**PHP orientado a objetos con clases y objetos**

En este artículo vamos a explorar los fundamentos de la programación orientada a objetos en PHP. Vamos a comenzar con una introducción a las clases y los objetos, y discutiremos un par de conceptos avanzados como la herencia y el polimorfismo en la segunda mitad de este artículo.

**¿Qué es la programación orientada a objetos (POO)?**

La programación orientada a objetos, comúnmente conocida como POO (o bien OOP, por sus siglas en inglés), es una aproximación que te ayuda a desarrollar aplicaciones complejas de una manera fácil de mantener y escalar a largo plazo. En el mundo de la POO, las entidades del mundo real como Person (Persona), Car (Automóvil) o Animal son tratadas como objetos. En la programación orientada a objetos interactúas con tu aplicación a través del uso de objetos. Esto contrasta con la programación por procedimientos, en la que interactúas principalmente con funciones y variables globales.

En la POO existe el concepto de **"clase" ("class" en inglés)**, que se usa para modelar o asignar una entidad del mundo real a una plantilla de datos (**propiedades, o properties**) y funcionalidad (**métodos, o methods**). Un **"objeto" ("object" en inglés)** es una instancia de una clase, y puedes crear múltiples instancias de la misma clase. Por ejemplo, existe una sola clase Person (Persona), pero muchos objetos de tipo persona pueden ser instancias de esta clase —dan, zainab, hector, etc.

La clase define propiedades. Por ejemplo, para la clase Person podemos tener name (nombre), age (edad) y phoneNumber (número telefónico). Entonces cada objeto de tipo persona tendrá sus propios valores para esas propiedades.

También puedes definir métodos en la clase, que te permiten manipular los valores de las propiedades de los objetos y llevar a cabo operaciones en ellos. Como ejemplo, puedes definir un método save (guardar) que almacene la información del objeto en una base de datos.

**¿Qué es una clase de PHP?**

Una clase es una plantilla que representa a una entidad del mundo real, y define propiedades y métodos para la misma. En esta sección discutiremos la anatomía básica de una típica clase en PHP.

La mejor manera de comprender conceptos nuevos es a través de un ejemplo. Así que echemos un vistazo a la clase Employee (Empleado) en el siguiente fragmento de código, que representa a la entidad del empleado.

|  |  |
| --- | --- |
| 1 | <?php |
| 2 | class Employee |
| 3 | { |
| 4 | private $first\_name; |
| 5 | private $last\_name; |
| 6 | private $age; |
| 7 |  |
| 8 | public function \_\_construct($first\_name, $last\_name, $age) |
| 9 | { |
| 10 | $this->first\_name = $first\_name; |
| 11 | $this->last\_name = $last\_name; |
| 12 | $this->age = $age; |
| 13 | } |
| 14 |  |
| 15 | public function getFirstName() |
| 16 | { |
| 17 | return $this->first\_name; |
| 18 | } |
| 19 |  |
| 20 | public function getLastName() |
| 21 | { |
| 22 | return $this->last\_name; |
| 23 | } |
| 24 |  |
| 25 | public function getAge() |
| 26 | { |
| 27 | return $this->age; |
| 28 | } |
| 29 | } |
| 30 | ?> |

La instrucción class Employee en la primera línea define a la clase Employee. A continuación declaramos las propiedades, el constructor y los otros métodos de la clase.

**Propiedades de clase en PHP**

Puedes imaginar a las propiedades de la clase como variables que se usan para almacenar información acerca del objeto. En el ejemplo anterior, hemos definido tres propiedades *—*first\_name, last\_name y age (nombre, apellido y edad). En la mayoría de los casos, se accede a las propiedades de las clases a través de objetos instanciados.

Estas propiedades son de tipo private (privado), lo que significa que solamente puede accederse a ellas desde el interior de la clase. Este es el nivel de acceso más seguro para las propiedades. Vamos a discutir los diferentes niveles de acceso para las propiedades y métodos de una clase posteriormente en este artículo.

