INTRODUCCIÓ a la Programació Orientada a Objectes, POO AMB PHP

INTRODUCCIÓ a POO AMB PHP

La POO és un paradigma de programació (o tècnica de programació) que utilitza objectes i interaccions entre ells en el disseny d'un sistema.

Paradigma: teoría, el nucli central de la qual [...] subministra la base i model per a resoldre problemes [...] (Definició de la Reial Acadèmia Espanyola, vint-i-tresena edició)

Com a tal, ens ensenya un mètode -provat i estudiat- el qual es basa en les interaccions d'objectes (tot el descrit en el títol anterior, Pensar en objectes) per a resoldre les necessitats d'un sistema informàtic.

Bàsicament, aquest paradigma es compon de 4 pilars i diferents característiques que veurem a continuació .

INTRODUCCIÓ a POO AMB PHP

CARACTERÍSTIQUES CONCEPTUALS DE LA POO

Abstracció
Capacitat d'un element de tenir aïllades del seu context les seves propietats bàsiques i mètodes bàsics. Defineix les característiques essencials d'un objecte.

Encapsulació
Reuneix el mateix nivell d'abstracció, mateixa quantitat de propietats i mètodes comuns a tots els elements que puguin considerar-se pertanyents a una mateixa entitat.

Herència
És la relació existent entre dues o més classes, on una és la principal (pare) i unes altres són secundàries i depenen (hereten) d'elles (classes "filles"), on alhora, els objectes hereten les característiques dels objectes dels quals hereten.

Polimorfisme
És la capacitat que dóna a diferents objectes, la possibilitat de comptar amb mètodes, propietats i atributs d'igual nom, sense que els d'un objecte interfereixin amb el d'un altre.

INTRODUCCIÓ a POO AMB PHP

CARACTERÍSTIQUES COMUNS DE LA POO

Modularitat
Característica que permet dividir una aplicació en diverses parts més petites (denominades mòduls), independents les unes de les altres. (estructuració de funcionalitats i processos diferenciats dins de mòduls majors (MVC).

Ocultació (aïllament)
Els objectes estan aïllats de l'exterior, protegint a les seves propietats per a no ser modificades per aquells que no tinguin dret a accedir a aquestes.

Recol·lecció d'escombraries
És la tècnica que consisteix a destruir aquells objectes quan ja no són necessaris, alliberant-los de la memòria.

INTRODUCCIÓ a POO AMB PHP

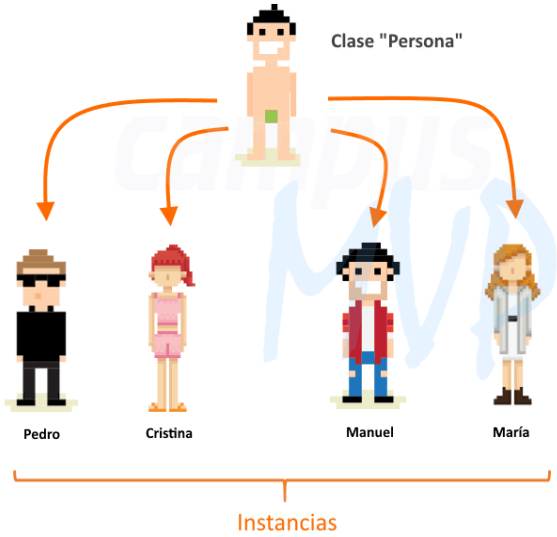
CLASSE

Una classe és un model (abstracció) que s'utilitza per a crear objectes que comparteixen un mateix comportament, estat i identitat. (encapsulació)

Persona és la metàfora d'una classe (l'abstracció de Manuel, Pedro, Cristina i María), els comportaments(mètodes) de la qual poden ser caminar, còrrer, estudiar, llegir, etc. Pot estar en estat despert, adormit, etc. Les seves característiques (propietats) poden ser el color d'ulls, color de pèl, el seu estat civil, etc. I les seves constants, cames, braços,...

```
<?php

class Persona {
#Propietats Constants
#Mètodes
}
?>
```



INTRODUCCIÓ a POO AMB PHP

```
<?php

class Persona {
#Propietats /constants
        public $nom;
        public $edat;
        public $altura;
#Mètodes
        function caminar(){
          #codi a executar
        }
}
?>
```

Propietats / atributs

Les propietats o atributs, són les variables que contenen dades associades a la class o model i per tant als objectes o representacions. les què són iguals per tota la class, es declaren com a constant.

```
class Persona {
#Propietats /constants
        public $nom;
        public $edat;
        public $altura;
```

