Aprenda Programacion Orientada a Objetos en (POO) en PHP

Por: Stefan Mischook - Septiembre 07 2007

www.killerphp.com - www.killersites.com

Preambulo:

La cosa mas dificil para aprendar (y ensenar dicho sea de paso) en PHP orientado a objetos es lo basico. Peor una vez que esten bajo tu dominio, el resto vendra mucho, mucho mas facil. Pero no se desanime! Usted ha encontrado el mas facil tutorial de entender sobre POO y PHP. Esto podria sonar como un reclamo jactancioso… lo se. Pero es lo que el nerd zeitgeist esta diciendo.

… O al menos se me ha dicho.

Videos:
Como un bono extra, he creado unos tutorials de video para usted. Ellos cubren el mismo material como el articulo escrito, y son disenados para reforzar el articulo.

- Introduccion to 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 usted tiene preguntas/comentarios, usted puede contactarme en: **stefan@killersites.com**

Gracias,

Stefan Mischook

Programacion Orientada a Objetos en PHP
Programacion Orientada a Objetos (POO) es un tipo de programacion agregada a php5 que hace la construccion compleja, modular, y aplicaciones web reusables y mucho mas faciles.
Con el lanzamiento del php5, los programadores php finalmente tienen el poder de codificar con los “grandes chicos” como Java y C#, php finalmente tiene una complete infraestructura POO.

En este tutorial, usted sera guiado (paso-a-paso) a traves del proceso de construir y trabajar con objetos usando las capacidades de POO. Al mismo tiempo usted aprendera:

- La diferencia entre construir una aplicacion y la del estilo antiguo (por procedimiento) versus la manera POO.
- Cuales son los principios basicos de la POO y como se usan en PHP.

- Cuando usted deberia querer usar POO en sus scripts PHP.

La gente entra en confusion cuando programa por causa de la falta de entendimiento de lo basico. Con esto en mente, estamos yendo lentamente sobre el tema de los principios de la POO mientras creamos nuestros propios objetos PHP. Con este conocimiento, usted sera capaz de explorer POO adicional. Para este tutorial, usted deberia entender una poco de lo basico de PHP: funciones, variables, condicionales y loops (repeticiones).

Para hacer la cosa facil, el tutorial esta 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 codigo modular, de manera que nuestro codigo PHP orientado a objetos sera contenido en archivos dedicados que entonces se insertaran en nuestra pagina normal PHP usando “includes” de PHP. En este caso todo nuestro codigo PHP OO estara en el archivo PHP:

class_lib.php

OOP se refiere a una construccion llamada 'clase' (class). Las clases son los corta cookies / platillas que son usados para definir objetos.

Paso 2:

Crear una clase PHP

En vez de tener un monton de funciones, variables y codigo flotando de manera espontanea, para disenar sus scripts php o librerias de codigos a la manera de POO, usted necesitara definir/crear sus propias clases. Usted define sus 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:

Agregue datos a su clase

Las clases son los planos de los objetos php- mas sobre eso 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 usted crea 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:

Agregue funciones y metodos a sus clases.
En la misma manera que las variables obtienen un nombre diferente cuando son creadas dentro de una clase (ellas son llamadas: propiedades) las funciones tambien referidas como (por los nerds) por un nombre diferente cuando son creadas dentro de una clase – son llamadas ‘metodos’.

Los metodos 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 olvide que en una clase, las variables son llamadas “propiedades”.

Paso 5

Funciones que obtienen y que establecen.
Hemos creado dos funciones/metodos interesantes: get_name() y set_name()

Estos metodos siguen una convencion de POO que usted puede ver en muchos lenguajes (incluyendo Java y Ruby) donde usted crea metodos que establecen y obtienen propiedades en una clase. Otra convencion (una convencion para nombrar) es que los nombres para obtener y establecer deberian emparejar con los nombres de las propiedades.

```
<?php

class person {
```

```
	var $name;
	function set_name($new_name) {
		$this->name = $new_name;
	}

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

?>
```

Nota: Note que los nombres que obtienen y establecen hacen juego con el nombre de propiedad asociado. De esta manera, otros programadores PHP van a querer usar sus objetos, ellos sabran que si usted tiene un metodo/funcion llamado “set_name()” (establecer nombre), habra una propiedad/variable llamada “nombre”.

Paso 6:

La variable '$this'

Usted probablemente ha notado esta linea de codigo:

$this->name = $new_name.

El $this is una variable incluida (construida dentro de todos los objetos) que apunta al objeto actual. O en otras palabras, $this es una variable especial de auto-referencia. Usted usa $this para accesar a las propiedades y llamar a otros metodos de la clase actual.