**Constructores para clases de PHP**

Un constructor es un método de clase especial que se ejecuta automáticamente al instanciar un objeto. Veremos cómo instanciar objetos en el siguiente par de secciones, pero por ahora solamente tienes que saber que un constructor se usa para inicializar propiedades de objetos cuando el objeto está siendo creado.

Puedes definir un constructor definiendo el método \_\_construct.

Advertisement

**Métodos para clases de PHP**

Podemos ver a los métodos de las clases como funciones que llevan a cabo acciones específicas asociadas a los objetos. En la mayoría de los casos se usan para acceder y manipular las propiedades de los objetos, así como llevar a cabo operaciones relacionadas.

En el ejemplo anterior hemos definido el método getLastName, que devuelve el apellido asociado al objeto.

Así que esta ha sido una breve introducción a la estructura de las clases en PHP. En la siguiente sección veremos cómo instanciar objetos de la clase Employee.

**¿Qué es un objeto en PHP?**

En la sección anterior discutimos la estructura básica de una clase en PHP. Ahora bien, cuando quieres usar una clase necesitas instanciarla, y el resultado final es un objeto. Así que podemos pensar en una clase como un modelo, y un objeto es algo real con lo que puedes trabajar.

En el contexto de la clase Employee que acabamos de crear en la sección anterior, veamos cómo instanciar un objeto de esa clase.

|  |  |
| --- | --- |
| 1 | <?php |
| 2 | $objEmployee = new Employee('Bob', 'Smith', 30); |
| 3 |  |
| 4 | echo $objEmployee->getFirstName(); // print 'Bob' |
| 5 | echo $objEmployee->getLastName(); // prints 'Smith' |
| 6 | echo $objEmployee->getAge(); // prints '30' |
| 7 | ?> |

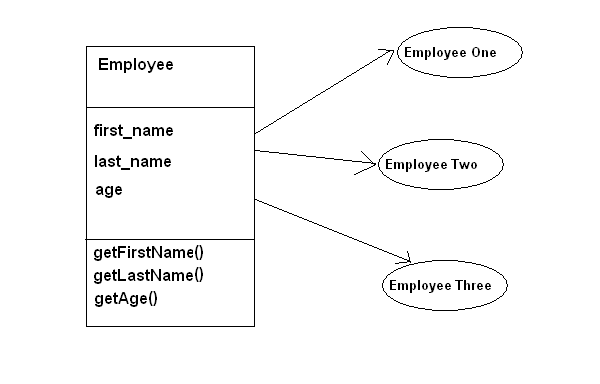
Necesitas usar la palabra clave new cuando quieres instanciar un objeto de cualquier clase junto con su nombre de clase, y obtendrás una nueva instancia de un objeto de dicha clase.

Si una clase ha definido el método \_\_construct y este requiere argumentos, necesitas enviar esos argumentos cuando creas una instancia de un objeto. En nuestro caso, el constructor de la clase Employee requiere tres argumentos, y por lo tanto los hemos enviado cuando creamos el objeto $objEmployee. Como discutimos anteriormente, el método \_\_construct se ejecuta automáticamente cuando el objeto es instanciado.

A continuación hemos invocado a métodos de la clase en el objeto $objEmployee para imprimir la información que fue inicializada durante la creación del objeto. Desde luego, puedes crear múltiples objetos de la misma clase, como se muestra en el siguiente fragmento de código.

|  |  |
| --- | --- |
| 1 | <?php |
| 2 | $objEmployeeOne = new Employee('Bob', 'Smith', 30); |
| 3 |  |
| 4 | echo $objEmployeeOne->getFirstName(); // prints 'Bob' |
| 5 | echo $objEmployeeOne->getLastName(); // prints 'Smith' |
| 6 | echo $objEmployeeOne->getAge(); // prints '30' |
| 7 |  |
| 8 | $objEmployeeTwo = new Employee('John', 'Smith', 34); |
| 9 |  |
| 10 | echo $objEmployeeTwo->getFirstName(); // prints 'John' |
| 11 | echo $objEmployeeTwo->getLastName(); // prints 'Smith' |
| 12 | echo $objEmployeeTwo->getAge(); // prints '34' |
| 13 | ?> |

La siguiente imagen es una representación gráfica de la clase Employee y algunas de sus instancias.



