Práctica 3

Sesiones 5, 6, 7 y 8

(Continuación)

Orientación a objetos en PHP

https://www.php.net/manual/es/language.oop5.php

La estructura básica de una clase es la siguiente:

```
class Coche
{
        /* Lista de atributos */
        private $color= "negro"; /* Asignación de un valor por defecto. */
        const NUM_RUEDAS=4;
        private $combustible;
        private $puertas= ["delanteras","traseras"];
        static public $numeroCoches=0;

        /* Lista de métodos */
        /* Constructor */
        function __construct($color)
        {
            Coche::$numeroCoches++;
            $this->combustible=60;
           $this->color=$color;
        }
        /* Destructor */
        function __destruct()
        {
             echo "Hasta luego, Lucas";
            Coche::$numeroCoches--;
        }
        private function echarGasofa(int $cantidad): void
```

```
        {
            $this->combustible +=$cantidad;
        }
        public function ponerCombustible (int $cantidad): void
        {
                echarGasofa($cantidad);
        }
        public function consumirCombustible (int $cantidad): void
        {
                $this->combustible=$this->combustible-$cantidad;
                try
                {
                        if ($this->combustible < 0)
                                { throw new Exception( "Se ha quedado sin gasofa."); }
                }
                catch (Exception $e)
                {
                        echo $e->getMessage();
                        echo "\n<br />\n";
                        exit();
                }
        }
        public static function calcularAutonomia()
        {
         /* Código */
        }
}
```

Varios comentarios:

- La referencia a las variables de clase o atributos desde dentro de la clase se hace mediante *$this->atributo*, en caso de que pueda haber problemas con variables que se llamen igual.
- Los atributos y los métodos pueden ser públicos (por defecto, los atributos y las funciones son públicos) o privados (*private* – hay que indicarlo explícitamente). Los primeros pueden ser referenciados desde fuera de la clase y los segundos no. También pueden ser *protected*: son públicos para cualquier clase que herede y privados para el resto.
- Además, también pueden ser readonly, evitando así que se modifique un atributo después de la inicialización:

```
class Usuario {
   public readonly string $id;


   public function __construct(string $id) {
```

```
    $this->id = $id;
  }
```

- Dentro de los métodos de la clase, éstos se invocan empleando *$this→metodo();*
- La creación de objetos se hace mediante la palabra reservada *new*: *$coche1= new Coche("rojo");*
- PHP ofrece una serie de métodos que comienzan con dos caracteres de subrayado consecutivos (__), entre ellos *__construct, __destruct* y *__clone*.
- El método *__clone* sirve para clonar un objeto (crear uno nuevo y copiar los valores del clonado):

```
$coche1= new Coche("rojo");
$coche1->ponerCombustible(10);
$coche2= clone $coche1;
```

Si no se sobreescribe en la clase, se copian todos los atributos de los objetos. Se puede sobreescribir, dándole la funcionalidad deseada:

```
function __clone()
        { $this-> combustible=100; }
```

En este ejemplo, sólo el atributo combustible se inicializa. El resto, no.

Hay que tener cuidado porque si tenemos dos objetos, o1 y o2, y se asigna uno a otro, *o1=o2*, no se está haciendo una copia, sino una referencia desde o1 a o2, es decir, referencian el mismo objeto en memoria.

- El método *__destruct* se emplea para ejecutar varias sentencias cuando se destruye un objeto. Para destruir un objeto, se emplea *unset(objeto),* y en ese momento se ejecuta el método *__destruct* si se ha incluido en la clase:

```
        $coche1= new Coche("azul");
        unset($coche1);
```

- La gestión de excepciones es del estilo a la de otros lenguajes: se lanza una excepción (indicando un mensaje que explique qué ha ocurrido) y seguidamente se captura, momento en el cual se trata la misma. En el ejemplo, se muestra el mensaje con que se lanzó (mediante el método *getMessage*) y se termina la ejecución del guión.
- Los atributos estáticos son comunes a todos los objetos y se acceden mediante :: junto con el nombre de una clase o de un objeto *(también funciona si MiClase se sustituye por un objeto de dicha clase)*: *MiClase::$numeroObjetos*.
- Las constantes de clase se declaran con la palabra reservada *const* y se accede como las variables estáticas:

```
        class MiClase {
            const MICONSTANTE=123;
        }
        /* Se usan como sigue: */
        echo MiClase::MICONSTANTE;
```

- Los métodos estáticos también se asocian a la clase, declarándose como *static* y se invocan a partir de ella y también con los dos puntos (*Coche::calcularAutonomia();*).
- Dentro de una clase, para referenciar un atributo o método estático se puede emplear el nombre de la clase, como ya hemos comentado, o la palabra reservada *self*, que se asocia con el nombre de la clase al igual que *$this* se asocia con el nombre del objeto (ej.: *self::metodo_estatico()*).

