos datos que nosotros insertamos en nuestros objetos usando el método set_name.

Cuando accesamos a los métodos y propiedades de una clase, usamos el operador (->) llamado Operador Flecha o técnicamente Operador de Objetos.

```
<?php include("class_lib.php"); ?>
</head>
<body>
<?php
$stefan = new person();
$jimmy = new person;
$stefan->set_name("Stefan Mischook");
$jimmy->set_name("Nick Waddles");
echo "Stefan's full name: " . $stefan->get_name();
echo "Nick's full name: " . $jimmy->get_name();
?>
</body>
</html>
```

Nota: El operador flecha (->) no es el mismo operador usado con los arreglos asociativos: (=>).

Felicitaciones. Ya llevas la mitad del camino a través del tutorial. Tiempo para tomarte un receso y algo de te. OK de repente un poco de cerveza.

En breve, habrás:

• Diseñado una clase PHP.
• Generado/creado un par de objetos basado en tu clase.
• Insertado datos en tus objetos.
• Recuperado datos de tus objetos.

... No tan mal para tu primer día en PHP POO.

Si no lo has hecho aun, ahora es el gran momento para escribir código y verlo en acción en tus propias paginas PHP.

Paso 12:

Directamente accesando las propiedades - No lo hagas!

No tienes que usar métodos para accesar a las propiedades de los objetos; Puedes directamente llegar a ellos usando el operador flecha (->) y el nombre de la variable.

Por ejemplo: con la propiedad $name (en el objeto $stefan,) tu podrías obtener su valor de la siguiente manera:

$stefan -> name

Aunque se puede hacer, es considerado una mala practica hacerlo porque puede llevar a un problema en el camino. Deberías usar métodos como el get_name para obtener el valor – mas sobre esto luego.

```
<?php include("class_lib.php"); ?>
</head>
<body>
<?php
$stefan = new person();
$jimmy = new person;
$stefan->set_name("Stefan Mischook");
$jimmy->set_name("Nick Waddles");
// directly accessing properties in a class is a no-no.
echo "Stefan's full name: " . $stefan->name;
?>
</body>
</html>
```

Paso 13:

Constructores

Todos los objetos tienen un método incorporado llamado un "constructor". Los constructores le permiten inicializar las propiedades de sus objetos (traducción: dar valores a las propiedades) cuando instancias (creas) un objeto.

Nota: Si creas una función construct() (si es su opción), PHP automáticamente llamara el método/función constructor cuando hayas creado el objeto desde su clase.

El método "construct" comienza con dos subrayados (__) y la palabra 'construct'. El método constructor se "alimenta" proveyendo una lista de argumentos(como una función) después del nombre de clase.

```
<?php
class person {
var $name;
function __construct($persons_name) {
$this->name = $persons_name;
}
function set_name($new_name) {
$this->name = $new_name;
}
```

```
function get_name() {
return $this->name;
}
}
?>
```

Para el resto del tutorial, voy a detenerme recordándote que:

• Funciones = métodos
• Variables = propiedades

... Ya que esto es un tutorial Orientado a Objetos de PHP usare ahora la terminología OO (Orientado a Objetos).

Paso 14:

Crear un objeto con un constructor.

Ahora que hemos creado un método constructor, podemos proveer un valor para la propiedad $name cuando creemos nuestro objeto persona.

Por ejemplo:

```
$stefan = new person("Stefan Mischook");
```

Esto nos salva de tener que llamar el método set_name() reduciendo la cantidad de código

Los constructores son comunes y son usados a menudo en PHP, Java, etc.

```
<?php include("class_lib.php"); ?>
</head>
<body>
<?php
$stefan = new person("Stefan Mischook");
echo "Stefan's full name: " . $stefan->get_name();
?>
</body>
</html>
```

Este es un pequeño ejemplo de como los mecanismos incorporados dentro de PHP OO pueden ahorrar tiempo y reducir la cantidad de código que necesitas escribir. Menos código significa menos errores.

Paso 15:

Restringiendo el acceso a las propiedades usando "modificadores de acceso".

