LUCRUL CU POO ÎN PHP

POO in PHP

SCOPUL:

1. **Insusirea notiunilor de baza privind POO in PHP**
2. **Elaborarea unei aplicatii utilizind POO in PHP /Anexa 1/**
3. **Pregătirea pentru TEST pe materialul POO in PHP.**

1. 1 Introducere in programarea orientata pe obiecte. Programe pe modele. POO versus PP (programare procedurală). Noțiuni de bază

Aproape toate programele modeleaza o problema din lumea reala.

Un model este o reprezentare simplificata sau o abstractie a unui sistem.

Modelul evidentiaza doar caracteristicile importante intr-un anumit context ale sistemului reprezentat.

De exemplu, o masina de jucarie prezinta anumite detalii exterioare cum ar fi carcasa si rotile, dar nu include componente ca: motorul, rezervorul etc.

Intr-un program, elementele cu ajutorul carora realizam modelarea sunt:

1. **structurile de date si**
2. **secventele de instructiuni sau operatiile.**

In functie de limbajul de programare utilizat, constructiile care redau structurile de date si operatiile sunt mai mult sau mai putin expresive, adica mai mult sau mai putin apropiate de obiectele ce formeaza domeniul problemei de rezolvat.

La nivelul cel mai de jos, din acest punct de vedere, se situeaza limbajele-masina si apoi limbajele de asamblare fcare dispun de structuri de date si operatii primitive cu care nu se pot modela complexe.

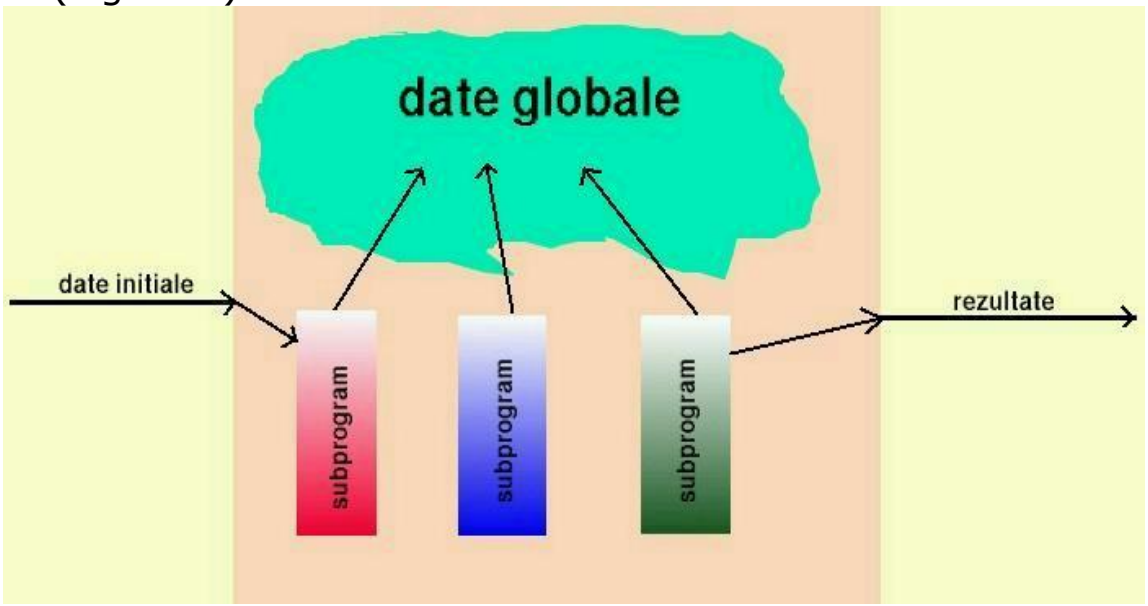
Limbajele de nivel inalt asa-numite structurate (de genul Basic, Pascal si C) aduc o crestere substantiala a nivelului de abstractizare mai ales in exprimarea operatiilor. Acestea sunt redate cu ajutorul structurilor de control (secventa, selectie si iteratie), iar

"caramizile" din care este construit un program sunt subprogramele (functiile si procedurile).

Structurile de date oferite de aceste limbaje raman insa destul de sarace pentru a acoperi complexitatea problemelor de rezolvat.

Limbajele structurate au inspirat metodele procedurale folosite la dezvoltarea produselor software; ele se caracterizeaza prin:

1. Un program conceput conform unor metode procedurale ***pune accentul pe algoritmii*** care transforma datele initiale ale unei probleme in rezultate.
2. ***Algoritmii sunt structurati in subprograme*** carora li se furnizeaza date sub forma de parametri sau care acceseaza datele respective prin intermediul unui fond global comun (Figura 1).



