UD-03c: Orientación a Objetos con PHP

Desarrollo Web en Entorno Servidor

Curso 2020/2021

Orientación a objetos en PHP

Herencia

La herencia es un mecanismo de la POO que nos permite **definir nuevas clases en base a otra ya existente**. Las nuevas clases que heredan se conocen con el nombre de clase hija o subclases. La clase de la que heredan se llama clase padre o superclase.

Vamos a usar un ejemplo para entenderlo mejor. En una aplicación de tienda web tendremos productos de distintos tipos. Para comenzar, creamos una clase *Producto* con los atributos comunes a todos los productos y un método que genera una salida personalizada en HTML:

```
class Producto
{
    private $codigo;
    private $nombre;
    private $nombre_corto;
    private $PVP;

    public function muestra()
    {
        print "<p>" . $this->codigo . "</p>";
    }
}
```

Orientación a objetos en PHP

Herencia

Esta clase es suficiente si la única información que tenemos de los distintos productos es la que tiene la clase. Sin embargo, si quieres personalizar la información que vas a tratar de cada tipo de producto (y almacenar, por ejemplo para los televisores, las pulgadas que tienen o su tecnología de fabricación), puedes crear nuevas clases que hereden de Producto.

```
class TV extends Producto
{
    private $pulgadas;
    private $tecnologia;
}
```

Orientación a objetos en PHP

Los nuevos objetos que se instancien a partir de la subclase serán también objetos de la clase padre. Se puede comprobar mediante el operador **instanceof**.

```
$tv = new TV();
if ($tv instanceof Producto)
    // Este código se ejecuta pues la condición es cierta
```

Funciones de utilidad en la herencia en PHP5

Función	Ejemplo	Significado
get_parent_class	`echo "La clase padre es: " . get_parent_class($t);`	Devuelve el nombre de la clase padre del objeto o la clase que se indica.
is_subclass_of	`if (is_subclass_of($t, 'Producto') { …`	Devuelve true si el objeto o la clase del primer parámetro, tiene como clase base a la que se indica en el segundo parámetro, o false en caso contrario.

Orientación a objetos en PHP

Acceso protegido (protected)

- La nueva clase hereda todos los atributos y métodos de la clase base o padre, pero no podrá acceder a aquellos que sean privados. Para crear en la clase base un método no visible al exterior, pero accesible desde las subclases, usa **protected**.
- Además, se puede redefinir el comportamiento de los métodos existentes en la clase base creando en la subclase un nuevo método con el mismo nombre.

```
class TV extends Producto
{
    private $pulgadas;
    private $tecnologia;

    private function muestra()
    {
        print "<p>" . $this->pulgadas . " pulgadas</p>";
    }
}
```

Orientación a objetos en PHP

Clases y métodos finales

Existe una forma de evitar que las clases heredadas puedan redefinir el comportamiento de los métodos existentes en la superclase: utilizar **final**.

```
class Producto
{
    private $codigo;
    private $nombre;
    private $nombre_corto;
    private $PVP;

    public final function muestra()
    {
        print "<p>" . $this->codigo . "</p>";
    }
}
```

En este caso, el método muestra ya no podría redefinirse en la clase TV.

Se puede **declarar una clase con final**, pero no se podrían crear clases heredadas.

Orientación a objetos en PHP

Clases y métodos abstractos (abstract)

Opuestamente a ***final***, existe **abstract**. **Se usa igual**, tanto con métodos como con clases completas, **pero** en lugar de prohibir la herencia, **obliga a que se herede**.

Por tanto, una clase con el modificador **abstract** no puede tener objetos que la instancien, pero sí podrá utilizarse de clase base.

Sus subclases sí podrán utilizarse para instanciar objetos.

```
abstract class Producto
{
        …
}
class TV extends Producto
{
        …
}
$tv = new TV();
```

Orientación a objetos en PHP

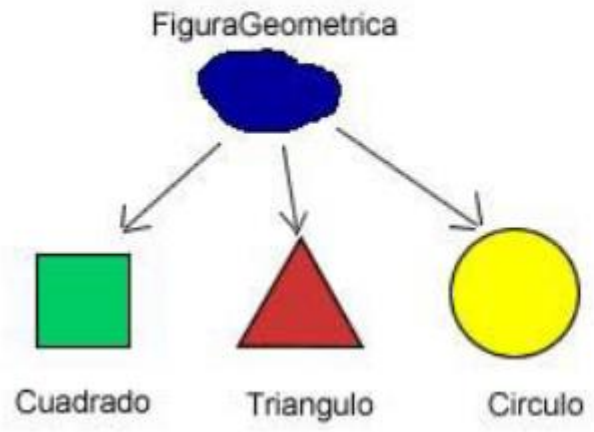
- Un **método abstracto** debe ser redefinido obligatoriamente por las subclases.
- No contiene código.

