CUPRINS

Programare orientata pe obiecte (POO) in PHP

Evolutiile spectaculoase din domeniul informaticii au impus realizarea unui nou model de programare capabil sa depaseasca limitarile programarii structurate si care sa fie capabil sa realizeze abstractizarea adecvata a datelor. Astfel s-a format clasa limbajelor bazate pe obiecte si apoi a celor orientate pe obiecte.

În programarea orientată-obiect un sistem informatic este privit ca un model fizic de simulare a comportamentului unei părţi din lumea reală sau conceptuală. Acest model fizic este definit prin intermediul unui limbaj de programare şi el se concretizează într-o aplicaţie ce poate fi executată pe un sistem de calcul.

Printre avantajele programarii pe obiecte se numara:

- cod mai structurat şi mai lizibil
- lucrul organizat
- depanarea mai usoara a programelor
- refolosirea codului

Dezavantajele ar fi:

- ruleaza mai incet
- timpii de dezvoltare sunt mai mari

In PHP 4 nu sunt implementate toate facilităţile POO. Pentru programarea pe obiecte in PHP, este recomandata utilizarea PHP5.

Clase

O clasa este o colectie de variabile si functii care opereaza asupra variabilelor respective. Implementarea unei clase contine atat variabile, cat si functii, ea reprezentand un sablon (template) cu ajutorul caruia pot fi create instante specifice.

De exemplu: Clasa "floare" desemneaza toate plantele care au flori

Liniile care încep cu var (variabile) sunt proprietăţile clasei. In PHP 5, proprietatile pot fi declarate ca publice, private sau protejate, fara a mai scrie *var* in fata lor. In PHP nu se specifica proprietatea public

Sintaxa folosita pentru declararea unei clase in PHP este:

```
class nume_clasa {
// date membre
var nume_variabila_1
...
var nume_variabila_m
// metode
function nume_functie_1 (parametri) {
... // definitia functiei
```

```
}...
function nume_functie_n (parametri) {
... // definirea functiei
}
}
```

Pentru numele unei clase poate fi utilizat orice identificator permis in PHP cu o singura exceptie: sdtclass; acest identificator este folosit de PHP in scopuri interne. Pentru a initializa variabilele cu valori care nu sunt constante trebuie folosit un constructor. Mai jos aveti exemple de clasa in care initializarile *nu sunt corecte*:

```
class Nepermis {
var $data = date ("Y-m-d");
var $nume = $prenume;
var $dest = 'Mihai '. 'Claudiu';
var $obiecte = array ("minge", "pantof");
```

Aceste cuvinte cheie sunt modificatori de acces la metodele si proprietãtile unor clase. Dacã în cadrul declaratiei unei clase, o metodã sau proprietate este precedatã de cuvântul cheie *public*, atunci acea metodã va putea fi accesatã din exteriorul declaratiei clasei.

Dacã în cadrul declaratiei unei clase, o metodã sau proprietate este precedatã de cuvântul cheie *protected*, atunci acea metodã va putea fi accesatã doar în cadrul declaratiei clasei curente si a claselor derivate din aceasta.

Dacã în cadrul declaratiei unei clase, o metodã sau proprietate este precedatã de cuvântul cheie *private*, atunci acea metodã nu va putea fi accesatã din exteriorul declaratiei clasei curente. Variabilele membre ale clasei pot fi apelate in exteriorul clasei doar prin intermediul functiilor create in clasa sau in clasa de baza, in cazul claselor derivate, cu conditia ca aceste functii sa fie *public*

Dacã pentru un membru al unei clase nu este specificat nici un modificator de acces, acel membru va avea implicit modificatorul public.

Se obisnuieste ca variabilele sa fie declarate *private* pentru restrictionarea accesului la ele, iar functiile se declara *public* pentru a putea fi accesate din exteriorul clasei, implicit pentru a avea efect asupra variabilei clasei.

Exemplu:

```
<?php
class Aritmetica {
public $x = 2;
public $y = 3;
public function Suma ( ) {
return $this -> x + $this -> y;
}
}
$a = new Aritmetica;
echo "Suma nr: ";
echo $a->x+$a->y;
```

```
?>
```

S-a declarat obiectul $a de tipul clasei si deoarece variabilele sunt declarate public s-a putut opera direct cu acestea, prin intermediul obiectului.