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

Nota: Esto podria ser un poquito confuso para alguno de ustedes… esto porque usted lo esta viendo por primera vez, una de aquellas capacidades OO (construidas dentro del mismo PHP5) y que hace las cosas automaticamente por 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 For now, just think of $this as a special OO PHP keyword. When PHP comes across $this, the PHP engine knows what to do.

… Esperamos que pronto, usted tambien lo hara!

Paso 7:

Incluya su clase en su pagina principal PHP.
Usted nunca deberia crear sus clases PHP directamente dentro de sus paginas PHP principales – esto ayudaria a quitar los propositos del PHP orientado a objetos en el primer lugar.

En vez de eso, es siempre la mejor practica crear paginas php separadas que solo contengan sus clases. Entonces usted accesaria sus objetos y clases php incluyendolos en su 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: Note como no hemos hecho nada con nuestra clase aun. Lo haremos en la siguiente.

Paso 8:

Instancie/cree su objeto

Las clases son planos/plantillas de objetos php. Las clases no se hacen objetos actualmente hasta que usted hace algo llamado: instanciamiento

Cuando usted instancia una clase, usted crea una instancia de ella, entonces creando el objeto. En otras palabras, el instanciamiento es el proceso de crear una instancia de un objeto en memoria. Que memoria? La memoria del servidor porsupuesto.

```
<?php include("class_lib.php"); ?>

</head>

<body>
      $stefan = new person();

</body>

</html>
```

Nota: Nota: La variable $stefan viene a ser la referencia(handle) de nuestro nuevo objeto de persona creado. Yo llamo a $stefan un “handle”, porque usaremos $stefan para controlar y usar el objeto persona. Si usted ejecuta el codigo PHP ahora, usted no vera nada mostrado en sus paginas. La razon para esto, es porque no le hemos dicho a PHP que haga algo con el objeto que justo hemos creado.

Paso 9:

La “nueva (new)” palabra clave. Para crear un objeto fuera de una clase debe usar la palabra clave “new”.

Cuando cree/instancia una clase, usted puede opcionalmente agregar llaves al nombre de la clase, como yo hice en el ejemplo abajo. Para ser claro, usted puede ver en el codigo abajo como yo puedo crear multiples objetos de la misma clase.

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

```
<?php include("class_lib.php"); ?>

</head>

<body>

      $stefan = new person();

      $jimmy = new person;

</body>

</html>
```

Nota: Cuando este creando un objeto, asegurese de no mencionar el nombre de la clase.

Por ejemplo:

$stefan = new 'person';

… te dara un error.

Paso 10:

Estableciendo las propiedades del objeto. Ahora que hemos creado/instanciado nuestros dos objetos separados “persona”, podemos establecer sus propiedad usando los metodos (setters) que nosotros hemos creado.

Tenga en mente que aunque ambos de nuestros objetos personas ($stefan and $nick) son basados en la misma clase “person”, pero en 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 los metodos para obtener para accesar los datos mantenidos en nuestros objetos… son los mismos datos que nosotros insertamos en nuestros objetos usando los metodos para establecer. Cuando accesamos los metodos y propiedades de una clase, usamos el operador flecha (->).

```
<?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. Usted ha hecho la mitad del camino a traves del tutorial. Tiempo para tomar un receso y algo de te. OK derepente un poco de cerveza.

En breve, usted habra:

- Disenado una clase PHP.
- Generado/creado un par de objetos basado en su clase.
- Insertado datos en sus objetos.
- Recuperado datos de sus objetos.

… No tan malo para su primer dia en el trabajo OO de PHP.

Si no lo has hecho aun, ahora es el gran momento para escribir codigo y verlo en accion en sus propias paginas PHP.

Paso 12:

Directamente accesando las propiedades. No lo haga!

Usted no tiene que usar metodos para accesar las propiedades de los objetos; usted puede directamente llegar a ellos usando el operador flecha (→) y el nombre de la variable.

Por ejemplo: con la propiedad $name (en el objeto $stefan,) usted podria obtener su valor como:

$stefan→name

Aunque se puede hacer, es considerado una mala practica hacerlo porque puede llevar a un problema en el camino. Usted deberia usar metodos para obtener – 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 metodo incorporado llamado un “constructor”. Los constructores le permiten inicializar las propiedades de sus objetos (traduccion: dar los valores de las propiedades) cuando usted instancia (crea) un objeto.

Nota: Si usted crea una funcion construct() (si es su opcion), PHP automaticamente llamara el metodo/funcion constructor cuando usted crea un objeto desde su clase.
El metodo “construct” comienza con dos subrayados (__) y la palabra 'construct'. Uste “alimenta” el metodo constructor proveyendo una lista de argumentos(como una funcion) despues 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 recordandote que:

- Functiones = metodos
- Variables = propiedades

… Desde que es un tutorial Orientado a Objetos de PHP ahora usare la terminologia OO.

Paso 14:

Crear un objeto con un constructor. Ahora que hemos creado un metodo constructor, podemos proveer un valor para la propiedad $name cuando creamos nuestro objeto persona.

Por ejemplo:

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

Esto no salva de teer que llamar el metodo set_name() reduciendo la cantidad de codigo. Constructores son communes 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 pequeno ejemplo de como los mecanismos incorporados dentro del PHP OO pueden ahorrarte tiempo y reducir la cantidad de codigo que necesitas escribir. Menos codigo significa menos errores.

Paso 15:

Restringiendo el acceso a las propiedades usando “modificadores de acceso”. Uno de los fundamentos principales en la POO es la “encapsulacion”. La idea es que tu puedas crear un codigo limpio y mejor, si tu restrings acceso a las estructuras de datos (propiedades) en sus objetos. Usted restringe acceso a las propiedades de la clase usando algo llamado “modificadores de acceso”. Hay 3 modificadores de acceso.

1. publico
2. privado
3. protegido

Publico es el modificador por defecto.