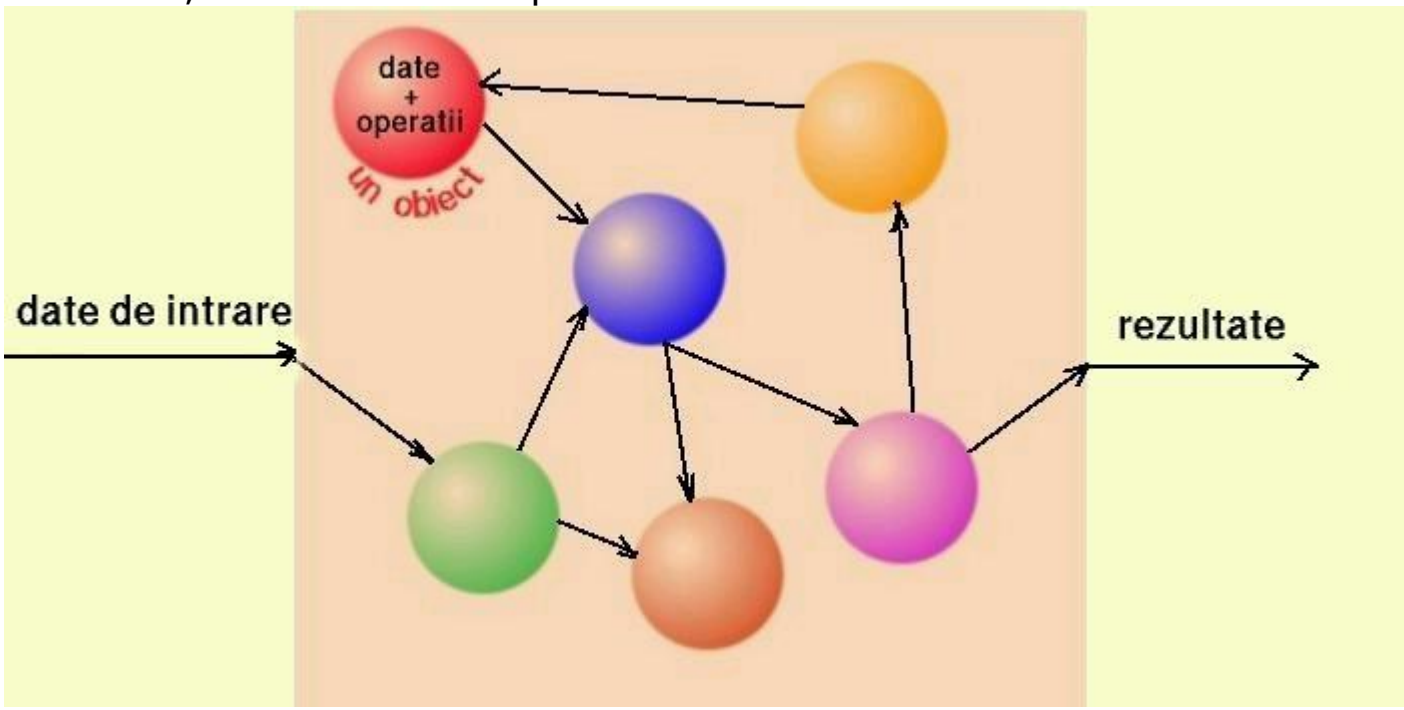
Limbajele de programare orientate pe obiecte (LPOO) aduc o noua viziune asupra dezvoltarii programelor, urmarind sa acopere "slabiciunile" limbajelor structurate privind structurile de date.

Mai precis, programatorului i se da posibilitatea **de a-si construi propriile tipuri de date**, **dotate cu operatiile necesare**, *care sa "mimeze" cat mai fidel entitatile din* ***domeniul PROBLEMEI de rezolvat***.

Un program dezvoltat cu ajutorul tehnologiei obiectuale are drept unitate de constructie nu subprogramul, ci obiectul (Figura 2).

Un obiect inglobeaza date si operatii si reprezinta o abstractie a unei entitati din lumea reala.

Obiectele componente interactioneaza, iar rezultatul acestei interactiuni o reprezinta ***transformarea datelor de intrare in date de iesire***, adica rezolvarea problemei.



Folosind tehnologia obiectuala, abstractizarea (intelegand prin asta tranzitia de la entitatile domeniului problemei de rezolvat la programul care o modeleaza) este mai directa, mai naturala. Motivul sta in faptul ca noi, oamenii percepem lumea inconjuratoare in termeni de obiecte fizice si concepte mentale care se misca, se transforma, interactioneaza unele cu altele, determinand in felul acesta modificari ale mediului inconjurator.