In exemplul urmator, variabilele clasei sunt declarate private, iar singurul mod prin care putem opera cu acestia este prin intermediul functiei Suma

Daca in loc de *echo $a -> Suma ( ) . "
";* am fi scris *echo $a->x+$a->y;* ar fi aparut urmatoarea eroare:

Suma nr:

Fatal error: Cannot access private property Aritmetica::$x in **C:\Program Files\EasyPHP 2.0b1\www\modul 5\test8.php** on line **12**

Exemplu

```
<?php
class Aritmetica {
private $x = 2;
private $y = 3;
public function Suma ( ) {
return $this -> x + $this -> y;
}
}
$a = new Aritmetica;
echo "Suma nr: ";
echo $a -> Suma ( ) . " <br>";
?>
```

Obiecte

Un obiect reprezinta o variabila de tipul clasei. Fiecare obiect are o serie de caracteristici, sau ***proprietati***, si anumite functii predefinite – ***metode***. Aceste proprietati si metode ale unui obiect corespund variabilelor si functiilor din definitia clasei.

Operatia de initializare a unui obiect se numeste instantiere. Clasele pot fi folosite pentru a genera instante multiple in memorie

Instantierea(trecerea de la clasa la obiect) inseamna atribuirea unor proprietati specifice clasei, astfel incat aceasta sa indice un obiect anume, care se diferentiaza de toate celelalte obiecte din clasa printr-o serie de atribute.

Daca vom considera ca "fruct_exotic" care desemneaza clasa tuturor fructelor exotice contine proprietatea "culoare" atunci atribuind acesteia valoarea "galben" noi vom crea o noua multime(clasa fructelor exotice care au culoarea galbena) care este o subclasa a clasei "fruct_exotic", deci realizam astfel o particularizare. Mai mult decat

atat, daca vom adauga noi si noi atribute vom individualiza clasa astfel incat sa ajungem la un caz concret, care este Obiectul.

Proprietati

In corpul unei clase, poti declara variabile speciale, numite proprietati. In PHP 4 acestea trebuiau declarate cu cuvantul cheie *var* in fata

```
/**
* PHP versiunea 4
*/

class Dictionar {
  var $traduceri = array();
  var $tip = “Ro”;
}
```

Aceasta sintaxa este in continuare acceptata (in PHP 5), dar doar pentru a asigura compatibilitate cu versiuni anterioare de PHP.

In PHP 5, codul arata astfel:

```
/**
 * PHP versiunea 5
 */

class Dictionar {
  public $traduceri = array();
  public $tip = “Ro”;
}
```

Poti accesa proprietatile obiectelor folosind operatorul "->". Ca atare, "$ro->tip" inseamna proprietate $tip din obiectul Dictionar referentiat de "$ro".

Metode

Metodele sunt functii cu ajutorul carora se poate opera asupra variabilelor clasei. Intr-o forma simpla, metodele sunt functii declarate in interiorul unei clase. Sunt de obicei - dar nu intotdeauna - apelat prin intermediul unei instante de obiect folosind operatorul "->"

$this este o pseudo variabila ce retine adresa obiectului curent (referinta catre obiectul curent)

In continuare vom defini o clasa Aritmetica cu doua date membre x si y care sunt numere intregi si doua metode care realizeaza adunarea, respectiv inmultirea lor.

```
class Aritmetica {
var $x = 2;
```

```
var $y = 3;
function Suma ( ) {
return $this -> x + $this -> y;
}
function Produs ( ) {
return $this -> x * $this -> y;
}
}
```

Pentru a crea un obiect de tipul Aritmetica vom utiliza o instructiune de tipul:

```
$aritm = new Aritmetica;
```

Acum putem utiliza metodele clasei; pentru a afisa suma sau produsul celor doua numere vom putea apela cele doua metode astfel:

```
echo "suma este ".$aritm -> Suma ( )."<br>";
echo "produsul este ".$aritm -> Produs ( );
```

Vom obtine rezultatele suma este 5 produsul este 6. Valorile datelor membre pot fi si ele modificate prin instructiuni de tipul:

```
$aritm -> x = 5
$aritm -> y = 4;
```

Daca, in urma modificarii apelam din nou metodele Suma ( ) si Produs ( ), rezultatele vor fi 9, respectiv 20.