En pocas palabras, una clase es un modelo que puedes usar para crear objetos estructurados.

**Encapsulación**

En la sección anterior discutimos cómo instanciar objetos de la clase Employee. Es interesante notar que el objeto $objEmployee en sí mismo envuelve propiedades y métodos de la clase. En otras palabras, este oculta esos detalles del resto del programa. En el mundo de la POO, esto se conoce como encapsulación de datos.

La encapsulación es un aspecto importante de la POO que te permite restringir el acceso a ciertas propiedades o métodos del objeto. Y eso nos lleva a otro tema de discusión*:*los niveles de acceso.

Advertisement

**Niveles de acceso**

Cuando defines una propiedad o un método en una clase, puedes declararlo para que tenga uno de estos tres niveles de acceso *—*public, private o protected (público, privado o protegido).

**Acceso público**

Cuando declaras una propiedad o un método como public (público), este puede ser accedido desde cualquier lugar del exterior de la clase. El valor de una propiedad pública puede ser modificado desde cualquier lugar en tu código.

Veamos un ejemplo para comprender el nivel de acceso público.

|  |  |
| --- | --- |
| 1 | <?php |
| 2 | class Person |
| 3 | { |
| 4 | public $name; |
| 5 |  |
| 6 | public function getName() |
| 7 | { |
| 8 | return $this->name; |
| 9 | } |
| 10 | } |
| 11 |  |
| 12 | $person = new Person(); |
| 13 | $person->name = 'Bob Smith'; |
| 14 | echo $person->getName(); // prints 'Bob Smith' |
| 15 | ?> |

Como puedes ver en el ejemplo anterior, hemos declarado la propiedad name para que sea pública. Por lo tanto, puedes establecer su valor desde cualquier lugar del exterior de la clase, como lo hemos hecho aquí.

**Acceso privado**

Cuando declaras una propiedad o un método como private (privado), solamente puede accederse a él desde el interior de la clase. Esto significa que necesitas definir métodos get y set para recuperar y establecer el valor de esa propiedad.

Nuevamente revisemos el ejemplo anterior para comprender el nivel de acceso privado.

|  |  |
| --- | --- |
| 1 | <?php |
| 2 | class Person |
| 3 | { |
| 4 | private $name; |
| 5 |  |
| 6 | public function getName() |
| 7 | { |
| 8 | return $this->name; |
| 9 | } |
| 10 |  |
| 11 | public function setName($name) |
| 12 | { |
| 13 | $this->name = $name; |
| 14 | } |
| 15 | } |
| 16 |  |
| 17 | $person = new Person(); |
| 18 | $person->name = 'Bob Smith'; // Throws an error |
| 19 | $person->setName('Bob Smith'); |
| 20 | echo $person->getName(); // prints 'Bob Smith' |
| 21 | ?> |

Si intentas acceder a una propiedad privada desde el exterior de la clase, esta mostrará el error fatal Cannot access private property Person::$name (No se puede acceder a la propiedad privada Person::$name). Por lo tanto, necesitas establecer el valor de la propiedad privada usando el método set, como lo hicimos usando el método setName.

Existen buenas razones por las que quizá quieras establecer una propiedad como privada. Por ejemplo, tal vez alguna acción deba llevarse a cabo (actualizar una base de datos, por ejemplo, o volver a mostrar una plantilla) si esa propiedad cambia. En ese caso puedes definir un método set y gestionar cualquier lógica especial cuando la propiedad haya cambiado.