Un obiect din lumea reala are cateva caracteristici importante, si anume:

1. **este perceput prin intermediul comportamentului si insusirilor sale exterioare**; de exemplu, vedem ca o pisica merge, fara sa stim (si fara sa ne intereseze chiar) care sunt mecanismele interioare care determina mersul. Cu alte cuvinte, obiectul are o față exterioară (interfata publica) pe care o arata altor obiecte, precum si o reprezentare (stare) interna care imprima comportarea exterioara;
2. **are granite distinct conturate, care il delimiteaza de celelalte obiecte si este autodeterminat**: pastrand exemplul cu pisica, pentru a vedea ca ea merge, nu avem nevoie sa vedem un alt obiect suplimentar; spunem ca obiectul are identitate distincta;

In cazul programarii procedurale abstractizarea presupune realizarea unui set de algoritmi (subprograme) care sa modeleze transformarile din lumea reala, folosind pentru reprezentarea unor entitati complexe structuri de date primitive.

In cazul programarii obiectuale, abstractizarea presupune definirea unor structuri de date complexe dotate cu un comportament exterior si cu o stare interna, care se suprapun peste obiectele din lumea reala.

Intr-un program dezvoltat in maniera obiectuala NU mai avem date globale (sau in orice caz, foarte putine). Datele sunt repartizate si inglobate in obiecte. Putem spune ca metoda obiectuala de dezvoltare conduce la o mai buna respectare a unor principii ale ingineriei software cum ar fi:

1. ***localizarea si modularizarea:*** codul sursa corespunzator unui obiect poate fi scris si actualizat independent de alte obiecte;
2. ***ascunderea informatiei:*** **un obiect are**, dupa cum vom vedea si in lucrarile urmatoare, o interfata publica pe care celelalte obiecte o pot utiliza in vederea comunicarii. Dar, pe langa interfata publica, un obiect poate include date si operatii private, "ascunse" fata de alte obiecte, si care pot fi modificate oricand, fara a afecta restul obiectelor.
3. ***reutilizarea codului:*** **multe clase de obiecte** pot fi definite pe baza altor clase deja existente, preluandu-se automat (prin mostenire) continutul acestora din urma;

Notă: Trecand la un limbaj de programare orientat pe obiecte nu inseamna ca "aruncam" principiile programarii structurate. Operatiile inglobate in obiecte nu sunt altceva decat un fel de subprograme (chiar daca au alt nume) care vor fi dezvoltate utilizand structurile de control cunoscute de la limbajele structurate.

Intr-o exprimare mai clasică:

Diferenta esentiala intre limbajele procedurale si cele obiectuale este aceea ca limbajele procedurale au un caracter imperativ, fiind axate pe "verbe" (subprograme), in timp ce limbajele obiectuale au un caracter mai declarativ si sunt concentrate in jurul "substantivelor" (datele).

Tinand cont de cele afirmate pana aici, putem formula ***principiul dupa care realizam transpunerea enuntului unei PROBLEME intr-un limbaj de programare orientat pe obiecte***:

1. ***identificam obiectele*** **ce populeaza spatiul programului, prin evidentierea substantivelor din enuntul problemei;**

2. ***pentru fiecare obiect identificam datele si operatiile*, prin evidentierea adjectivelor si verbelor ce caracterizeaza substantivul respectiv.**

POO versus PP (programare procedurală)

In cele ce urmeaza prezentăm un exemplu de problema modelata in maniera procedurala si in maniera obiectuala, pentru a vedea mai bine diferentele intre cele doua viziuni.

Problema se refera la crearea si utilizarea unei stive de numere intregi. Dupa cum probabil se cunoaște, o stiva este o colectie cu structura liniara, in/din care se pot introduce/extrage elemente pe la un singur capat, considerat a fi varful stivei (celalalt capat este baza).

Operatiile asupra stivei au denumiri consacrate in literatura, si anume: introducerea elementelor este numita **push**, iar extragerea - **pop**; in plus, exista o operatie de consultare a valorii din varful stivei, numita **top**.

Pentru a nu va solicita atentia cu detalii sintactico-semantice specifice unui anumit limbaj de programare, vom considera ca dispunem de un fel de **pseudo-cod** pentru ambele maniere de modelare.