INTRODUCCIÓ a POO AMB PHP

Mètodes

Mètodes o funcions als quals respon (comportaments), com caminar, parlar,… Indiquen la capacitat de lo que pot fer aquest model d'objecte a partir de les seves propietats o variables, i per tant els objectes o representacions de la class.

```
#Mètodes
        function caminar(){
          #còdi de la funció
        }
}
?>
```

Objecte o Instància

És una representació concreta, una entitat proveïda de propietats concretes(atributs), com nom, edat…Pedro de 27 años de edad. Proveïda de mètodes(funcions) als quals respon (comportaments), com caminar, parlar,…

```
$persona = new Persona();
#Objecte concret, o instancia a partir dem model
classPersona{ amb propietats i mètodes}
```

INTRODUCCIÓ a POO AMB PHP

CODI ESTRUCTURAT O IMPERATIU VS POO

Moltes vegades en PHP o altres llenguatges de programació es busca encapsular, a través d'una funció, lògica pròpia a un únic bloc de codi. Això se'l coneix com a codi estructurat/imperatiu.

```
#Codi Imperatiu
$automobil1 = ["marca"=>"Toyota", "model"=>"Corolla"];
$automobil2 = ["marca"=>"Hyundai", "model"=>"Accent Vision"];
function mostrar($automobil){
        echo "<p>Hola! sòc un $automobil[marca], model $automobil[model]</p>";
}
mostrar($automobil1);
mostrar($automobil2);
?>
```

Aquesta metodolodia, o paradigma té problemes. En veure el codi anterior sorgeixen dubtes:

Què passa quan es comencen a tenir moltes propietats? Podem organitzar-nos segons patrons MVC . O si es comencen a tenir moltes funcions? Patrons MVC i molta memòria.

INTRODUCCIÓ a POO AMB PHP

Què passa si volem un altre conjunt de funcions i els noms es repeteixen?
(MVC) i molta memoria, les funcions no están organitzades, i les variables han de ser creades com a globals, o locals, convertint l'organització i estructuració en una parameritzación personal . I és aquí on la POO arriba per a salvar la situació.

```
<?php
class Automobil
{
      public $marca;
      public $model;

     public function getMarca($marca)
     {
       $this->marca =$marca;
     }
     public function getModelo($modelo)
     {
       $this->modelo =$modelo;
     }
     public function setDatos()
     {
       echo "<p>Hola! soy un {$this->marca}, modelo {$this->modelo}</p>";
     }

}
```

INTRODUCCIÓ a POO AMB PHP

```
$a = new Automobil();
$a -> getMarca("Toyota");
$a -> getModel("Corolla");
$a -> setDatos();


$b = new Automobil();
$b -> getMarca("Mazda");
$b -> getModel("2");
$b -> setDatos();

#Principis de la POO què es compleixen en aquest codi:
#ABSTRACCIÓ: Nous tipus de dades (crea les dades que vols) els atributs o propietats són únics
per aquesta class, poden repetir-se a diferents classes
#ENCAPSULACIÓ: Organitzar el codi en grups lògics, ordenació en grups de propietats, mètodes....
#OCULTACIÓ: Ocultar detalls d'implementació i exposar només els detalls que siguin necessaris per a la
resta del sistema
?>
```

El mateix codi de l'exemple anterior, però en aquest cas orientat a objectes. En el codi veiem una classe anomenada Automòbil que posseeix dues propietats i tres mètodes.

INTRODUCCIÓ a POO AMB PHP

De moment és interessant observar com en només un bloc de codi agrupat entre les paraules claus class Automobil{…} Tenim la informació que defineix i concreta un objecte “Automòbil”.

En crear la instància, és a dir, realitzar còpies d'un automòbil, veiem com la POO ens fa la vida fàcil. En el codi anterior $a i $b són còpies diferents, permetent així que $a tingui una propietat ‘model’ que és pròpia i diferent de la propietat ‘model’ de la variable $b.

CODI BÀSIC AMB POO

Com ja s'ha reflectit abans, tota classe consta de la paraula clau class seguit del nom de la classe i un bloc de codi entre claus.

Dins del bloc de codi es poden crear tres tipus de blocs bàsics:

- Constants
- Variables
- Mètodes

Una vegada creades dins de les claus, tant la constant, com la variable, com la funció pertanyen a la classe, i per a ser utilitzades cal accedir a través de la classe.