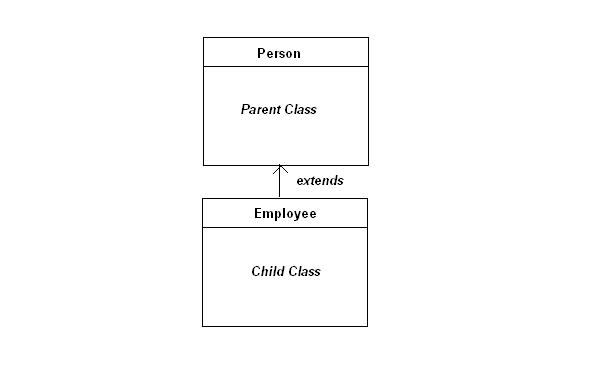
**Acceso protegido**

Finalmente, cuando declaras una propiedad o un método como protected (protegido), es posible acceder a él desde la misma clase que lo haya definido y desde las clases que hereden la clase en cuestión. Vamos a discutir la herencia en la siguiente sección, así que regresaremos al nivel de acceso protegido un poco más adelante.

**Herencia**

La herencia es un aspecto importante del paradigma de la programación orientada a objetos, que te permite heredar propiedades y métodos de otras clases extendiéndolas. La clase que está siendo heredada se conoce como **clase padre**, y la clase que hereda se conoce como **clase hija**. Cuando creas una instancia de un objeto de la clase hija, esta hereda las propiedades y métodos de la clase padre también.

Echemos un vistazo a la siguiente captura de pantalla para comprender el concepto de herencia.



En el ejemplo anterior, la clase Person es la clase padre, y la clase Employee extiende o hereda de la clase Person, por lo que se conoce como clase hija.

Intentamos revisar un ejemplo del mundo real para comprender cómo funciona.

|  |  |
| --- | --- |
| 1 | <?php |
| 2 | class Person |
| 3 | { |
| 4 | protected $name; |
| 5 | protected $age; |
| 6 |  |
| 7 | public function getName() |
| 8 | { |
| 9 | return $this->name; |
| 10 | } |
| 11 |  |
| 12 | public function setName($name) |
| 13 | { |
| 14 | $this->name = $name; |
| 15 | } |
| 16 |  |
| 17 | private function callToPrivateNameAndAge() |
| 18 | { |
| 19 | return "{$this->name} is {$this->age} years old."; |
| 20 | } |
| 21 |  |
| 22 | protected function callToProtectedNameAndAge() |
| 23 | { |
| 24 | return "{$this->name} is {$this->age} years old."; |
| 25 | } |
| 26 | } |
| 27 |  |
| 28 | class Employee extends Person |
| 29 | { |
| 30 | private $designation; |
| 31 | private $salary; |
| 32 |  |
| 33 | public function getAge() |
| 34 | { |
| 35 | return $this->age; |
| 36 | } |
| 37 |  |
| 38 | public function setAge($age) |
| 39 | { |
| 40 | $this->age = $age; |
| 41 | } |
| 42 |  |
| 43 | public function getDesignation() |
| 44 | { |
| 45 | return $this->designation; |
| 46 | } |
| 47 |  |
| 48 | public function setDesignation($designation) |
| 49 | { |
| 50 | $this->designation = $designation; |
| 51 | } |
| 52 |  |
| 53 | public function getSalary() |
| 54 | { |
| 55 | return $this->salary; |
| 56 | } |
| 57 |  |
| 58 | public function setSalary($salary) |
| 59 | { |
| 60 | $this->salary = $salary; |
| 61 | } |
| 62 |  |
| 63 | public function getNameAndAge() |
| 64 | { |
| 65 | return $this->callToProtectedNameAndAge(); |
| 66 | } |
| 67 | } |
| 68 |  |
| 69 | $employee = new Employee(); |
| 70 |  |
| 71 | $employee->setName('Bob Smith'); |
| 72 | $employee->setAge(30); |
| 73 | $employee->setDesignation('Software Engineer'); |
| 74 | $employee->setSalary('30K'); |
| 75 |  |
| 76 | echo $employee->getName(); // prints 'Bob Smith' |
| 77 | echo $employee->getAge(); // prints '30' |
| 78 | echo $employee->getDesignation(); // prints 'Software Engineer' |
| 79 | echo $employee->getSalary(); // prints '30K' |
| 80 | echo $employee->getNameAndAge(); // prints 'Bob Smith is 30 years old.' |
| 81 | echo $employee->callToPrivateNameAndAge(); // produces 'Fatal Error' |
| 82 | ?> |