In acest exemplu a fost utilizata pseudo-variabila $this. Aceasta este folosita pentru a indica faptul ca se opereaza asupra unei date membre a obiectului curent.

Pentru a intelege mai bine cele de mai sus, luam ca exemplu un animal, cum ar fi un urs, si incercam sa il reprezentam ca un obiect.

Orice urs are anumite caracteristici – varsta, greutate, sex – care sunt echivalente cu proprietatile unui obiect. In plus, fiecare urs are anumite activitati – mananca, doarme, merge, alearga si hiberneaza – acestea reprezentand metodele unui obiect.

Deoarece toti ursii au in comun anumite caracteristici, putem crea un template Urs(), care defineste caracteristicile de baza si abilitatile fiecarui urs de pe planeta. Aceasta clasa Urs() poate fi utilizata pentru a crea un obiect $urs, proprietatile individuale ale unui Urs putand fi manipulate independent fata de cele ale altor obiecte de acelasi tip.

In PHP 5, clasa Urs ar arata in felul urmator:

```
<?php
// PHP 5
// definitia clasei
class Urs {
```

```
    // definitia proprietatilor
    public $nume;
    public $greutate;
    public $varsta;
    public $sex;
    public $culoare;
    // definitia metodelor
    public function mananca() {
        echo $this->nume." mananca... ";
    }
    public function alearga() {
        echo $this->nume." alearga... ";
    }
    public function vaneaza() {
        echo $this->nume." vaneaza... ";
    }

    public function doarme() {
        echo $this->nume." doarme... ";
    }
}
?>
```

Avand aceasta clasa, putem crea oricati ursi vrem:

```
<?php
// primul urs
$daddy = new Urs;
// sa-i dam un nume
$daddy->nume = "Tata urs";
//  cati ani are
$daddy->varsta = 8;
// ce sex este
$daddy->sex = "mascul";
// culoarea blanii
$daddy->culoare = "negru";
// cat cantareste
$daddy->greutate = 300;

// cel de-al doilea urs
$mommy = new Urs;
$mommy->nume = "Mama urs";
$mommy->varsta = 7;
$mommy->sex = "femela";
$mommy->culoare = "negru";
$mommy->greutate = 310;

// cel de-al treilea urs
$baby = new Urs;
$baby->nume = "Puiul urs";
$baby->varsta = 1;
$baby->sex = "mascul";
$baby->culoare = "negru";
$baby->greutate = 180;

// o seara placuta pentru familia Urs
```

```
// ursul tata vaneaza si aduce prada acasa
$daddy->vaneaza();

// ursul mama mananca
$mommy->mananca();
// la fel si puiul
$baby->mananca();

// ursul mama doarme
$mommy->doarme();
// la fel si tatal
$daddy-> doarme ();

// puiul mananca in continuare
$baby->mananca();
?>
```

Dupa cum se poate observa din exemplul de mai sus, atunci cand sunt definite noi obiecte, metodele si proprietatile lor pot fi accesate in mod independent pentru fiecare obiect.

Constructori

Un constructor este o metoda (functie) a unei clase care este apelata automat in momentul in care este creata o noua instanta a clasei (cu ajutorul operatorului **new**). In PHP este considerata ca fiind un constructor orice functie care are acelasi nume cu clasa in interiorul careia este definita. Constructorii pot fi folositi pentru initializarea datelor membre cu valori care nu sunt constante. Ei pot avea argumente, iar acestea pot fi optionale. Pentru a putea utiliza clasa fara a specifica nici un parametru in momentul crearii unui obiect, se recomanda stabilirea unor valori implicite pentru toate argumentele constructorului. In cazul in care nu este definit un constructor pentru o anumita clasa, se utilizeaza constructorul clasei de baza, daca aceasta exista. De exemplu, pentru urmatoarea secventa de cod, in momentul crearii obiectului corespunzator variabilei $b, va fi apelat constructorul clasei A.

```
<?php
class Oameni{
public $varsta;
public$ocupatie;
public$insurat;
public $iq;
public $nume;
function Oameni($nume, $varsta, $ocupatie, $insurat, $iq){ //definirea constructorului
$this->nume=$nume;
$this->varsta=$varsta;
$this->ocupatie=$ocupatie;
$this->insurat=$insurat;
$this->iq=$iq;
```

```
}
function spuneSalut(){
echo "Buna, ma numesc ".$this->nume.", am ".$this->varsta." de ani, ".($this->insurat?"sunt":"nu sunt")." insurat, sunt ".$this->ocupatie." si am IQ ".$this->iq;
}
}
$gigi=new Oameni("Gigi",20,"maturator",false,50);//apelul constructorului
$gigi->spuneSalut();
?>
```