Convenim ca pentru reprezentarea stivei folosim un tablou cu elemente intregi, caruia i se asociaza un indicator de stiva reprezentand indicele ultimului element ocupat in tablou. De asemenea, vom folosi o variabila necesara memorarii dimensiunii totale a tabloului.

```
-- Stiva in varianta procedurala -
subprogram initStiva(unTab, dimTab, spTab)
subprogram push(unTab, dimTab, spTab, elem)
else
{
       *semnaleaza eroare - depasire
       capacitate stiva
}
subprogram int pop(unTab, dimTab, spTab)
else
{
            *semnaleaza eroare - stiva vida
}
subprogram int top(unTab, dimTab, spTab)
subprogram client( )
```

```
-- Stiva in varianta obiectuala
clasa Stiva
      operatia push(elem)
      else
      {
*semnaleaza eroare - depasire
capacitate stiva
      }
      operatia int pop( )
      else
      {
*semnaleaza eroare - stiva vida
      }
      operatia int top( )
{
-- end clasa Stiva --
}
clasa Client( )
{
-- end clasa Client --
}
```

Comparand cele doua secvente de mai sus, un prim aspect care trebuie remarcat este faptul ca **in varianta procedurala** am definit un set de **operatii reprezentate ca subprograme**, care primesc ca parametri stivele asupra carora lucreaza, in timp ce **in varianta obiectuala**, operatiile sunt inglobate in interiorul unei clase reprezentand stiva.

Putemt sa avem oricate obiecte de acest fel, fiecare din ele va avea setul sau propriu de operatii care vor lucra asupra datelor interne ale obiectului. Din acest motiv operatiile nu necesita atatia parametri, ca in varianta procedurala.

Cu alte cuvinte: in varianta procedurala operatiile (subprogram-le) sunt detinatoarele datelor asupra carora lucreaza, in timp ce in varianta obiectuala datele (obiectele) sunt cele care poseda operatiile.

O alta observatie importanta este faptul ca simbolul **Stiva** definit ca o **clasa** este de fapt un tip definit de utilizator. Ca orice tip, el poate fi folosit pentru a declara variabile asupra carora se vor aplica operatii recunoscute de tipul respectiv.

Pentru a intelege mai bine, sa ne gandim ce este un tip dintr-un limbaj de programare:

un tip este o multime de valori dotata cu un anumit set de operatii.

Exemplu: In majoritatea limbajelor procedurale suntem obisnuiti sa lucram cu asa-numitele tipuri predefinite sau primitive. Spre exemplu, tipul int din limbajul C desemneaza multimea numerelor intregi reprezentabile pe 2 octeti.

Setul de operatii cu care este dotat acest tip cuprinde, printre altele cele patru operatii aritmetice.

Dar, de multe ori tipurile predefinite sunt prea sarace pentru a permite modelarea unor entitati complexe din lumea reala.

Iata de ce, este necesar ca programatorul sa aiba libertatea de a-si crea propriile tipuri (impreuna cu operatiile aferente) pe care apoi sa le foloseasca la fel cum foloseste tipurile predefinite. Limbajele de programare obiectuale au fost concepute tocmai in acest sens.

Tot pe marginea celor doua exemple prezentate, ar trebui spus ca in varianta obiectuala **programatorul este ajutat sa evite o serie intreaga de erori**, datorita incapsularii datelor si operatiilor caracteristice unei abstractiuni (in cazul nostru stiva) in interiorul unui obiect.

Spre exemplu, in varianta procedurala, programatorul poate la un moment dat sa modifice accidental, printr-o atribuire simpla,

valoarea lui sp1, dand astfel peste cap stiva respectiva. In cealalta varianta, asa cum vom vedea pe parcurs, datele din interiorul unui obiect pot fi declarate ca ascunse fata de exteriorul obiectului, astfel incat modificarea indicatorului de stiva sa nu se poata face decat via operatiile push si pop, adica intr-un mod controlat.

Definitia POO. ***Programarea orientata pe obiecte este o metoda de implementare in care programele sunt organizate ca ansamble de obiecte ce interactioneaza unele cu altele, fiecare obiect reprezentand instanta unei clase; fiecare clasa apartine unei ierarhii in cadrul careia clasele sunt legate prin relatii de mostenire.***

Aceasta definitie cuprinde trei parti importante, si anume:

1. **obiectele** si nu algoritmii sunt blocurile logice fundamentale;
2. **fiecare obiect** este o **instanta a unei clase**. **Clasa** este o descriere a unei multimi de obiecte caracterizate prin structura si comportament similare, iar prin ***instanta intelegem o valoare particulara*** din multimea respectiva
3. **clasele** sunt legate intre ele prin **relatii de mostenire**.

Instanta de clasa

Orice obiect are un tip. Deoarece un tip nu e o multime moarta de date, ci este dotata cu operatii si in general anumite tipuri au anumite seturi de operatii (ex: numerele au adunari scaderi inmultiri impartiri etc)

prin instanta / instantiere a unei clase

intelegem procesul prin care ***inzestram acel tip de date cu operatiile specifice tuturor tipurilor din acea clasa.***

Un limbaj de programare care ofera suport pentru utilizarea claselor si a obiectelor, dar care nu are implementat mecanismul relatiilor de mostenire intre clase este un limbaj de programare bazat pe obiecte.