Uno de los fundamentos principales de la POO es la "encapsulacion". La idea es que tu puedas crear un código mas limpio y mejor, si tu restringes acceso a las estructuras de datos (propiedades) en sus objetos. Puedes restringir acceso a las propiedades de la clase usando algo llamado "modificadores de acceso". Existen 3 modificadores de acceso:

1. Publico
2. Protegido
3. Privado

Publico es el modificador por defecto. Significa que si no lo colocas se asume que es Publico.

```
<?php
class person {
var $name;
public $height;
protected $social_insurance;
private $pinn_number;
function __construct($persons_name) {
$this->name = $persons_name;
}
function set_name($new_name) {
$this->name = $new_name;
}
function get_name() {
return $this->name;
}
}
?>
```

Nota: Cuando declaras una propiedad con la palabra "var" se considera publica.

Paso 16:

Restringiendo acceso a las propiedades. Parte 2.

Cuando declaras una propiedad como "privada" solo la misma clase puede acceder a la propiedad.

Cuando una propiedad es declarada como "protegida" solo la misma clase y las clases derivadas de esta clase pueden acceder a la propiedad. Esto tiene que ver con la herencia... mas de esto luego.

Propiedades declaradas como "publicas" no tienen restricciones de acceso, significa que cualquiera las puede accesar.

Para ayudar a entender este concepto algo confuso de la POO, intenta el siguiente código y mira como reacciona PHP. Consejo: Lee los comentarios en el código para mayor información

```
<?php include("class_lib.php"); ?>
</head>
<body>
<?php
$stefan = new person("Stefan Mischook");
echo "Stefan's full name: " . $stefan->get_name();
/*
Since $pinn_number was declared private, this line of code will
generate an error. Try it out!
*/
echo "Tell me private stuff: " . $stefan->$pinn_number;
?>
</body>
</html>
```

Paso 17:

Restringiendo el acceso a los métodos

Asi como las propiedades, también puedes controlar el acceso a los métodos usando uno de los tres modificadores de acceso:

1. Publico
2. Protegido
3. Privado

Porque tenemos modificadores de acceso ?

La razón para modificadores de acceso se puede resumir basicamente en el control: Tiene sentido el hecho de poder controlar como la gente usa las clases.

Las razones para los modificadores de acceso y otras construcciones OO, pueden llegar a ser difíciles de entender... ya que somos principiantes aquí Asi que date un chance.

Con esto dicho, pienso que podemos resumir y decir que muchas construcciones POO existen, con la idea de que muchos programadores puedan estar participando en un proyecto.

```
<?php
class person {
var $name;
public $height;
protected $social_insurance;
private $pinn_number;
function __construct($persons_name) {
$this->name = $persons_name;
}
private function get
return $this->$pinn_number;
}
?>
```

Nota: Ya que el método get_pinn_number() es 'privado', el único lugar donde puedes utilizar este método es en la misma clase – típicamente a través de otro método Si quisieras llamar/usar este método directamente en tus paginas PHP, tendrías que declararlo "publico".

Nota de Nerd: Nuevamente, es importante (mientras continuamos con el aprendizaje) que utilices el código tu mismo. Esto hará una diferencia muy grande en el aprendizaje.

Paso 18:

Reutilizando el código a la manera POO: herencia

Herencia es la capacidad fundamental/construcción en POO donde puedes utilizar una clase, como la base para otra clase... u otras clases.

Porque hacerlo?

Haciendo esto te permite de forma eficientemente reutilizar el código encontrado en la clase base. Digamos, que quieres crear una nueva clase "employee" (empleado)... ya que podemos decir que "employee" (empleado) es un tipo de "person" (clase person menciona anteriormente), ellos compartirían propiedades comunes y métodos

... Tiene algo de sentido?

En este tipo de situación, la herencia puede hacer tu código mas ligero... porque estas reutilizando el mismo código en dos clases distintas. Pero a diferencia de la forma vieja de programación en PHP:

1. Tu solo tienes que tipear el código una vez.

2. El código actual siendo reutilizado, puede ser reutilizado en muchas (ilimitadas) clases pero es solamente tipeado en un solo lugar... conceptualmente, este es un tipo de include() de PHP.

Mire el siguiente código de ejemplo:

```
// 'extends' es la palabra clave que extiende la herencia
class employee extends person {
function __construct($employee_name) {
}
}
```

Paso 19:

Reutilizando código con herencia: parte 2

Ya que la clase "employee" (empleado) esta basada en la clase "person" (persona), "employee" automáticamente obtiene todas las propiedades publicas y protegidas de "person" así como los métodos de "person".

