PRÁCTICA 2. EVOLUCIÓN DEL PHP

El propósito de este ejercicio práctico es:

- Repasar de forma práctica las características que PHP5 ofrece como lenguaje orientado a objetos.
- Reflejar las diferencias y similitudes entre las versiones 4 y 5 del lenguaje PHP haciendo que el alumno las ponga de manifiesto.

2.1 Ejercicio "*empresa*".

Dado el enunciado siguiente, cree las clases con sus atributos y métodos correspondientes:

Una empresa tiene una razón social, un domicilio fiscal, una dirección web, un tipo (S.A, S.L, autónomo, S.A.U) y un presupuesto de salarios, para todos sus empleados.

Un empleado es una persona con nombre y apellidos, y un número de seguridad social. Cuando se contrata a una persona, se le proporciona un email del dominio de la empresa, un teléfono móvil y un sueldo anual que tiene que estar dentro del presupuesto salarial.

El sistema de información podrá mostrar tanto a una empresa con todos sus empleados, como a un empleado mediante un método "*show*". Implemente el método "*show*" en ambas clases.

Recomendaciones: Los atributos son siempre privados por lo que hay que establecer métodos para fijar y obtener resultados. Defina las interfaces que considere necesarias. Utilice iteradores para navegar por las relaciones. Defina y utilice las excepciones que considere necesarias para mantener las restricciones del enunciado.

2.2 Ejercicio "*Lista enlazada*".

Implemente con la clase *ListaEnlazada*. Como su nombre indica, este *TAD* (tipo abstracto de datos) tan común, implementa internamente una lista enlazada de nodos (clase *Nodo*). Las operaciones de esta lista serán las siguientes:

- Constructor.
- Destructor.
- InsertarEnCabeza.
- InsertarEnCola.
- EsVacia.

Haga que su lista cumpla la interfaz *Iterator* e *IteratorTraversable* vista en los ejemplos de teoría y añada el método:

- *MostrarLista*.
- Sobrescriba el método *__clone*() para que realice una copia de la lista completa.
- Sobrescriba el método *__destruct*() para liberar todos los recursos (nodos). Sobrecargue también el destructor de la clase nodo para que muestre un mensaje antes de desaparecer.

2.3 Sobrecarga.

Hemos visto que una diferencia importante de PHP con respecto a otros lenguajes OO como Java, C++ es la peculiar forma en la que ofrece soporte a la sobrecarga.

Dada la clase "Multiplicador" que aparece a continuación:

```php
<?php
class Multiplicador extends Sobrecargable
{
    function multiplica_2($uno, $dos)
    {
        return $uno * $dos;
    }
    function multiplica_3($uno, $dos, $tres)
    {
        return $uno * $dos * $tres;
    }
}
?>
```

El alumno debe implementar la superclase abstracta "*Sobrecargable*" de la que "*Multiplicador*" hereda para que pueda ejecutarse correctamente el código siguiente:

```php
<?php
$multi = new Multiplicador();
echo $multi->multiplica(5, 6)."\n";
echo $multi->multiplica(5, 6, 3)."\n"
?>
```

Veasé que el método invocado se llama *multiplica()*, y no *multiplica_2()* ni *multiplica_3()*. Además, está prohibida la modificación de la clase "*Multiplicador*", y que el método *multiplica()* haga directamente las operaciones de multiplicación.

2.4 Clonado.

Hemos visto la importancia del método __clone() cuando trabajamos con objetos que contienen otros objetos (agregaciones).

Dadas las clases siguientes:

```php
<?php
class Direccion
{
      protected $ciudad;
      protected $pais;

      public function setCiudad($ciudad) { $this->ciudad = $ciudad; }
      public function getCiudad() { return $this->ciudad; }
      public function setPais($pais) { $this->pais = $pais; }
      public function getPais() { return $this->pais;}
}

class Persona
{
      protected $nombre;
      protected $direccion;

      public function __construct() { $this->direccion = new Direccion(); }
      public function setNombre($nombre) { $this->nombre = $nombre; }
      public function getNombre() { return $this->nombre; }
      public function __call($method, $arguments) {
         if (method_exists($this->direccion, $method)) {
            return call_user_func_array( array($this->direccion, $method), $arguments);
         }
      }
}
?>
```

Indique el resultado de la ejecución del siguiente código:

```php
<?php
$jose = new Persona ();
$jose->setNombre('Jose Fernández');
$jose->setCiudad('Valencia');

$pedro = clone $jose;
$pedro->setNombre('Pedro López');
$pedro->setCiudad('Castellón');

print $jose->getNombre() . ' vive en ' . $jose->getCiudad() . '.';
print $pedro->getNombre() . ' vive en ' . $pedro->getCiudad() . '.';
?>
```

¿Es el comportamiento deseado? ¿Cómo arreglarlo?

2.5 Cuestiones

Aporte su opinión respecto a las siguientes cuestiones:

1. Durante el desarrollo de la teoría, hemos observado que muchas características propias de los lenguajes *OO* no estaban disponibles en *PHP4*. Sin embargo, en caso de necesitarlas el programador disponía de mecanismos para simularlas. ¿Cómo podríamos convertir una clase *PHP4* en *final*? Es decir, ¿cómo podríamos evitar que una clase tenga clases derivadas o subclases? Proponga un ejemplo.

2. ¿Qué nos permiten hacer las interfaces que no permiten las clases abstractas?

3. ¿Qué nos permiten hacer las clases abstractas que no permiten las interfaces?