- En ciertas ocasiones, y para poder detectar problemas en cuanto a los tipos de argumentos formales y actuales de métodos en particular y de funciones en general, se emplean "pistas", o "hint", indicando el tipo de los argumentos formales. Si queremos que salte una excepción cuando se pasa como parámetro una variable del tipo incorrecto, recordar que es necesario usar *declare(strict_types=1)*.

```
public function pintar(Coche $coche, string $color): void  {…}
```

- Métodos interesantes: *__get, __set* y *__call*. ¿Para qué sirven?
- En PHP7 se introducen las clases anónimas, que permiten crear un único objeto sin necesidad de nombrar la clase.

```
$objeto→metodo( new class {
        public function escribir(string $texto): void
            { echo $texto; }
});
```

En este ejemplo, pasamos como parámetro al método *metodo* de la instancia *$objeto* una instancia de una clase anónima (creada con *new class {...}*), la cual tiene un método público *escribir*.

- Herencia: se emplea la palabra reservada *extends*.

```
<!-- Ejemplo tomado del libro: Beginning PHP 5.3. Matt Doyle. Wiley Publishing, Inc. 2010 -->
<!DOCTYPE html>
<html>
<head>
    <title>Creating Shape Classes using Inheritance</title>
    <link rel="stylesheet" type="text/css" href="common.css" />
</head>
<body>
  <h1>Creating Shape Classes using Inheritance</h1>
 <?php
    class Shape {
    private $_color = "black";
    private $_filled = false;
    public function getColor() {
            return $this->_color;
    }
    public function setColor( string $color ): void {
            $this->_color = $color;
    }
    public function isFilled() : bool{
            return $this->_filled;
    }
    public function fill(): void {
            $this->_filled = true;
```

```
}
public function makeHollow(): void {
        $this->_filled = false;
}
}
class Circle extends Shape {
private $_radius = 0;
public function getRadius(): int{
        return $this->_radius;
}
public function setRadius( int $radius ): void {
        $this->_radius = $radius;
}
public function getArea(): int {
        return M_PI * pow( $this->_radius, 2 );
}
}
class Square extends Shape {
private $_sideLength = 0;
public function getSideLength(): int {
        return $this->_sideLength;
}
public function setSideLength( $length ): int {
        $this->_sideLength = $length;
}
public function getArea(): int {
        return pow( $this->_sideLength, 2 );
}
}
$myCircle = new Circle;
$myCircle->setColor( "red" );
$myCircle->fill();
$myCircle->setRadius( 4 );
echo "<h2>My Circle</h2>";
echo "<p>My circle has a radius of " . $myCircle->getRadius() . ".</p>";
echo "<p>It is " . $myCircle->getColor() . " and it is " . ( $myCircle-> isFilled() ? "filled" : "hollow"
) . ".</p>";
echo "<p>The area of my circle is: " . $myCircle->getArea() . ".</p>";
$mySquare = new Square;
```

```
    $mySquare->setColor( "green" );
    $mySquare->makeHollow();
    $mySquare->setSideLength( 3 );
    echo "<h2>My Square</h2>";
    echo "<p>My square has a side length of " . $mySquare->getSideLength() . ".</p>";
    echo "<p>It is " . $mySquare->getColor() . " and it is " . ( $mySquare-> isFilled() ? "filled" :
    "hollow" ) . ".</p>";
    echo "<p>The area of my square is: " . $mySquare->getArea() . ".</p>";
 ?>
</body>
</html>
```