Programarea bazata pe clase si pe obiecte**, care nu face uz de relatia de mostenire se mai numeste programare cu tipuri de date abstracte.**

AICI

Sarcină:

Prezentati obiecte ce ar putea sa faca parte dintr-un program care modeleaza diverse situatii desprinse din viata de zi cu zi, cum ar fi: cumpararea unor produse dintr-un magazin cu autoservire, operatia de vanzare-cumparare de marfuri intre doua firme (cu fluxul de documente aferent: facturi, chitante de plata etc), functionarea unui ascensor.

Sugestie:
descrieti mai intai in limbaj natural procesele care au loc in situatiile respective, apoi transpuneți enuntul intr-un limbaj obiectual.

1.2 PROGRAMAREA ORIENTATA PE OBIECTE 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 /citeț/
- 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 si cele ce urmeaza.

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** (obiecte).

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**

Exemplu: 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***:

Exemplu: /ERONAT!!/

```
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.

Foarte important!!!

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.***

Foarte important!!!

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>";
?>
```

Obiect

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.

Proprietăți

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 = “En”;
}
```

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 = “En”;
}
```

Putem accesa proprietatile obiectelor folosind operatorul "->".

Ca atare, "$en->tip" inseamna proprietate $tip din obiectul Dictionar referentiat de "$en".

Metod

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;
```

La nivel de obiect creat prin

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

Noi putem schimba/modifica **proprietatile individuale** ale **$aritm** putand fi manipulate independent fata de cele ale altor obiecte de acelasi tip.

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

Desigur de aici si rezultatul va fi unul diferit al metodelor ce sunt commune si definite de clasa respectiva!!!

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;
}
}
$ion=new Oameni("Ion",20,"inginer",false,50);//apelul constructorului
$ion->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, $n)
function Student($y = "4", $a = "22", $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 $this->name;
}
Function afisare()
{
echo "Numele este ".$this->name. “anul este”. $this->year. “$this->age.”   "<br>";
}
}
$st = new Student("2020","20","Petru");
echo "Primul an este ".$st->getYear()."<br>";
$st->setYear("1998");
echo "Al doilea an este ".$st->getYear()."<br>";

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

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

$stud = new Student();
$stud->afisare();
$stud = new Student(2);
$stud->afisare();
$stud = new Student(2, 20);
$stud->afisare();
$stud = new Student(5, 30,”Petru”);
$stud->afisare();
?>
```

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 *get*** – 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,”Petru”);
// year = 5, age = 30, name = "Petru"
```

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().

Destructor

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 si 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

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()
    $tim = new UrsPolar;
    $tim->nume = "Timofei";
    // creeaza o noua instanta a clasei UrsPolar()
    $mis = new Urs;
    $mis->nume = "Misca";

    /* $tim poate apela toate metodele claselor Urs() si UrsPolar() */
    echo "Ursul polar, ". $tim->nume. " are culoarea ".$tim->culoare.", greutatea
".$tim->greutate."<br>";;
    $tim->mananca(5);
    $tim->alearga();
```

```
$tim->vaneaza();
$tim->doarme();
$tim->inoata();
// $mis poate apela doar metodele clasei Urs()
echo "<br> Ursul ". $mis ->nume. " are varsta ".$mis->varsta." ani, greutatea ".$mis->greutate."<br>";;

$mis->mananca(8);
$mis->doarme();
$mis->vaneaza();
$mis->alearga();
echo "<br> !!! Misca 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 $mis->inoata() va genera o eroare deoarece clasa **Urs()** nu contine nici o metoda **inoata()**. In acelasi timp, instructiunile **$tim->alearga()** si **$tim->vaneaza()** vor fi executate cu succes deoarece clasa **UrsPolar()** mosteneste toate metodele si proprietatile clasei **Urs()**.

In exemplul anterior, putem 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
$tim = new UrsPolar;
$tim->nume = "Ursul Timofei";
echo $tim->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 există 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 bază

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).

http://marplo.net/php-mysql/atribuire-valoare-referinta.html

Atribuirea prin referinta se face folosind semnul "&" in fata variabilei.

- Exemplu:

```
$x = 'abc';
$y = &$x;
echo $y;       // abc
```

Atribuirea prin referinta e mai mult decat o simpla atribuire de valoare, de fapt leaga intre ele cele 2 variabile pe parcursul scriptului. Astfel, daca se modifica ulterior valoarea uneia dintre variabile, se transmite si la cealalta.

Exemplu (vedeti comentariile din cod):