INTRODUCCIÓ a POO AMB PHP

```
<?php
class blocs
{
//declaració de constants com a propietat
const NAMEMYCONST =“valor”;
// declaració var com a propietat (pública)
public $myvar =“valor inicial”;
// declaració mètodes
public function nameMetode(){
echo $this->$myvar;}
}
?>
```

Recorda …

Una analogia apropiada (però molt bàsica) al moment de pensar en una classe és pensar en un motlle del qual s'extrauran múltiples “còpies” o “objectes” similars. A diferència d'un objecte físic, en aquest cas les còpies seran dinàmiques i poden canviar el seu comportament i estructura al moment d'executar un programa.

En el nostre cas ‘Automovil’ és la classe i la variable $a és un objecte (instància o còpia personalitzada) sobre la classe Automòbil. La paraula clau new permet crear un nou objecte Automòbil al costat de totes les seves propietats i funcions s'emmagatzemin en $a.

Com els arrays, un objecte o còpia o instancia, es una variable més, una variable complexa. Un objecte, és una variable global i per tant accesible des de qualsevol lloc de l’aplicació

INTRODUCCIÓ a POO AMB PHP

Es poden fer totes les còpies d'una classe que es vulguin.

Una instància o objecte sol tenir el mateix nom que la class de la qual neix

Segons el Manual Oficial de PHP, per a definir una classe s'han de complir amb els següents passos:

```
<?php

class Automobil
{
   public $marca;
   public $model;

   public function mostrar(){
     echo "<p>Hola! sóc un $this->marca, model $this->model</p>";
   }
}
$a = new Automobil();
$a -> marca = "Toyota";
$a -> model = "Corolla";
$a -> mostrar();
```

[…] “La definició bàsica de classes comença amb la paraula clau class, seguit per un nom de classe, continuat per un parell de claus que tanquen les definicions de les propietats i mètodes pertanyents a la classe. El nom de classe pot ser qualsevol etiqueta vàlida que no sigui una paraula reservada de PHP. Un nom vàlid de classe comença amb una lletra o un guió baix, seguit de la quantitat de lletres, números o guions baixos que sigui.” […]

INTRODUCCIÓ a POO AMB PHP

REGLES D'ESTIL SUGGERIDES

Utilitzar CamelCase per al nom de les classes. La clau d'obertura en la mateixa línia que el nom de la classe, permet una millor llegibilitat del codi.

Herència de Classes

Els objectes poden heretar propietats i mètodes d'altres objectes. Per a això, PHP permet la "extensió" (herència) de classes, la característica de les quals representa la relació existent entre diferents objectes. Per a definir una classe com a extensió d'una classe "pare" s'utilitza la paraula clau extends.

```
<?php
class classePare {
#codi
}
Class classFilla extends classPare {
/* Aquesta class hereta totes les propietats i mètodes del pare o superclasse */
}
```

INTRODUCCIÓ a POO AMB PHP

Declaració de classes abstractes

Les classes abstractes són aquelles que no necessiten ser instanciadas (No tinc necessitat de crear una variable que contingui una representació d'ella o instància) però no obstant això, seran heretades en algun moment. Es defineixen anteposant la paraula clau abstract:

```
<?php
abstract class classeAbstracta {
#codi
}
```

Aquest tipus de classes, serà la que contingui mètodes abstractes i generalment, la seva finalitat, és la de declarar classes "genèriques" que necessiten ser declarades però a les quals, no es pot atorgar una definició precisa (No es poden instanciar), d'això, s'encarregaran les classes que l'heretin).

Un exemple vist amb més profunditat:

Es crea una classe abstracta anomenada servei la qual contindrà 2 mètodes:

```
<?php
abstract class servei {
        abstract public function saludo($nom);#no té codi
        public function carta($nom,$esmorzar){
                echo $this->saludo($nom);
                echo "Li puc oferir: <br/>";
        foreach($esmorçar as $key =>$value){
                echo $key.": ".$value."<br/>";
                }
        }
}
```

INTRODUCCIÓ a POO AMB PHP

- carta($nom, $esmorzar) la qual rebrà el nom de la persona i l'envia al mètode abstracte saludo, el qual s'ha de sobreescriure en la classe que l'hereti. També rebrà un array, el qual es recorrerà i mostrarà la seva clau i valor, representant a una carta de restaurant.