In PHP apelul constructorului clasei de baza trebuie sa fie explicit daca este necesara executarea operatiilor corespunzatoare. In majoritatea limbajelor de programare exista functii speciale numite destructori care sunt apelate automat in momentul "distrugerii" unui obiect. In PHP nu exista destructori.

```
<?php
class Student {
// date-membru
public $year; // an
public $age; // vârsta
public $name; // nume
// constructor
function Student($y, $a, $n) {
$this->year = $y;
$this->age = $a;
$this->name = $n;
}
// metode
function setYear($y) {
$this->year = $y;
}
function getYear() {
return $this->year;
}
function setAge($a) {
$this->age = $a;
}
function getAge() {
return $this->age;
}
function setName($n) {
$this->name = $n;
}
function getName() {
return $this->name;
}
}
```

```
$st = new Student("2008","20","Gigel");

echo "Primul an este ".$st->getYear()."<br>";
$st->setYear("1950");
echo "Al doilea an este ".$st->getYear()."<br>";


echo "Primul nume este ".$st->getName()."<br>";
$st->setName("Ionel");
echo "Al doilea nume este ".$st->getName()."<br>";


echo "Prima varsta este ".$st->getAge()."<br>";
$st->setAge("30");
echo "A doua varsta este ".$st->getAge()."<br>";

?>
```

In exemplul de mai sus, avem doua tipuri de functii membre ale clasei:

- Functii de tip *set* – Permit atribuirea de valori variabilelor membre din calsa
- Functii de tip g*et* – Permit accesul la valorile variabilelor membre din clasa

Constructorii si metodele, fiind functii PHP obisnuite, pot avea specificate valori implicite pentru argumente (ca în C++):

```
function Student($y = "4", $a = "22", $n = "")
```

Dacã scriem în acest mod constructorul, atunci în urmãtoarele cazuri vom avea:

```
$stud = new Student();
// year = 4, age = 22, name = ""
```

```
$stud = new Student(2);
// year = 2, age = 22, name = ""
```

```
$stud = new Student(2, 20);
// year = 2, age = 20, name = ""
```

```
$stud = new Student(5, 30,”Gigel”);
// year = 5, age = 30, name = "Gigel"
```

Dacã în variante mai vechi ale PHP, constructorii puteau avea orice tip de parametri, începând cu versiunea 4, tipurile permise pentru parametrii unui constructor

sunt doar cele simple (întregi, siruri de caractere), deci nu vor putea fi executate transmiteri de tablouri sau de obiecte.

În PHP în momentul construirii unui obiect, nu este apelat constructorul clasei pãrinte, acesta trebuind apelat, dacã este cazul, folosind instructiunea:

parent::__construct().

Destructori

În cadrul claselor definite în limbajul PHP 5 pot fi declarati destructori de clase. Destructorii sunt utilizati în momentul distrugerii (eliberãrii memoriei alocate) unui obiect. Un obiect este distrus (eliminat din memorie) în momentul în care nu mai existã referinte cãtre el. Necesitatea destructorilor a apãrut în momentul în care s-a pus problema transmiterii obiectelor prin referintã si nu prin valoare cum se efectua transmiterea obiectelor în versiunile anterioare ale interpretorului.

Un destructor este dat de o metodã al cãrui nume este

__destruct()

si care nu are parametri.