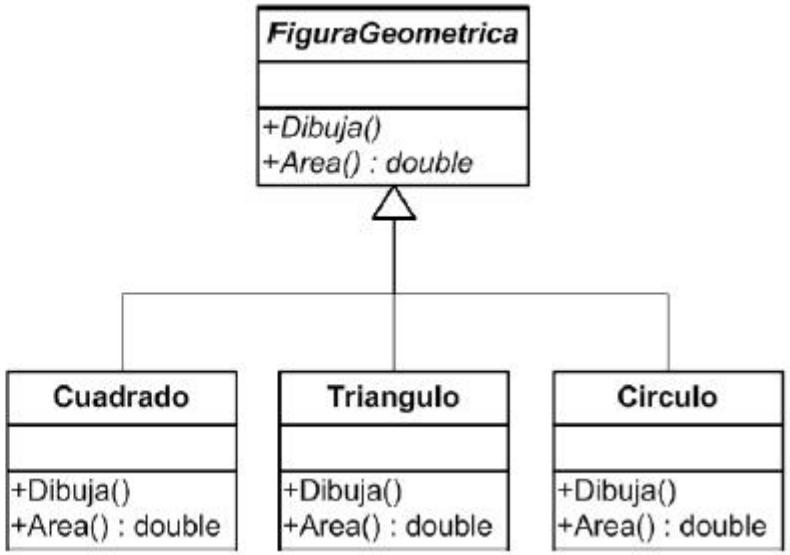
```
class Producto
{
        …
        abstract public function muestra();
}
```

Obviamente, no se puede declarar una clase como abstract y final simultáneamente. Abstract obliga a que se herede para que se pueda utilizar, mientras que final indica que no se podrá heredar.

Orientación a objetos en PHP

Ejemplo con figuras geométricas





Orientación a objetos en PHP

Ejemplo con figuras geométricas

```php
abstract class Figura
{
     public $color;
     public function __set($name, $value)
     {
          if ($name == 'Color' && is_string($value) === true)
              $this->color = $value;
     }
     abstract public function Dibuja();
     abstract public function Area();
}

class Cuadrado extends Figura
{
     public function Dibuja()
     {
          echo 'Dib Cuadrado '.$this->color;
     }
     public function Area()
     {
          return 0;
     }
}

// Círculo y Triángulo se implementan igual que Cuadrado
```

```php
// Uso de las clases
$cuadrado = new Cuadrado();
$cuadrado->Color = 'Negro'; // Estamos llamando al __set de Figura
$cuadrado->Dibuja();
```

Dib Cuadrado Negro

Orientación a objetos en PHP

Llamar a métodos de la clase base

Para facilitar la creación de nuevos objetos, vamos a crear un constructor al que se le pasará un array con los valores de los atributos del nuevo producto.

```
class Producto
{
     public $codigo;
     public $nombre;
     public $nombre_corto;
     public $PVP;
     public function muestra()
     {
          print "<p>" . $this->codigo . "</p>";
     }
     public function __construct($row)
     {
          $this->codigo = $row['cod'];
          $this->nombre = $row['nombre'];
          $this->nombre_corto = $row['nombre_corto'];
          $this->PVP = $row['PVP'];
     }
}
```

Orientación a objetos en PHP

Ahora cuando creemos un nuevo objeto la clase TV, que hereda de Producto, se llamará al constructor de Producto. Pero podemos crear un nuevo constructor específico de TV que redefina el comportamiento de la clase base.

Dependiendo de si programamos o no el constructor en la clase heredada, se llamará a uno u otro, y si queremos, además, llamar al constructor de la clase base, utilizaremos **parent::**

```
class TV extends Producto
{
     public $pulgadas;
     public $tecnologia;
     public function muestra()
     {
          print "<p>" . $this->pulgadas . " pulgadas</p>";
     }
     public function __construct($row)
     {
          parent::__construct($row);
          $this->pulgadas = $row['pulgadas'];
          $this->tecnologia = $row['tecnologia'];
     }
}
```

Orientación a objetos en PHP

Interfaces

- Son como una clase que solamente contiene declaraciones de métodos.
- Se definen utilizando la palabra **interface**.
- Por ejemplo, vimos que podíamos crear nuevas clases heredadas de Producto, como TV y que en las subclases podíamos redefinir el comportamiento del método ***muestra()*** para cada tipo de producto. Pues, si queremos asegurarnos de que todos los tipos de productos tengan un método ***muestra()***, hemos de crear un interface como el siguiente:

```
interface iMuestra
{
    public function muestra();
}
```

Orientación a objetos en PHP

Ahora cuando creamos las subclases indicaremos con la palabra **implements** que han de implementar los métodos declarados en esta interface.

```
class TV extends Producto implements iMuestra
{
    …
    public function muestra()
    {
        print "<p>" . $this->pulgadas . " pulgadas</p>";
    }
    …
}
```

- **Todos los métodos que se declaren en un interface deben ser públicos.**
- Además de métodos, los interfaces podrán contener constantes, pero no atributos.

Al implementar todos los métodos declarados en una interface se asegura la interoperabilidad entre clases. Si sabemos que una clase implementa un interface determinada, sabremos qué nombre tienen sus métodos, qué parámetros debemos pasar y podremos averiguar con más facilidad con qué objetivo han sido escritos.