- Es crea una funció abstracta saluddo($nom) la qual rep el nom de la persona i haurà de ser obligatòriament sobreescrita en la classe que estengui de servei.

El que realitzarem ara, serà una classe que hereti els mètodes de la classe abstracta i sobreescrigui el mètode de saludar.

```
<?php
class cambrer extends servei {
        public function saludo($nom){
                return “Bona tarda: $nom <br/>; }
}
$cambrer1 = new cambrer();
$nom=“Pepito Pérez”;
$esmorzar = [ “Primer”    =>  ”Sopa de verdures”,
                “Segon” =>  ”Combinat de carns brassa”,
                “Postre” =>  ”Suc de Poma” ];
$cambrer1-> carta($nom,$esmorzar);
```

INTRODUCCIÓ a POO AMB PHP

Com es pot visualitzar en l'anterior codi s'estén de la classe servei i sobreescriu la funció saludo. A més es realitzen les proves al instanciar l'objecte cambrer.

Declaració de Classes finals En PHP

PHP des de la versió 5.1 incorpora classes finals que no poden ser heretades per una altra. Es defineixen anteposant la paraula clau final.

```
<?php
final class finalClass{
        // class que no pot ser heretada per cap altra. No és obligatori fer el final per no fer fills
}
?>
```

Això passaria si es fa una herència d'una classe final

INTRODUCCIÓ a POO AMB PHP

```
<?php
final class Automobil{
        // class que no pot ser heretada per cap altra. No és obligatori fer el final per no fer fills
}
//Fatal error:
//class Camio may not inherit from final class…
class Camio extends Automobil{
}
?>
```

QUIN TIPUS DE CLASSE DECLARAR?

Fins aquí, hem definit quatre tipus de classes diferents:

classes instanciables: instanciable i/o heretable -> class que té relació en propietats i mètodes amb altra class.
classes abstractes: class què només serveix de model per altra class pare sense poder ser instanciada.
classes hereues i finals: class que pot ser instanciada però no pot tenir clases filles (ser heretada)

INTRODUCCIÓ a POO AMB PHP

OBJECTES I INSTÀNCIES

Una vegada que les classes han estat declarades, serà necessari crear els objectes i utilitzar-los, encara que hem vist que algunes classes, com les classes abstractes són sol models per a unes altres, i per tant no necessiten instanciar cap objecte.

Instanciar una classe

Per a instanciar una classe, només és necessari utilitzar la paraula clau new. L'objecte serà creat, assignant aquesta instància a una variable (la qual, adoptarà la forma d'objecte).
Lògicament, la classe ha d'haver estat declarada abans de ser instanciada, com es mostra a continuació :

```
<?php
class Persona{
        // codi
}
$persona1 = new Persona();
?>
```

INTRODUCCIÓ a POO AMB PHP

Regles per crear instàncies de les classes

Per a una millor llegibilitat i ús de les classes, es recomana utilitzar noms de variables (objectes) descriptius, sempre, la primera lletra ha de ser en minúscula, i la següent paraula en majúscula. Per exemple si el nom de la classe és nombreClase com a variable
utilitzar $nombreClase. Això permetrà una major llegibilitat del codi.

Definició d'atributs o propietats en PHP

Les propietats representen certes característiques de l'objecte en si mateix, poden gaudir de diferents característiques, com per exemple, la visibilitat: poden ser públiques, privades o protegides. La visibilitat de les propietats, és aplicable també a la visibilitat dels mètodes.

Nivells d'accés

1.Propietats públiques

Les propietats públiques es defineixen anteposant
la paraula clau public al nom de la variable.
Aquestes, poden ser accedides des de qualsevol part de l'aplicació, sense restricció.

```
<?php
class Persona{
        public $nom;
        public $genere;
}
```

INTRODUCCIÓ a POO AMB PHP

2.Propietats privades

Les propietats privades es defineixen anteposant la paraula clau private al nom de la variable.

Aquestes només poden ser accedides per la classe que les va definir.

```
<?php
class Persona{
        public $nom;
        public $genere;
        private $edat;
}
```

3.Propietats protegides

Les propietats protegides poden ser accedides per la pròpia classe que la va definir, així com per les classes que l'hereten,
però no, des d'altres parts de l'aplicació.