Ca si în cazul constructorilor, în momentul apelului unui destructor pentru o clasã nu este apelat automat destructorul pãrintelui, acesta trebuind apelat folosind instructiunea:

parent::__destruct().

Exemplu pentru utilizarea constructorilor so destructorilor

```
<?php
class Linie{
private $capat1;
private $capat2;


function __construct($p1, $p2){
$this->capat1 = $p1;
echo "am construit capatul 1<br>";
$this->capat2 = $p2;
echo "am construit capatul 2<br>";
}


function __destruct(){
$this->capat1 = null;
echo "am distrus capatul 1<br>";
$this->capat2 = null;
echo "am distrus capatul 2<br>";
}
```

```
}

$x=new Linie(10,20);
?>
```

Extinderea claselor

Deseori este necesara definirea unor clase cu proprietati (date membre) si metode asemanatoare. Pentru a usura definirea unor astfel de clase a fost introdus conceptul de **extindere** (derivare) a claselor. O clasa derivata va pastra toate proprietatile si metodele clasei pe care o extinde si poate contine diferite proprietati si metode noi. Nu exista nici o posibilitate de a elimina din clasa derivata anumite proprietati sau metode ale clasei de baza.

Dacã o clasã derivatã nu posedã propriul ei constructor, va fi apelat implicit constructorul clasei pãrinte. Atunci când un obiect al unei clase derivate este creat, numai constructorul lui propriu va fi apelat, constructorul clasei de bazã nefiind apelat implicit. Dacã dorim ca si constructorul clasei pãrinte sã fie apelat, o vom face într-o manierã explicitã.

```
class A {
function A ( ) {
echo "Constructorul clasei A";
}
function B ( ) {
echo "O functie obisnuita a clasei A.";
}
}
class B extends A {
function C ( ) {
echo "O functie obisnuita a clasei B.
";
}
}
$b = new B;
```

Pentru a extinde o anumita clasa se utilizeaza cuvantul cheie **extends**. In urmatorul exemplu vom extinde clasa *Aritmetica*; vom adauga inca o variabila si vom crea doua noi functii: una pentru calculul sumei celor trei variabile si una pentru calcularea produsului lor:

Daca definim un obiect prin intermediul unei instructiuni de genul:

$a = new Aritmetica3;

atunci pentru acest obiect vom putea utiliza atat metodele definite in cadrul clasei *Aritmetica3: Suma3 ( )* si *Produs3 ( ),* cat si metodele definite in cadrul clasei de baza

Aritmetica: Suma ( ) si *Produs ( ).* In continuare aveti un exemplu care ilustreaza modul in care pot fi create si utilizate clasele derivate.

```
<?php

class Aritmetica {
public $x = 2;
public $y = 3;
function Suma ( ) {
return $this -> x + $this -> y;
}
function Produs ( ) {
return $this -> x * $this -> y;
}
}
class Aritmetica3 extends Aritmetica
{
var $z = 4;

function Suma3 ( ) {
return $this -> x + $this -> y + $this -> z;
}
function Produs3 ( ) {
return $this -> x * $this -> y * $this -> z;
}
}
$a = new Aritmetica3;
echo "Suma primelor doua nr (functie din clasa de baza): ";
echo $a -> Suma ( ) . " <br>";
echo "Produsul primelor doua nr (functie din clasa de baza): ";
echo $a -> Produs ( ) . "<br>";

echo "Suma celor trei numere: (functie din clasa extinsa)";
echo $a ->Suma3 ( ) . "<br>";
echo "Produsul celor trei numere: (functie din clasa extinsa)";
echo $a -> Produs3 ( ) . "<br>";

?>
```

Mostenirea

Mostenirea reprezintã posibilitatea folosirii datelor sau metodelor definite în prealabil de o anumitã clasã în cadrul unei clase derivate din prima.