```
<?php
$x = 'MarPlo.net';
$y = &$x;
echo $y;     // MarPlo.net
// Se modifica valoarea lui $x, si va afecta si pe $y
$x = 'CoursesWeb.net';
echo '<br/>'. $y;    // CoursesWeb.net
// Se modifica valoarea lui $y, se schimba si la $x
$y = 'php.net';
echo '<br/>'. $x;     // php.net
?>
```

ATRIBUIRE PRIN REFERINTA SI FUNCTII

Atribuirea prin referinta se transfera si de la variabile (parametru) definite in interiorul unei functii catre variabila din exterior transmisa la apelare, daca se adauga caracterul "&" la parametru functiei cand e definita. Va afecta orice variabila folosita ca argument pentru acel parametru la apelare.
- Exemplu, variabila folosita ca argument la apelare e afectata /modificata si ea de valoarea respectivului parametru definit cu "&":

```
<?php
function foo(&$a) {
  $a = $a + 2;
}

$x = 7;

// Se apeleaza functia cu variabila $x la argument
// Parametrul lui foo() avand caracterul "&", va transmite ultima valoare a lui ( setata in corpul functiei) si in exterior, la $x

foo($x);
echo $x;     // 9
?>
```

.Dupa cum observati, atribuirea prin referinta poate crea situatii /efecte destul de complexe, de aceea este indicat a se evita folosirea ei, mai ales la incepatori.

Variabile si referinte in functii

Curs Php-mysql
http://marplo.net/php-mysql/functii_variabile.html

- învatati sa folositi variabile globale, locale si statice
- învatati sa folositi referintele

Variabilele functiilor in PHP sunt de doua tipuri principale:

- **Variabile globale**
- **Variabile locale**

Variabilele globale sunt create în exteriorul functiei, în timp ce variabilele locale sunt create în interiorul unei functii.

1. Utilizarea variabilelor globale

Asa cum s-a explicat anterior, variabilele globale sunt cele declarate în afara oricarei functii. Totalitatea locurilor unde este accesibila o variabila se numeste "domeniu de existenta al variabilei".

Variabilele globale nu pot fi accesibile din interiorul corpului unei functii; cu alte cuvinte, domeniul de existenta al unei variabile globale, nu include corpurile functiilor. Daca doriti sa obtineti accesul la o variabila globala în cadrul unei functii, puteti extinde domeniul de existenta al variabilei prin specificarea in functie a numelui variabilei, în cadrul unei instructiuni **GLOBAL**.
Instructiunea GLOBAL are urmatoarea forma:

- **GLOBAL variabila1, variabila2, variabila3;**

Iata un exemplu concludent :

```
<?php
$var1 = 135;
$var2 = 250;
function Suma() {
    return $var1 + $var2;
}
echo "Suma este ". Suma();
?>
```

- Functia "Suma()" foloseste variabilele $var1 si $var2 declarate anterior, dar aceste variabile nu au domeniu de valabilitate in interiorul functiei. Prin urmare rezultatul functiei este NULL.
Pentru a rezolva aceasta problema vom declara cele doua variabile ca fiind variabile globale, ca in exempul urmator:

```
<?php
$var1 = 135;
$var2 = 250;
function Suma() {
    GLOBAL $var1, $var2;
    return $var1 + $var2;
}
echo "Suma este ". Suma();
?>
```

- In exemplul de mai sus declaratia: "GLOBAL $var1, $var2" face ca variabilele $var1 si $var2 sa fie recunoscute si in interiorul functiei.
- *Puteti scrie si cu litere mici "global", dar e mai usor de recunoscut cu litere mari.*

O alta metoda prin care putem solutiona aceasta problema este prin folosirea variabilei PHP predefinite **$GLOBALS**, dupa cum puteti vedea in exemplu urmator:

```
<?php
$var1 = 135;
$var2 = 250;
function Suma() {
    return $GLOBALS['var1'] + $GLOBALS['var2'];
}
echo "Suma este ". Suma();
?>
```

$GLOBALS este o variabila predefinita, este de fapt un array ; elementele acestei matrice au cheia egala cu numele variabilelor declarate si valoarea egala cu cea a variabilelor declarate.
$GLOBALS este o variabila superglobala, ea va fi recunoscuta in orice script.
Puteti folosi oricare din cele doua variante prezentate: instrucrtiunea **GLOBAL** sau variabila **$GLOBALS**.
In cele doua exemple prezentate mai sus rezultatul afisat va fi acelasi :

Suma este 385

2. Utilizarea variabilelor locale

Variabilele locale sunt create in interiorul functiei si sunt distruse cand se încheie apelul la functia respectiva. În consecinta, variabilele locale sunt disponibile numai pe durata executiei functiei asociate. Argumentele functiilor constituie un tip important de variabila locala. Cu toate acestea, puteti crea o variabila locala prin simpla atribuire a unei valori unei variabile din interiorul unei functii.
Pentru a ilustra deosebirea dintre variabilele locale si cele globale, iata un script care defineste o variabila locala denumita "$x" si o variabila globala cu acelasi nume:

```
<?php
function v_local() {
    $x = 5;
    echo "<br />In corpul functiei x = $x";
}
$x = 2;
echo "<br />In corpul scriptului x = $x";
v_local();
echo "<br />In corpul scriptului x = $x";
?>
```

Cand executati acest script, veti primi urmatoarele rezultate:

In corpul scriptului x = 2
In corpul functiei x = 5
In corpul scriptului x = 2

Remarcati diferenta dintre cele doua variabile $x, chiar daca numele variabilelor este acelasi.
Cand este apelata functia v_local(), $x ia valoarea 5, in rest, inainte si dupa, ia valoarea 2.
Domeniul de existenta al variabilei globale $x nu se extinde în interiorul corpului functiei v_local(), iar domeniul de existenta al variabilei locale $x nu se extinde dincolo de corpul functiei respective. Cu alte cuvinte, domeniile de existenta ale celor doua variabile sunt complet distincte.

3. Utilizarea variabilelor statice

Uneori este nevoie ca o variabila locala sa-si pastreze valoarea de la un apel al functie asociate la altul. Altfel spus, nu doriti ca variabila sa fie distrusa la încheierea apelului la functie.
Pentru acest lucru puteti folosi instructiunea **STATIC** (*la fel cu "static"*).
Forma instructiunii **STATIC** este similara cu aceea a instructiunii **GLOBAL**.

- **STATIC $var1, $var2, $var3;**

O variabila afisata într-o instructiune STATIC este cunoscuta sub numele de "variabila statica".
Iata un exemplu care prezinta modul de utilizare a unei variabile statice:

```
<?php
function v_local() {
      $x = $x + 1;
      echo "<br /> x = $x";
}
function v_static() {
      STATIC $x;
      $x = $x + 1;
      echo "<br /> x = $x";
}
v_local();
v_local();
v_local();

v_static();
v_static();
v_static();
?>
```

Daca rulati acest script, veti primi urmatoarele rezultate:

```
x = 1
x = 1
x = 1
x = 1
x = 2
x = 3
```

Observati ca variabila locala $x, definita în cadrul functiei v_local(), este creata din nou la fiecare apelare a functiei, în consecinta, valoarea sa este întotdeauna afisata ca fiind egala cu 1.
Prin contrast, variabila statica $x, definita în cadrul functiei v_static(), îsi pastreaza valoarea de la un apel al functiei la urmatorul; ca atare, valoarea sa creste de fiecare data când este executata functia.

4. Utilizarea referintelor

In mod prestabilit, argumentele transferate unei functii PHP sunt transmise prin valoare, ceea ce înseamna ca valorile argumentelor sunt copiate si functiile utilizeaza copii ale valorilor argumentelor lor, nu argumentele în sine. Ca o consecinta, o functie PHP nu poate modifica valorile argumentelor sale.

Totusi, puteti stabili ca o functie sa aiba posibilitatea de a modifica valoarea unui argument, specificând ca argumentul sa fie transferat prin referinta.

Când un argument este transferat prin referinta, valoarea sa nu este copiata; functia lucreaza cu valoarea argumentului si are libertatea de a modifica acea valoare.

Pentru a specifica faptul ca un argument urmeaza a fi transferat prin referinta, inaintea argumentului va fi adaugat un caracter ampersand (**&**). Puteti atasa acest caracter la argument în antetul functiei sau în apelul la functie.

Iata un exemplu care prezinta apelul prin valoare si apelul prin referinta:

```
<?php
function p_valoare($a) {
      $a = 1;
}
function p_referinta(&$a) {
```

```
    $a = 1;
}

$b = 0;
p_valoare($b);
echo "<br /> \$b = $b";

$b = 0;
p_valoare(&$b);
echo "<br /> \$b = $b";

$b = 0;
p_referinta($b);
echo "<br /> \$b = $b";
?>
```

Dupa rulati acest script, veti obtine urmatoarele date de iesire:

```
$b = 0
$b = 1
$b = 1
```

Retineti ca scriptul contine doua functii, si anume "p_valoare()" si "p_referinta()". Fiecare functie preia un singur argument, denumit $a. Antetul functiei "p_referinta()" specifica faptul ca argumentul sau este transferat prin referinta; argumentul functiei "p_valoare()" este transferat prin valoare.

Primul paragraf al programului invoca functia "p_valoare()", transferând argumentul prin valoare. În consecinta, functia lucreaza cu o copie a argumentului sau, iar valoarea variabilei $b nu se modifica.

Cel de-al doilea paragraf al programului invoca de asemenea functia "p_valoare()"; dar, foloseste un caracter ampersand (**&**) pentru a determina transferul prin referinta al valorii variabilei $b. În consecinta, functia modifica valoarea argumentului sau, care se transforma din 0 în 1.

Cel de-a treilea paragraf al programului invoca functia "p_referinta()". Antetul functiei respective foloseste un caracter ampersand (**&**) pentru a specifica faptul ca valoarea argumentului sau este transferata prin referinta, în consecinta, functia modifica valoarea argumentului sau, care se transforma din 0 în l.

Prin utilizarea referintelor se evita suprasarcina de copiere a valorilor argumentelor si implicit se obtine o viteza mai mare de executie a programului. Cu toate acestea, programele devin astfel mai dificil de înteles, iar referintele sau apelurile prin referinta pot cauza erori de program. Cel mai indicat este sa evitati referintele, acolo unde este posibil, si sa definiti functii care returneaza valori, si nu functii care modifica valorile propriilor argumente.

*In versiunile PHP 5+, transmiterea la functiei a unui parametru prin referinta "Call-time pass-by-reference", **p_valoare(&$b)**, a fost scoasa din uz. Dar poate fi activata din php.ini, daca aveti acces, prin setarea 'true' la "allow_call_time_pass_reference".*

Variabile globale în PHP

http://www.php.su/learnphp/vars/?glvars

Variabilele **globale** sunt variabile disponibile întregului program, inclusiv rutine-lor (funcțiilor definite de utilizator).

Variabilele **locale** sunt variabile definite în cadrul unui subprogram (funcția definita utilizator). Sunt disponibile doar în cadrul funcției în care sunt definite.

Pentru PHP, **toate variabilele** declarate și utilizate într-o funcție **sunt locale** în mod implicit funcției. Adică, în mod implicit, nu există nici o modalitate de a modifica valoarea unei variabile globale în corpul funcției.

Dacă utilizați o variabilă cu numele identică cu numele variabilei globale (situată în afara funcției definite de utilizator) în corpul funcției definite de utilizator, atunci această variabilă locală nu va avea nicio legătură cu variabila globală. În această situație, în funcția definită de utilizator va fi creată o variabilă locală cu un nume identic cu numele variabilei globale, dar această variabilă locală va fi disponibilă numai în cadrul acestei funcții definite de utilizator.

Să explicăm acest fapt cu un exemplu specific:

```
<?php
$a = 100;

function funct() {
$a = 70;
 echo "<h4>$a</h4>";
}
funct();
echo "<h2>$a</h2>";
?>
```

Mai intii va fi prezentat rezultatul 70 urmat de al doilea egal cu 100:

```
70
100
```

Pentru a scăpa de acest neajuns, în PHP există o instrucțiune globală specială care permite unei funcții definite de utilizator să lucreze cu variabile globale. Vedeti acest principiu in exemplele ce urmeaza:

```
<?php
$a = 1;
$b = 2;
function Sum()
{
     global $a, $b;
     $b = $a + $b;
}
Sum();
echo $b;
?>
```

Scriptul de mai sus va scoate „**3**”. După definirea lui **$ a** și **$ b** în interiorul funcției ca globală, toate referințele la oricare dintre aceste variabile vor indica versiunea lor globală. Nu există restricții privind numărul de variabile globale care pot fi procesate de funcțiile definite de utilizator.

Al doilea mod de accesare a variabilelor de aplicare globală este utilizarea tabloului special **$ GLOBALS** definit de PHP. Exemplul anterior poate fi rescris astfel:

Folosind **$ GLOBALS** în loc de global:

```
<?php
$a = 1;
$b = 2;

function Sum()
{
     $GLOBALS["b"] = $GLOBALS["a"] + $GLOBALS["b"];
}

Sum();
echo $b;
?>
```

$ GLOBALS este un tablou asociativ a cărui cheie este numele și valoarea este conținutul variabilei globale. Rețineți că $ GLOBALS există în orice domeniu vizibil, deoarece acest tablou este superglobal. Următorul este un exemplu care demonstrează capacitățile variabilelor **superglobale**:

```
<?php
function test_global()
{
      // Majoritatea variabilelor predefinite nu sunt
      // "super" și ca să fie disponibil în zona vizibilitatii locale a
      // funcției necesită declarația „global”.
     global $HTTP_POST_VARS;

     echo $HTTP_POST_VARS['name'];

      // Variabilele superglobale sunt disponibile în orice zonă de
      // vizibilitate și nu necesită indicarea „global”.
      // Variabilele Superglobale sunt disponibile începând cu PHP 4.1.0
     echo $_POST['name'];
}
?>
```

Vedeti de asemenea si:

Локальные переменные в PHP