Nota de Nerd: Nerds deberían decir que 'employee' es un tipo de person (persona).

El código:

```
// 'extends' es la palabra clave que permite la herencia
class employee extends person {
function __construct($employee_name) {
$this->set_name($employee_name);
}
}
```

Has notado como somos capaces de usar set_name() en "employee", aunque no hemos declarado ese método en la clase "employee". Esto es porque ya hemos creado set_name() en la clase "person".

Nota de Nerd: La clase 'person' es llamada (por los nerds) la clase "base" o la clase "padre" porque es la clase en la cual "employee" esta basada. Esta jerarquía de clase puede llegar a ser importante a lo largo del camino cuando tus proyectos sean mas complejos.

Paso 20:

Reutilizando código con herencia: parte 3

Como puedes ver en el fragmento de código abajo, podemos llamar a get_name en nuestro objeto, cortesía de "person".

Este es un clásico ejemplo de como la POO puede reducir el numero de lineas de código (no tiene que escribir el mismo método dos o mas veces) mientras manteniendo su código modular y mucho mas fácil de mantener.

```
<title>OOP in PHP</title>
<?php include("class_lib.php"); ?>
</head>
<body>
<?php
// Using our PHP objects in our PHP pages.
$stefan = new person("Stefan Mischook");
echo "Stefan's full name: " . $stefan->get_name();
$james = new employee("Johnny Fingers");
echo "---> " . $james->get_name();
?>
</body>
</html>
```

Paso 21:

Sobrescribiendo métodos

A veces (cuando se usa herencia) puedes necesitar cambiar como un método funciona desde la clase base. Por ejemplo, digamos el método set_name() en la clase "employee" tiene que hacer algo diferente a lo que hace en la clase "person". Tu "sobrescribes" la versión de la clase "person" de set_name() declarándola de nuevo en "employee".

El código:

```
<?php
class person {
// explícitamente agregando propiedades de la clase
// son opcionales pero una Buena practica
var $name;
function __construct($persons_name) {
$this->name = $persons_name;
}
public function get_name() {
return $this->name;
}
// métodos protegidos y propiedades restringen el acceso a aquellos
// elementos
protected function set_name($new_name) {
if (name != "Jimmy Two Guns") {
$this->name = strtoupper($new_name);
}
}
}
// 'extends' es la palabra clave que permite la herencia
class employee extends person {
protected function set_name($new_name) {
if ($new_name == "Stefan Sucks") {
$this->name = $new_name;
}
}
function __construct($employee_name) {
$this->set_name($employee_name);
}
}
?>
```

Notese como set_name() es diferente en la clase "employee" en comparación con la versión encontrada en la clase padre: "person".

Paso 22:

Sobrescribiendo métodos: parte 2

Algunas veces necesitaras accesar a la versión de tu clase base de un método que sobrescribistes en la clase derivada (a veces llamada "clase hija").

En nuestro ejemplo. Nosotros sobrescribimos el método set_name() en la clase "employee".

Ahora he usado este código:

person::set_name($new_name);

... para accesar la clase padre (person) del método set_name().

El código:

```
<?php
class person {
// agregando explícitamente propiedades de clase son opcionales
// pero es una Buena practica
var $name;
function __construct($persons_name) {
$this->name = $persons_name;
}
public function get_name() {
return $this->name;
}
// métodos protegidos y propiedades restringen el acceso a
// aquellos elementos
protected function set_name($new_name) {
if (name != "Jimmy Two Guns") {
$this->name = strtoupper($new_name);
}
}
}
// 'extends' es la palabra clave que permite la herencia
class employee extends person {
```

```
protected function set_name($new_name) {
if ($new_name == "Stefan Sucks") {
$this->name = $new_name;
}
else if($new_name == "Johnny Fingers") {
person::set_name($new_name);
}
}
function __construct($employee_name) {
$this->set_name($employee_name);
}
}
?>
```

Paso 23:

Sobrescribiendo métodos: parte 3

Usando :: permite específicamente nombrar la clase donde quieres que PHP busque un método – "person::set_name()" dice a PHP que busque set_name() en la clase "person".

También existe un atajo si solo quieres referir a la clase padre actual: Usando la palabra reservada "parent".