Aquestes, es defineixen anteposant la paraula clau protected al nom de la variable:

```
<?php
class Persona{
        public      $nom;
        public      $genere;
        private     $edat;
        protected $passaport;
}
```

INTRODUCCIÓ a POO AMB PHP

4.Propietats estàtiques

Les propietats estàtiques representen una característica de “variabilitat” de les seves dades, de gran importància en PHP. Una propietat declarada com a estàtica, pot ser
accedida sense necessitat de instanciar un objecte i el seu valor és estàtic (és a dir, no pot ser modificada per a cada objecte, és com una variable global per a totes les instàncies que es creen d'aquest objecte, o com una constant).

Aquesta, es defineix anteposant la paraula clau static al nom de la variable:

```
<?php
class Persona{
        public      $nom= “Pepito”;
        public      $genere;
        public static $tipusSang=“A+”;
}
//no tenim per que crear l’objecte per accedir al tipusSang
echo Persona::$tipusSang.”<br/>”;
//El modifiquem y cambia per tots els elements Persona
Persona::$tipusSang=“O+”;
echo Persona::$tipusSang.”<br/>”;
```

INTRODUCCIÓ a POO AMB PHP

```
//una vegada creat l'objecte, podré accedirals diferents atributs, i mètodes
$p = new Persona();
$p->nom ="Felipe";
echo $p->nom."<br/>";
//------altre
$p = new Persona();
$p->nom ="Ana";
echo $p->nom."<br/>";
```

ACCEDINT A les PROPIETATS D'UN OBJECTE:

Per a accedir a la propietat d'un objecte, existeixen diverses maneres de fer-lo. Totes elles, dependran de l'àmbit des del qual les hi invoqui així com de la seva condició i visibilitat.

Accés a variables des de l'àmbit de la classe

S'accedeix a una propietat no estàtica dins de la classe, utilitzant la pseudo-variable $this sent aquesta pseudo-variable una referència a l'objecte mateix, s'ha de tenir en compte que la variable que es cridés no portés endavant el $:

INTRODUCCIÓ a POO AMB PHP

```
<?php
class Persona{
        public      $nom = “Pepito”;  //per defecte
        public      $cognom;
        public static $tipusSang=“A+”;

        public function hola()
        {
        echo $this->nom.”<br/>”;
        echo self::$tipusSang.”<br/>”;
        }
}
```

Quan la variable és estàtica, s'accedeix a ella mitjançant l'operador de resolució d'àmbit, doble dos- punts :: anteposant la paraula clau self o parent segons si tracta d'una variable de la mateixa classe o d'una altra de la qual s'ha heretat, respectivament:

INTRODUCCIÓ a POO AMB PHP

Accés a variables des de l'exterior de la classe

S'accedeix a una propietat no estàtica amb la següent sintaxi:

$objecte->variable

```
$p2 = new Persona();
$p2->nombre=“Ana”;
echo $p2->nombre=“<br/>”;
```

Tenint en compte que aquest accés dependrà de la visibilitat de la variable. Per tant, només variables (propietats) públiques poden ser accedides des de qualsevol àmbit fora de la classe o classes heretades

Per a accedir a una propietat pública estàtica l'objecte no necessita ser instanciado, permetent així, l'accés a aquesta variable mitjançant la següent sintaxi:

Classe::$variable_estàtica
echo Persona::$tipusSang.”
”;

CONSTANTS:

Un altre tipus de “propietat” d'una classe, són les constants, aquelles que mantenen el seu valor de manera permanent i sense canvis. A diferència de les propietats estàtiques que poden ser declarades dins d'una classe, les constants només poden tenir una

INTRODUCCIÓ a POO AMB PHP

visibilitat pública i no han de ser creades dins de les classes. L'accés a constants és exactament igual que al d'altres propietats.

```
<?php
//valors no modificables
define (MISSATGE, “<h1> Benvingut al Curs</h1>”);
echo MISSATGE;
const MYCONST =“Es un valor estàtic”;
echo MYCONST
```

```
<?php
class Persona{
public    $nom = “Pepito”;  //per defecte
public    $cognom;

public function donar_sang($nom, $cognom){
                        #code
                        }
}
```

MÈTODES PHP

Cal recordar, per als qui vénen de la programació estructurada, que el mètode d'una classe, és un algorisme igual al d'una funció.

La manera de declarar un mètode és anteposant la paraula clau function al nom del mètode, seguit per un parèntesi d'obertura i tancament i claus que tanquin l'algorisme:

Poden ser amb i sense paràmetres

INTRODUCCIÓ a POO AMB PHP

Mètodes públics, privats, protegits i estàtics

Els mètodes, igual que les propietats, poden ser públics, privats, protegits o estàtics. La manera de declarar la seva visibilitat tant com les característiques d'aquesta, és exactament la mateixa que per a les propietats.