Relatia de derivare se specificã prin cuvântul-cheie extends:

Exemplu:

```
<?php

class Student{
public $age; // varsta
        function setAge($age){
                $this->age=$age;
                        }
        function getAge(){
                return $this->age;
                        }
}
class GoodStudent extends Student {
// date-membru
public $prizes; // premii
// metode
function setPrizes($p) {
$this->prizes = $p;
}
function getPrizes() {
return $this->prizes;
}
}
$goodstud = new GoodStudent;
// apel de metodã din clasa de bazã
$goodstud->setAge(21);
// apel de metodã din clasa derivatã
$goodstud->setPrizes(2);
echo "Studentul cu varsta: ".$goodstud->getAge()." a castigat premiul: ".$goodstud->getPrizes();
?>
```

In PHP clasele trebuie definite inaintea utilizarii lor; asadar clasa parinte va fi definita intotdeauna inaintea clasei **fiu**.

Pentru a ilustra acest concept, sa consideram clasa **UrsPolar()** care mosteneste clasa **Urs()** si defineste o noua metoda:

```
<?php
// PHP 5
// definitia clasei
class Urs {
   // definitia proprietatilor
   public $nume;
   public $greutate;
   public $varsta;
   public $sex;
   public $culoare;
   // constructor
   public function __construct() {
      $this->varsta = 10;
      $this->greutate = 100;
   }
   // definitia metodelor
   public function mananca($unitati) {
      echo $this->nume." mananca ".$unitati." unitati
      de mancare... ";
      $this->greutate += $unitati;
   }
   public function alearga() {
      echo $this->nume." alearga... ";
   }
   public function vaneaza() {
      echo $this->nume." vaneaza... ";
   }
   public function doarme() {
      echo $this->nume." doarme... ";
   } }
// extindem definitia clasei Urs
class UrsPolar extends Urs {
   // constructor
   public function __construct() {
      parent::__construct();
      $this->culoare = "alba";
      $this->greutate = 600;
   }
   // definitia metodelor
   public function inoata() {
      echo $this->nume." inoata... ";
   } }
  // creeaza o noua instanta a clasei Urs()
```

```
        $tom = new Urs;
        $tom->nume = " Tommy";
        // creeaza o noua instanta a clasei UrsPolar()
        $bob = new UrsPolar;
        $bob->nume = " Bobby";

        /* $bob poate apela toate metodele claselor Urs() si UrsPolar() */
        echo "Ursul polar, ". $bob->nume. " are culoarea ".$bob->culoare.", greutatea
".$bob->greutate."<br>";;
        $bob->mananca(5);
        $bob->alearga();
        $bob->vaneaza();
        $bob->doarme();
        $bob->inoata();
        // $tom poate apela doar metodele clasei Urs()
        echo "<br> Ursul ". $tom->nume. " are varsta ".$tom->varsta." ani, greutatea
".$tom->greutate."<br>";;

        $tom->mananca(8);
        $tom->doarme();
        $tom->vaneaza();
        $tom->alearga();
        echo "<br> !!! tom nu stie sa inoate pentru ca e doar urs, nu e urs polar !!!"
        ?>
```

Cuvantul cheie extends este utilizat pentru a crea o clasa copil dintr-o clasa parinte. In acest mod, toate functiile si variabilele din clasa parinte sunt disponibile in clasa copil, dupa cum se poate observa in exemplul de mai jos:

In acest caz, apelul final **$tom->inoata()** va genera o eroare deoarece clasa **Urs()** nu contine nici o metoda **inoata()**. In acelasi timp, instructiunile **$bob->alearga()** si **$bob->vaneaza()** vor fi executate cu succes deoarece clasa **UrsPolar()** mosteneste toate metodele si proprietatile clasei **Urs()**.

In exemplul anterior, poti observa cum a fost apelat constructorul clasei parinte din constructorul clasei **UrsPolar()**. In general, acest lucru este util pentru a ne asigura ca toate initializarile din clasa parinte au fost efectuate inaintea altor initializari in constructorul clasei copil. Daca o clasa mostenita nu are constructor, va fi apelat in mod implicit constructorul clasei pe care o mosteneste.