Lo importante a destacar aquí es que la clase Employee ha usado la palabra clave extends para heredar de la clase Person. Ahora bien, la clase Employee puede acceder a todas las propiedades y métodos de la clase Person que estén declarados como públicos o protegidos (no puede acceder a miembros declarados como privados).

En el ejemplo anterior, el objeto $employee puede acceder a los métodos getName y setName que están definidos en la clase Person, ya que están declarados como públicos.

A continuación hemos accedido al método callToProtectedNameAndAge usando el método getNameAndAge definido en la clase Employee, ya que está declarado como protegido. Finalmente, el objeto $employee no puede acceder al método callToPrivateNameAndAge de la clase Person, ya que está declarado como privado.

Por otro lado, puedes usar el objeto $employee para establecer la propiedad age de la clase Person, como lo hicimos en el método setAge que está definido en la clase Employee, ya que la propiedad age está declarada como protegida.

Así que esta fue una breve introducción a la herencia. Te ayuda a reducir la duplicación de código, y por lo tanto fomenta la reutilización del código.

**Polimorfismo**

El polimorfismo es otro concepto importante en el mundo de la programación orientada a objetos, que se refiere a la habilidad para procesar objetos de manera diferente en base a sus tipos de datos.

Por ejemplo, en el contexto de la herencia, si la clase hija quiere cambiar el comportamiento del método de la clase padre, entonces puede sobrescribir ese método. Esto se conoce como sobreescritura de métodos. Echemos un vistazo rápido a un ejemplo de la vida real para comprender el concepto de la sobreescritura de métodos.

|  |  |
| --- | --- |
| 1 | <?php |
| 2 | class Message |
| 3 | { |
| 4 | public function formatMessage($message) |
| 5 | { |
| 6 | return printf("<i>%s</i>", $message); |
| 7 | } |
| 8 | } |
| 9 |  |
| 10 | class BoldMessage extends Message |
| 11 | { |
| 12 | public function formatMessage($message) |
| 13 | { |
| 14 | return printf("<b>%s</b>", $message); |
| 15 | } |
| 16 | } |
| 17 |  |
| 18 | $message = new Message(); |
| 19 | $message->formatMessage('Hello World'); // prints '<i>Hello World</i>' |
| 20 |  |
| 21 | $message = new BoldMessage(); |
| 22 | $message->formatMessage('Hello World'); // prints '<b>Hello World</b>' |
| 23 | ?> |

Como puedes ver, hemos cambiado el comportamiento del método formatMessage sobreescribiéndolo en la clase BoldMessage. Lo importante es que un mensaje recibe un formato diferente en base al tipo de objeto, ya sea que se trate de una instancia de la clase padre o de la clase hija.

(Algunos lenguajes orientados a objetos también tienen un tipo de sobrecarga de métodos que te permite definir múltiples métodos de clase con el mismo nombre, pero con un número diferente de argumentos. Esto no es compatible directamente en PHP, pero existen un par de soluciones para lograr una funcionalidad similar).

**Conclusión**

La programación orientada a objetos es un tema amplio, y apenas hemos tocado la superficie de su complejidad. Espero que este tutorial te haya ayudado a comenzar con los fundamentos de la POO y que te motive a continuar y aprender más temas avanzados sobre ella.