Els mètodes abstractes per contra, només poden ser creats en les classes la declaració de les quals hagi estat abstracta. En cas contrari mostrarà un error en executar el codi.

```
<?php
class Persona{
public    $nom = “Pepito”;  //per defecte
public    $cognom;

            public function donar_sang($tipusSang){
             #code
             }
             public static function a(){  }
             protected function b(){  }
             private function c(){  }
}
```

INTRODUCCIÓ a POO AMB PHP

MÈTODES ABSTRACTES

A diferència de les propietats, els mètodes, poden ser abstractes com succeeix amb les classes.
Per a entendre millor els mètodes abstractes, podríem dir que a grans trets, els mètodes abstractes són aquells que es declaren inicialment en una classe abstracta, sense especificar l'algorisme que implementaran, és a dir, que només són declarats però no contenen un "codi" que especifiqui què faran i com el faran.

Classe pare

```
<?php
abstract class servei {
        abstract public function saludo($nom);#no té codi
        public function carta($nom,$esmorzar){
                echo $this->saludo($nom);
                echo "Li puc oferir: <br/>";
        foreach($esmorçar as $key =>$value){
                echo $key.": ".$value."<br/>";
                }
        }
}
```

Classe filla

```
<?php
class cambrer extends servei {
        public function saludo($nom){
                return "Bona tarda: $nom <br/>"; }
}
```

INTRODUCCIÓ a POO AMB PHP

Com es pot visualitzar la classe filla deu obligatòriament sobreescriure el mètode saludar ja que va ser declarat com a abstracte en la classe pare.

Mètodes màgics en PHP

PHP, ens porta una gran quantitat de mètodes denominats “mètodes màgics”.
Aquests mètodes, atorguen una funcionalitat pre-definida per PHP, que poden aportar valor a les nostres classes i estalviar-nos grans quantitats de codi.
El que moltsprogramadors considerem, ajuda a convertir a PHP en un llenguatge orientat a objectes, cada vegada més robust.

Entre els mètodes màgics, podem trobar els següents:

El Mètode Màgic construct()

```
<?php
class Automobil
{
public $marca;
public $model;
public function __construct($marca,$model){
//método mágico para inicializar classes
            $this->marca=$marca;
            $this->model=$model;
}
public function mostrar(){
            echo "<p>Hola! sóc un $this->marca, model $this-
>model</p>";
            }
}
$a = new Automobil("Toyota","Corolla");
$a -> mostrar();
```

INTRODUCCIÓ a POO AMB PHP

En l'exemple anterior, el constructor de la classe s'encarrega de definir i inicialitzar el producte. El mètode construct() serà invocat de manera automàtica, al instanciar un objecte. La seva funció és la d'executar qualsevol inicialització que l'objecte necessiti abans de ser utilitzat.

El mètode màgic destruct()

El mètode destruct() és l'encarregat d'alliberar de la memòria, a l'objecte quan ja no és referenciat. Es pot aprofitar aquest mètode, per a fer altres tasques que s'estimin necessàries al moment de destruir un objecte.

```
public function __destruct(){
//método mágico paradestruir classes
        $this->marca;
        $this->model;
}
```

Altres mètodes màgics

PHP ens ofereix altres mètodes màgics com ara __call, __callStatic, __get, __set, __isset, __unset, __sleep, __wakeup, __toString, __invoke, __set,__state i __ clone.

INTRODUCCIÓ a POO AMB PHP

Interfícies

Una interfície és un conjunt de mètodes abstractes i de constants la funcionalitat de les quals és la de determinar el funcionament d'una classe, és a dir, funciona com un motlle o com una plantilla. A l'ésser els seus mètodes abstractes aquests no té cap funcionalitat, només es defineixen el seu tipus, argument i tipus de retorn.

Per a implementar una interface és necessari que la classe que vulgui fer ús dels seus mètodes utilitzi la paraula reservada implements.
La classe que la implementi, d'igual manera ha de sobreescriure els mètodes i afegir la seva funcionalitat.

La construcció d'una Interfície és la següent:

```
<?php
Interface Filtro {
public function filtrar($emements)
{ }
class filtroVocals implements Filtro{
        public function filtrar($cadena){
        return preg_match("/[aeiuo]/","",$cadena);
        }
}
 $a = new filtroVocals();
$a -> filtrar("Australopithecus");
Echo $a;
```