Pentru a impiedica mostenirea unei clase sau a unor metode ale sale, foloseste cuvantul cheie final inaintea numelui clasei sau al metodei (aceasta este o facilitate a PHP5 care nu este disponibila in versiunile PHP mai vechi). Iata un exemplu care modifica definitia clasei **Urs()** astfel incat aceasta sa nu mai poata fi mostenita:

```
<?php
// PHP 5
// definitia clasei
final class Urs {
   // definitia proprietatilor

   // definitia metodelor
}
/* extinderea definitiei clasei va genera o eroare
  deoarece clasa Urs nu poate fi mostenita */
class UrsPolar extends Urs {
  // definitia metodelor
}
// crearea unei instante a clasei UrsPolar()
// apelul va esua deoarece clasa Urs nu poate fi mostenita
$bob = new UrsPolar;
$bob->nume = "Ursul Bobby";
echo $bob->greutate;
?>
```

Operatorul ::

Uneori este utila folosirea unor metode sau variabile ale clasei de baza sau ale unei clase care nu a fost instantiata inca. In acest scop a fost introdus operatorul ::. Pentru a descrie modul de utilizare al acestui operator vom prezenta mai intai un exemplu:

```
<?php
class A
{
function exemplu ( )
        {
echo "Functia clasei de baza. "."<br>";
        }
}
class B extends A
{
function exemplu ( )
        {
echo "Functia redefinita "."<br>";
```

```
A :: exemplu ( );
        }
}
A :: exemplu ( );
$b = new B;
$b -> exemplu ( );
?>
```

Prin intermediul instructiunii A :: exemplu ( ); este apelata metoda exemplu ( ) a clasei A, asadar se afiseaza mesajul 'Functia clasei de baza' cu toate ca nu exista nici un obiect care este o instanta a acestei clase, deci nu putem scrie o instructiune de tipul $a -> exemplu ( );

In schimb apelam metoda $b -> exemplu ( ); ca "o functie a clasei" si nu ca "o functie a unui obiect", deoarece nu existe variabile in cele doua clase. De fapt, in momentul unui astfel de apel nu se creeaza nici un obiect care este instanta a clasei respective. Ca urmare, o functie a unei clase nu poate opera asupra unor proprietati ale clasei, dar poate utiliza variabile locale sau globale. In plus, o astfel de functie nu poate utiliza pseudo-variabila $this. In exemplul anterior, in cadrul clasei B este redefinita functia exemplu ( ). Asadar, definitia "originala" (din cadrul clasei A) nu poate fi accesata in interiorul clasei B decat daca ne referim la ea explicit prin intermediul operatorului ::

Accesarea clasei de baza

In exemplul anterior am utilizat o functie a clasei de baza. In locul utilizarii denumirii clasei de baza poate fi folosita denumirea speciala parent care este o referinta la clasa de baza definita in cadrul constructiei extends. Folosirea denumirii speciale este utila in cazul in care ierarhia de clase se modifica. In acest caz este suficienta o singura modificare in cadrul constructiei extends, fara a mai fi necesare modificari in interiorul clasei derivate. Asadar, definitia clasei B poate fi rescrisa astfel:

```
class B extends A
{
function exemplu ( )
{
echo "Functia redefinita";
parent :: exemplu ( );
}
}
```

Serializarea obiectelor

Prin serializare se intelege crearea unui sir de octeti care contine reprezentarea interna (binara) a variabilei respective. Asadar, serializarea permite "salvarea" valorilor unei variabile. Daca este serializat un obiect, sunt salvate doar proprietatile acestuia

(variabilele membre) si numele clasei din care face parte, nu si metodele deoarece functiile nu reprezinta valori. Pentru a serializa un obiect este utilizata functia *serialize ( )* care returneaza sirul de octeti care contine reprezentarea binara. Pentru a deserializa un obiect se foloseste functia pereche *unserialize ( )*. Pentru ca o astfel de operatie sa functioneze corect este necesara definirea clasei din care face parte obiectul respectiv. Functia returneaza valoarea variabilei serializate. In exemplul urmator aveti prezentat modul in care poate fi serializat si deserializat un obiect. sirul de octeti obtinut in urma serializarii va fi scris intr-un fisier si va fi citit din fisierul respectiv pentru efectuarea deserializarii. De obicei serializarea si deserializarea sunt realizate in documente php diferite deoarece aeste operatii nu au aproape nici o utilitate daca sunt folosite in cadrul aceluiasi document. Primul document in care se realizeaza serializarea trebuie sa contina o secventa asemanatoare cu urmatoarea:

```
<?php
class A {
var $msg = "Buna seara";
function scrie ( ) {
echo $this -> msg;
}
}
$a = new A;
$s = serialize ($a);
// salvarea sirului intr-un fisier
$fp = fopen ("fisier", "w");
fputs ($fp, $s);
fclose ($fp);

?>
```