```
<?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;
		}
}

?>
```

Paso 16:

Restringiendo el acceso a las propiedades. Parte 2.
Cuando tu declaras una propiedad como “privada” solo la misma clase puede accesar la propiedad. Cuando una propiedad es declarada “protegida” solo la misma clase y las clases derivadas de esa clase puede accesar la propiedad. Esto tiene que ver con la herencia… mas de eso luego.

Propiedades declaradas como “publicas” no tienen restricciones de acceso, significando que cualquiera las puede accesar. Para ayudar a entender esto (probablemente) el aspecto nuboso de la POO, intenta el siguiente codigo y mira como reacciona PHP. Consejo: Lee los comentarios en el codigo para mayor informacion.

```
<?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 metodos.
Como las propiedades, usted puede controlar el acceso a los metodos usando uno de los tres modificadores de acceso:

1. publico
2. protegido
3. privado

Porque tenemos modificadores de acceso ?

La razon para modificadores de acceso viene al control: este da sentido a controlar como la gente usa las clases.

Las razones para accesar los modificadores y otras construcciones OO, pueden llegar a ser dificiles de entender… desde que somos principiantes aqui. Asi que usted mismo dese una oportunidad. Se dijo. Dicho eso, pienso que podemos resumir y decir que muchas construcciones POO existen, con idea de que muchos programadores puedan estar participando de 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;

}
?>
```

Notas: Desde que el metodo get_pinn_number() es 'privado', el unico lugar donde usted puede usar este metodo es en la misma clase – tipicamente en otro metodo. Si usted queria llamar/usar este metodo directamente en sus paginas PHP, ustede deberia declararlo “publico”.

Nota de Nerd: Nuevamente, es importante (mientras estamos de acuerdo) que actualmente usted mismo pruebe el codigo. Esto hace una inmensa diferencia.

Paso 18:

Reutilizando el codigo a la manera POO: herencia

Herencia es la capacidad fundamental/construccuion en POO donde usted puede usar una clase, como la base para otra clase… o muchas otras clases.

Porque hacerlo?

Haciendo esto permite que usted eficientemente reuse el codigo encontrado en su clase base.Digamos, usted queria crear una nueva clase “employee” (empleado) desde que podemos decir que “employee” (empleado) es un tipo de “persona”, ellos compartiran propiedades comunes y metodos.

… Haciendo algo de sentido?

En este tipo de situacion, la herencia puede hacer tu codigo mas ligero… porque estas reutilizando el mismo codigo en dos diferentes clases. Sino que a diferencia de la “antigua-escuela” PHP:

1. Usted solo tiene que tipear el codigo una vez.

2. El codigo 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 al siguiente codigo de ejemplo:

```
// 'extends' es la palabra clave que extiende la herencia

class employee extends person {

      function __construct($employee_name) {

      }
}
```

Paso 19:

Reutilizando codigo con herencia: parte 2

Porque la clase “employee” (empleado) esta basada en la clase “person” (persona), “employee” automaticamente tiene todas las propiedades publicas y protegidas y metodos de “person”.

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

El codigo:

```
// 'extends' es la palabra clave que permite la herencia

class employee extends person {

      function __construct($employee_name) {
            $this->set_name($employee_name);
      }
}
```

Note como somos capaces de usar set_name() en “employee”, aunque no hemos declarado ese metodo en la clase “employee”. Esto es porque ya hemos creado set_name() en la clase “person”.

Nota de Nerd: La clase 'person' class es llamada (por nerds) la clase “base” o la clase “padre” porque es la clase que “employee” esta **basada**. Esta jerarquia de clase puede llegar a ser importante a lo largo del camino cuando sus proyectos se hagan mas complejos.

Paso 20:

Reutilizando codigo con herencia: parte 3

Mientras usted pueda ver en el fragmento de codigo abajo, podemos llamar a get_name en nuestro objeto cortesia de “person”.

Este es un clasico ejemplo de como la POO puede reducir el numero de lineas de codigo (no tiene que escribir el mismo metodo dos veces) mientras aun esta guardando su codigo modular y mucho mas facil 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

Sobreescribiendo metodos
A veces (cuando se usa herencia) usted podria necesitar de cambiar como un metodo funciona desde la clase base. Por ejemplo, digamos el metodo set_name() en la clase “employee” tenia que hacer algo diferente que lo que hace en la clase “person”.Usted “sobreescribe” la version de las clases “person” de set_name() declarando el mismo metodo en “employee”.

El codigo:

```
<?php

class person {
      // explicitamente 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;
      }

      // metodos 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);
      }
}

?>
```

Note como set_name() es diferente en la clase “employee” desde la version encontrada en la clase padre: “person”.

Paso 22:

Sobre escribiendo metodos: parte 2
Algunas veces usted podria necesitar accesar a la version de su clase base de un meotdo que usted sobreescribio en la clase derivada (a veces llamada “clase hija”).

En nuestro ejemplo. Nosotros sobreescribimos el metodo set_name() en la clase “employee”. Ahora he usado este codigo:

person::set_name($new_name);

… para accesar la clase padre (person) del metodo set_name().

El codigo:

```
<?php

class person {

      // agregando explicitamente 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;
      }

      // metodos 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:

Sobreescribiendo metodos: part 3

Usando :: permie especificamente nombrar la clase donde usted quiere que PHP busque un metodo – “person::set_name()” dice a PHP que busque por set_name() en la clase “person”. Hay tambien un atajo si usted solo se quiere referir a la clase padre actual: por usar la palabra clave “parent”.

The code:

```
<?php

      class person {
      // explicitamente 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;
	}

	//metodos 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 basico de PHP OO. Pero usted deberia tener suficiente informacion para sentirse comodo yendo hacia adelante.

Recuerde que la mejor manera de tener esto “empapado” es actualmente escribiendo el codigo por diversion.

Yo sugeriria crear 10 objetos simples que hagan cosas simples, y entonces usar aquellos objetos en paginas PHP actuales. Una vez que lo has hecho eso, usted se sentira muy comodo con los objetos.

Porque aprender POO en PHP. Otra toma o punto de vista.

Para gente nueva a POO y que estan mas “comodos” con el php procedural , usted podria estar preguntandose porque ustede deberia aun molestar en aprender conceptos de objetos … porque ir a traves del problema?

El mundo PHP:

El PHP se esta moviendo en la direction de POO. Por ejemplo, mucha extensions importantes del PHP como PEAR y Smarty son basadas en OO. Asi que realmente entender y usar estas armazones propiamente, usted necesitara entender PHP orientado a objetos.

Las ventajs practicas/funcionales:

Para pequenos proyectos, usar PHP orientado a objetos podria ser destructor. Se dijo, que 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 tu encuentras que tu has dicho 10-20 mas funciones y tu encuentras que algunas de las funciones estan haciendo cosas similares… es tiempo de considerar empaquetar las cosas en objetos usando POO.

POO y su carrera como programador:

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

Mas alla de hacerte un codificador PHP de mayor valor, entendiendo POO en PHP te dara conocimiento de (conocimiento POO) que tu seras capaz de tomar contigo en otros lenguajes.
… Cuando tu aprendes POO en PHP, usted aprenderan programacion orientada a objetos por cualquier lenguajes basado en OO.

Usted encontrara con tiempo que crear projectos PHP basados en objetos, hara tu vida de programador mas facilmente. Como bono adicional, usted pronto desarrollarasu propia coleccion de objetos reutilizables, que te servira de palanca en otros proyectos. Finalmente, usted tambien encontrara que el PHP basado en POO es mas facil de mantener y actualizar.

Retos POO:

PHP OO presenta algunos retos cuando usted comienza porque usted necesitara aprender a pensar acerca de sus proyectos PHP en una manera diferente: usted necesitara conceptualizar el proyecto en terminos de objetos.

Mas detalles…

Una via comun de comenzar un proyecto orientado a objetos es comenzar dibujando un simple diagrama de sus objetos. Mientras usted empieza a trabajar con diagramas orientados a objetos, usted encontrara que ellos ayudan a hacer el desarrollo de proyectos PHP basados en POO muchos mas faciles.

Aqui hay unos cuantos consejos acerca de diagramas de dibujos de objeto.

- Use un papel y un lapicero
- Dibuje cajas que representen cada objeto.
- En aquella cajas,liste sus metodos y propiedades
- Use flechas y lineas entre cajas para denotar relaciones (padre – hijo) entre objetos.

Asi que si usted se ha estado sentando en la baranda esperando saltar en PHP OO, ahora es buen tiempo para que usted comience.

Stefan Mischook

www.killerphp.com
www.killersites.com