Orientación a objetos en PHP

Interfaces y herencia múltiple

En *php* no existe la herencia múltiple. Sin embargo, es posible crear clases que implementen varios interfaces, simplemente, separando la lista de interfaces por comas después de la palabra **implements**.

```
class TV extends Producto implements iMuestra, Countable
{
        …
}
```

La única restricción es que los nombres de los métodos que se deban implementar en los distintos interfaces no coincidan. Por ejemplo, el interface `iMuestra` no podría contener un método `count`, pues éste ya está declarado en `Countable`.

Orientación a objetos en PHP

```php
interface IACadena
{
        function __toString();
}

interface IDestructor
{
        function __destruct();
}

class A
{
    protected $datoA;

    public function __construct($dato)
    {
        $this->datoA = $dato;
    }
}
```

```php
class B extends A implements IACadena, IDestructor
{
    private $datoB;

    public function __construct($datoB, $datoA)
    {
        parent::__construct($datoA);
        $this->datoB = $datoB;
    }

    // Método forzado a implementar por el interfaz IDestructor
    public function __destruct()
    {
        $this->datoB = null;
    }

    // Método forzado a implementar por el interfaz IACadena
    public function __toString()
    {
        return sprintf("%04d %04d", $this->datoA, $this->datoB);
        /* sprintf se usa para lo mismo que printf,
        pero en vez de escribirlo en la salida estándar,
        lo devuelve como una cadena de caracteres.  */
    }
}
```

Orientación a objetos en PHP

Herencia de interfaces

Podemos también crear interfaces heredando de otros ya existentes, mediante extends.

```php
interface a
{
    public function foo();
}

interface b extends a
{
    public function baz(Baz $baz);
}
```

```php
class c implements b
{
    public function foo()
    {
    }

    public function baz(Baz $baz)
    {
    }
}
```

Orientación a objetos en PHP

Interfaces VS clases abstractas

Clases abstractas	Interfaces
Permiten definir reglas para las clases que los implementen o hereden	
No permiten instanciar objetos	
Los métodos pueden contener código (nos podría interesar si varias subclases tienen un comportamiento común)	Los métodos NO pueden contener código (si varias subclases tienen un comportamiento común, habría que repetir el código en ellas)
Pueden contener atributos	NO pueden contener atributos
NO se puede crear una clase que herede de dos clases abstractas	Se puede crear una clase que implemente varios interfaces

EJERCICIOS

- A partir de la clase ***Contacto*** creada con anterioridad, crea otras dos clases que **heredan** de ella, con las siguientes características:

- La primera clase debe llamarse **EContacto** y contendrá dos atributos adicionales: **email** para almacenar el correo electrónico y **url** para almacenar una URL de una página personal.
- La segunda clase debe llamarse **PContacto** y tendrá los campos **dirección, ciudad y provincia.**

Ambas clases han de disponer de su propio constructor, así como de un método para realizar la conversión a cadena.

Por último, crea un objeto Agenda en un script php llamado **ud03ejer05.php**. Añade tres objetos **EContacto**. Después, añade otros tres objetos **PContacto** y muestra el contenido de la agenda en el navegador.

EJERCICIO *'LIGA DE BALONCESTO'*

Se precisa una app para el control de estadísticas de una liga de baloncesto.

Se han de construir las siguientes clases:

Jugador

dorsal
nombre
estatura (en centímetros)

+ __construct()
+ __get()
+ __set()

Base (hereda de jugador)

- Asistencias

+ __construct()
+ __get()
+ __set()

Pivot (hereda de jugador)

- Rebotes

+ __construct()
+ __get()
+ __set()

Escolta (hereda de jugador)

- Robos

+ __construct()
+ __get()
+ __set()

Alero (hereda de jugador)

- Puntos

+ __construct()
+ __get()
+ __set()

AlaPivot (hereda de jugador)

- Tapones

+ __construct()
+ __get()
+ __set()

Equipo

- nombre
- quinteto (array de 5 objetos, NUNCA DE LA CLASE JUGADOR, que no se debe poder instanciar)

Un Equipo tendrá 5 objetos, uno de cada una de las 5 clases hijas de Jugador.

+ __construct() que recibe el nombre del equipo y los 5 objetos del array. Este método debe visualizar el nombre y el tipo (clase) de los 5 integrantes.
+ __get()
+ estaturaMedia(), que devuelve la estatura media de los 5 integrantes del equipo.
+ estaturaMaxima(), que devuelve la mayor estatura de los integrantes.

Enunciado:

Crea las 7 clases en archivos diferentes y ubícalas en una carpeta.
A continuación, fuera de esa carpeta, crea una página llamada **ud03ej06.php.**
En esta página, inventa un equipo con sus jugadores. Imprímelo por pantalla.
Después, muestra los resultados de invocar a los métodos *estaturaMedia* y *estaturaMaxima*. Se tendrá en cuenta la estética de las visualizaciones y la idoneidad de los procedimientos realizados.