Pentru deserializare al doilea document va contine urmatoarea secventa:

```
<?php

class A {
function scrie ( ) {
echo $this -> msg;}
}
// citirea sirului din fisier
$s = implode ("", @file ("fisier"));
$a = unserialize ($s);
// dupa deserializare obiectul poate fi folosit
$a -> scrie ( );

?>
```

Referinte

Referintele PHP permit unor variabile cu denumiri diferite sa corespunda unui acelasi continut. Spre deosebire de limbajul C, in PHP referintele nu sunt pointeri, ci *alias*-uri intr-o tabela de simboluri. In PHP denumirile variabilelor si continutul acestora nu sunt unul si acelasi lucru. Asadar este posibil ca acelasi continut sa aiba denumiri diferite.

Cu alte cuvinte, instructiunea $a = &$b are ca efect faptul ca $a si $b refera aceeasi variabila. In aceasta situatie $a si $b au acelasi statut. Nu se poate spune ca $a refera $b sau invers. O alta posibilitate de utilizare a referintelor este transmiterea prin referinta a parametrilor unei functii. Efectul unei astfel de transmisii este crearea unei variabile locale care refera spre acelasi continut ca variabila din contextul apelant. Sa luam in considerare urmatorul exemplu:

```
<?php
function inc (&$var)
{
$var++;
echo "var: ".$var."<br>";
}
$a = 5;
inc ($a);
inc ($a);
inc ($a);
echo "a: ".$a;
?>
```

$var este transmisa prin referinta, deci este recunoscuta automat in afara functiei si refera aceasi variabila ca si $a.

Initial valoarea variabilei $a este 5. Dupa apel variabila locala $var si variabila din contextul apelant $a indica spre acelasi continut. Valoarea pastrata in locatia de memorie respectiva este incrementata (devine 6) prin intermediul instructiunii *$var++;*. Datorita faptului ca cele doua variabile au acelasi continut, valoarea variabilei $a va fi 6 dupa prima executarea a functiei. Variabila $var este afisata in interiorul functiei, deci la fiecare executare a functiei se va afisa continutul ei. Variabila $a este afisata in afara functiei, dupa cele trei apelari, deci se va afisa ultima valoare a lui $a, respectiv a lui $var

```
var: 6
var: 7
var: 8
a: 8
```

Un parametru transmis prin referinta poate fi:
- o variabila;
- o instructiune **new**;

- o referinta returnata de o functie

Daca unei astfel de functii i se transmite ca parametru un alt tip de expresie rezultatul este nedefinit. Asadar, pentru o functie care are un parametru transmis prin referinta nu se poate folosi o constanta in momentul apelului. De exemplu, pentru functia *inc ( )* prezentata anterior nu este permis un apel de forma *inc (5)*.

Referinte globale

In momentul declararii unei variabile globale (printr-o instructiune de tipul **global** *$var*) se creeaza de fapt o referinta spre o variabila globala. Cu alte cuvinte, aceasta instructiune este echivalenta cu *$var = &$GLOBALS ["var"];*.

Referinta $this

In cadrul unei metode a unui obiect pseudo variabila **$this** este intotdeauna o referinta spre obiectul care utilizeaza functia (obiectul curent).

TEME

1. Sa se creeze clasa complex in care sa se defineasca doua variabile membre $real si $imaginar si care sa cuprinda functii pentru calculul sumei, produsului, dierenta, catul si modulul celor doua numere complexe.
2. Sa se creeze clasa dreptunghi, cu variabilele $lungime si $latime si functiile arie, perimetru. Sa se extinda aceasta clasa, in clasa paralelipiped cu variabilele $inaltime si functia volum. Sa se stabileasca functii de tip set si get in ambele clase. Sa se defineasca obiecte de tipul primei clase, respectiv a doua si sa se apeleze toate metodele posibile.

http://ro1.php.net/manual/ro/language.oop5.php