El código:

```
<?php
class person {
// explícitamente agregar propiedades de las clases es opcional
// pero buena practica
var $name;
function __construct($persons_name) {
$this->name = $persons_name;
}
public function get_name() {
return $this->name;
}
//métodos protegidos restringen el acceso a aquellos elementos
```

```
protected function set_name($new_name) {
if (name != "Jimmy Two Guns") {
$this->name = strtoupper($new_name);
}
}
}
// 'extends' es la palabra clave que habilita la herencia
class employee extends person {
protected function set_name($new_name) {
if ($new_name == "Stefan Sucks") {
$this->name = $new_name;
}
else if($new_name == "Johnny Fingers") {
parent::set_name($new_name);
}
}
function __construct($employee_name) {
$this->set_name($employee_name);
}
}
?>
```

Comentarios Finales

Solo hemos tocado lo básico de PHP OO. Pero deberías tener suficiente información para sentirse cómodo yendo hacia adelante.

Recuerda que la mejor manera de tener esto "bien grabado" es escribiendo el código por diversión y practicando.

Yo sugeriría crear 10 objetos simples que hagan cosas simples, y entonces usar estos objetos en paginas PHP actuales. Una vez que hayas hecho eso, te sentirás muy cómodo con los objetos.

Porque aprender POO en PHP. Otra punto de vista.

Para gente nueva a POO y que están mas "cómodos" con el PHP por procedimiento, podrán estar preguntándose porque deberían molestarse en aprender conceptos de objetos ...

Porque tomarse la molestia de aprender?

El mundo de PHP:

El PHP se esta moviendo en la direccion de POO. Por ejemplo, mucha extensiones importantes del PHP como PEAR y Smarty son basadas en OO. Asi que realmente entender y usar estos frameworks propiamente, necesitaras entender PHP orientado a objetos.

Las ventajas practicas/funcionales:

Para pequeños proyectos, usar PHP orientado a objetos podría ser demasiado. Esto dicho, el PHP orientado a objetos realmente comienza a brillar cuando los proyectos se hacen mas complejos, y cuando tu tienes mas de una persona haciendo el programa.

Por ejemplo:

Si encuentras que tienes, digamos 10, 20 o mas funciones y encuentras que algunas de las funciones están haciendo cosas similares... es tiempo de considerar empaquetar las cosas en objetos usando POO.

POO y tu carrera como programador:

POO es la manera moderna del desarrollo de software y todos los lenguajes mayores (Java, PERL, PHP, C#, Ruby) usan este método de programación Como desarrollador/programador de software, solo tiene sentido (en términos de carrera,) mantener tus habilidades actualizadas.

Mas allá de hacerte un programador de PHP de mayor valor, entendiendo POO en PHP te dará conocimientos (conocimientos de POO) que tu seras capaz de llevar contigo a otros lenguajes.

... Cuando aprendas POO en PHP, aprenderás programación orientada a objetos para cualquier lenguajes basado en OO.

Encontraras con tiempo que crear proyectos PHP basados en objetos, hará tu vida de programador mas fácil Como un bonus adicional, pronto desarrollaras tu propia colección de objetos reutilizables, que te servirán para otros proyectos.

Finalmente, también encontraras que el PHP basado en POO es mas fácil de mantener y actualizar.

Retos en POO:

PHP OO presenta algunos retos en cuanto comienzas porque necesitas aprender a pensar acerca de tus proyectos PHP de una manera diferente: necesitaras conceptualizar el proyecto en términos de objetos.

Mas detalles...

Una vía común de comenzar un proyecto orientado a objetos es comenzar dibujando un simple diagrama de tus objetos. Mientras empiezas a trabajar con diagramas orientados a objetos, encontraras que ellos te ayudaran en hacer el desarrollo de proyectos PHP basados en POO muchos mas fáciles

Aquí tienes unos cuantos consejos acerca de diagramas de dibujos de objeto.

• Use un papel y un lápiz
• Dibuja cajas que representen cada objeto.
• En aquellas cajas, crea una lista de sus métodos y propiedades.
• Usa flechas y lineas entre cajas para denotar relaciones (padre – hijo) entre objetos.

Asi que si has estado esperando para saltar en el mundo de PHP OO, ahora es el mejor momento para hacerlo.

Stefan Mischook

www.killerphp.com
www.killersites.com