- Sobrescritura de métodos:

```
<!-- Ejemplo tomado del libro: Beginning PHP 5.3. Matt Doyle. Wiley Publishing, Inc. 2010 -->
<!DOCTYPE html >
<html>
<head>
    <title>Overriding Methods in the Parent Class</title>
    <link rel="stylesheet" type="text/css" href="common.css" />
</head>
<body>
    <h1>Overriding Methods in the Parent Class</h1>
    <?php
    class Fruit {
            public function peel(): void{
                    echo "<p>I'm peeling the fruit...</p>";
            }
            public function slice(): void {
                    echo "<p>I'm slicing the fruit...</p>";
            }
            public function eat(): void {
                    echo "<p>I'm eating the fruit. Yummy!</p>";
            }
            public function consume(): void {
                    $this->peel();
                    $this->slice();
                    $this->eat();
            }
    }
    class Grape extends Fruit {
```

```
        public function peel(): void {
            echo "<p>No need to peel a grape!</p>";
        }
        public function slice(): void {
            echo "<p>No need to slice a grape!</p>";
        }
    }
    echo "<h2>Consuming an apple...</h2>";
    $apple = new Fruit;
    $apple->consume();
    echo "<h2>Consuming a grape...</h2>";
    $grape = new Grape;
    $grape->consume();
    ?>
    </body>
</html>
```

- En la sobrescritura en clases derivadas de métodos de la clase base, hay ocasiones en que podemos llamar al método original. Para tal fin, empleamos *parent::metodo_original(...);*
- Para evitar que se produzca la herencia o la sobrescritura de métodos se emplea delante de la clase o del método correspondiente, respectivamente, la palabra reservada *final*.
- Para declarar funciones abstractas, se emplea la palabra reservada *abstract* delante de la definición del método. Si una clase tiene uno o más métodos abstractos, la clase también se tiene que declarar abstracta. Las clases que hereden de ella deberán implementar los métodos abstractos. No se pueden crear objetos de una clase abstracta.

```
abstract class MiClaseAbstracta {
    abstract public function miMetodoAbstracto(string $param1, string $param2 ):void;
}
```

Interfaces:

```
interface MiInterfaz {
    public function metodo1( string $param1, string $param2 ): void;
    public function metodo2( string $param1, string $param2 ): void;
}
class MiClase implements MiInterfaz {
    public function metodo1( string $param1, string $param2 ): void {
        // (implementación del método.)
    }
    public function metodo2( string $param1, string $param2 ): void {
        // (implementación del método.)
    }
```

```
}
```

- Lo habitual es disponer cada clase en un fichero aparte y luego incluirlas en el fichero .php donde se vayan a emplear. Por ejemplo, hemos implementado la clase *Persona* en el fichero *Person.inc.php* y en lo incluimos en el guión correspondiente donde vamos a usarla:

```
<?php
        require_once( __DIR__."classes/Persona.inc.php" );
        $p = new Persona();
?>
```

Pero puede haber veces que se nos olvide realizar la inclusión de las clases en los guiones. En ese caso es interesante dotar al guión de una función denominada spl_autoload_register, la cual será llamada cuando intentemos hacer uso de una clase no definida en el guión, en cuyo caso esta será cargada automáticamente. Esta función podría tener la siguiente forma:

```
<?php
        spl_autoload_register(function($nombreDeLaClase)
        { require_once __DIR__ . "/classes/$nombreDeLaClase.php"; });
        /* Si ClaseEjemplo no existe en el guión, se intentará cargar automáticamente la clase
            ejecutando  require_once(__DIR__. "classes/ClaseEjemplo.php" );
        $objeto= new ClaseEjemplo("hola");
        /*... Resto del guión ...*/
?>
```

- Para convertir objetos a cadenas de caracteres se emplea la función *$cadena=serialize($objeto);* y para el fin contrario*, $objeto=unserialize($cadena);*
- Para obtener la clase de un objeto se usa la función *get_class($objeto).*

Ejercicio:

Escribe una clase Calculadora, que almacene dos valores y que pueda realizar las operaciones de suma, resta, multiplicación y división con ellos. Crea otra clase, CalculadoraAvanzada, que herede de Calculadora y que implemente las operaciones de máximo común divisor, mínimo común múltiplo y potencia (a^b).

Ejercicio:

Escribe una clase para construir un formulario con campos de texto. Como dato miembro se almacena un array donde se guardarán los campos de texto que se vayan creando. Existirá un método que añada al formulario un nuevo campo de texto a partir del identificador del campo, valor del atributo name y del valor por defecto y de la etiqueta asociada. Además habrá otro método que se encargue de devolver el formulario completo en html.

Bibliografía

- https://php.net/manual/es/language.oop5.php