Orientación a objetos en PHP

Realizado por A.Garay (dpto. de Informática)

Clase básica

```php
class ClaseBasica
{
    // Atributo privado
    private $atributo = 'valor por defecto';
    // Método público
    public function mostrarAtributo() {
        echo $this->atributo;
    }
}
$c = new ClaseBasica();
$c->mostrarAtributo();
```

- Los atributos, al igual que el resto de variables en PHP deben ir precedidos del símbolo $.
- El equivalente al operador **.** de JAVA es **->** en PHP.
- ¡¡ OJO!! Se accede al atributo sin el $
- Para crear un objeto, el operador es también “new” .
- Los atributos y métodos pueden ser **public, private o protected**. No tiene sentido el acceso “default” o “de paquete”, ya que no existe el concepto de paquete en PHP.
- También podrían ser **static**.
- Para acceder a un método o atributo static se haría con el operador **::**

Constructores

```
class Punto
{
    private $x;
    private $y;
    private $z;
    public function Punto($x=0, $y=0, $z=0)
    {
        $this->x = $x;
        $this->y = $y;
        $this->z = $z;
    }
}
```

- Al igual que en JAVA, son functions cuyo nombre es idéntico al de la clase que construye.
- Sirven normalmente para dar un valor inicial a los atributos o para hacer labores de conexión a Bases de Datos o preparación en general.
- A partir de PHP5 se puede utilizar el nombre reservado **__construct(...)**
  - Es como el “create(...)” que se utiliza en UML para designar a los constructores
- También se puede utilizar el método **__toString()** que tiene el mismo significado y funcionalidad que en JAVA

La variable $this

```php
class A
{
    function foo()
    {
        if (isset($this)) {
            echo '$this está definida (';
            echo get_class($this);
            echo ")\n";
        } else {
            echo "\$this no está definida.\n";
        }
    }
}
```

```php
$a = new A();
$a->foo();
// Nota: la siguiente línea arrojará un
Warning si E_STRICT está habilitada.
A::foo();
```

- Se refiere al propio objeto, igual que **this** en JAVA
- get_class($obj) devuelve un string con el nombre de la clase de un objeto
- En PHP se puede hacer una llamada estática a un método incluso si éste no lo es, aunque no tiene sentido y es bastante desaconsejable.

Sobrecarga

```php
<?php
class A {
    public function f($x) {
        switch (gettype ( $x )) {
            case "integer" :
                echo $x + 3, "<br/>";
                break;
            case "string" :
                echo "Hola $x";
        }
    }
}
$a = new A ();
$a->f ( 5 );          //Mostraría 8
$a->f ( "Pepe" );     // Mostraría Hola Pepe
?>
```

- En PHP no existe la sobrecarga tal como se concibe en otros lenguajes como JAVA.
- El uso de valores por defecto y funciones *variádicas* hacen que podamos simular sobrecarga por número de parámetros, utilizando las funciones **func_num_args()**, **func_get_arg()** y **func_get_args()**
- Para sobrecargar por tipo habrá que recurrir a argucias como la del ejemplo, utilizando la función **gettype()**

Herencia

```php
<?php
class A {
    public function f() {
        echo "f-A";
    }
}
class B extends A {
    public function f() {
        echo "f-B"," # ";
        parent::f();
    }

}
$b = new B ();
$b->f (); // Saldría f-B # f-A
?>
```

- Se utiliza la palabra reservada “extends”.
- El equivalente a **super** en JAVA es **parent** en PHP.
- PHP tampoco soporta la herencia múltiple.
- Al igual que en JAVA, se pueden sobrescribir los métodos.
- Para acceder al código del método sobrescrito se puede hacer con **parent**

Interfaces

```php
interface ICantante {public function cantar();}
interface IDeportista {public function entrenar();}
class CantanteSano implements ICantante,
                              IDeportista {
    function cantar(){echo "AAAHH <br/>";}
    function entrenar(){echo "UUFFF <br/>";}
}
class Deportista implements IDeportista {
    function entrenar(){echo "OUCHHH <br/>";}
}
$c = new CantanteSano();
$d = new Deportista();
$a = [$c,$d];
foreach ($a as $v) {
    $v->entrenar();
}   //Sale UFFF y OUCHHH
```

Describen la “estructura” de una clase sin especificar su implementación.

Las clases que las implementan deben escribir código para sus métodos o declararlos abstractos.

PHP soporta la implementación múltiple, sólo si no hay métodos que tengan el mismo nombre y distinto número de parámetros

El polimorfismo funciona, sin embargo no es tan crítico como en JAVA porque no existen las complicaciones derivadas del up/down casting, al ser PHP tan débilmente tipado