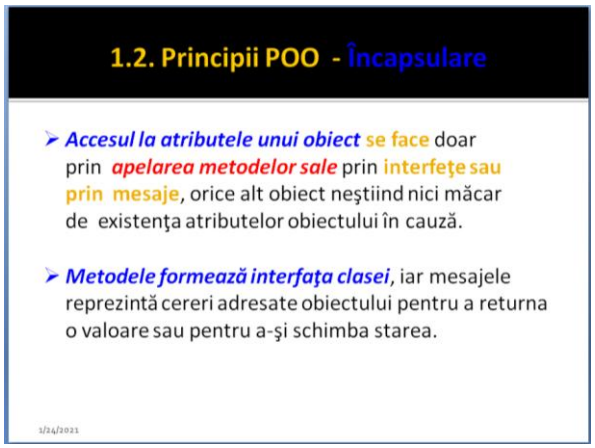
Programarea Obiect Orientată (POO) în limbajul PHP

Pentru aplicaţiile de mari dimensiuni, o dezvoltare structurată a codului (orientată pe funcţii/proceduri), implică existenţa unui număr foarte mare de linii de cod, prin modul de organizare a codului și conduce la mari dificultăţi privind realizarea unor modificări ulterioare în cadrul aplicaţiei.

Programarea orientată pe obiecte a apărut ca o necesitate în contextul creşterii complexităţii codului aplicaţiilor software.

Programarea Obiect Orientată (POO) în limbajul PHP oferă o modalitate diferită de organizare a codului şi a datelor în cadrul unui program - programele sunt reprezentate sub forma unor colecţii de obiecte care interacţionează între ele prin intermediul mesajelor.



ELEMENTELE DE BAZĂ ÎN POO

Clasa şi obiectul – reprezintă noţiunile de bază în metoda OO.

Obiectul – este o reprezentare a unei entităţi din lumea reală asupra căruia se poate ***întreprinde o acţiune*** sau ***care poate întreprinde o acţiune***. Sau, altfel spus, ***obiectul*** este o ***variabilă structurată, care conţine toată informaţia necesară referitoare la o entitate fizică*** sau ***o noţiune realizată prin coduri de program***.

Clasa reprezintă ***definirea, descrierea unuia sau a mai multor obiecte*** şi ***ale acţiunilor pe care acestea le pot realiza***

Prin instanţiere a unei clase este creat un obiect (evident se pot face instanţieri multiple, construindu-se mai multe obiecte ale aceleiaşi clase)

PHP şi POO

PHP dispune de posibilităţi OO foarte generale. ***PHP nu este un limbaj de programare OO dedicat,*** aşa cum sunt de exemplu **C++ sau Java**. Dar principiile de bază ale POO sunt susţinute şi în PHP:

Abstractizarea - este procesul de grupare a datelor şi metodelor de prelucrare specifice rezolvării unei probleme.

Încapsularea - înseamnă gruparea datelor şi a operaţiilor asupra acestor date în acelaşi întreg (agregat), ascunzându-se detaliile de implementare (proiectare-realizare) ale acestui întreg.

Moştenirea - permite unei clase sa ***moştenească atributele şi metodele*** unei alte clase existente.

Polimorfismul - unele dintre proprietăţile şi metodele definite in superclasa pot fi redefinite (rescrise) in subclasele de obiecte derivate.

DEFINIREA CLASEI ÎN PHP

În PHP clasa se defineşte cu ajutorul cuvântului rezervat **class** şi un nume aleatoriu pentru clasă.

Sintaxa de definire:

class denumire_clasă **{ Corpul clasei }**

Mai detaliat, structura **corpului clasei** poate fi expusă astfel:

```
class denumire_clasă
{
var /vizibilitate $denumire_proprietate;
/*listă proprietăţi*/
function denumire_metodă( )
{
/* Definire metodă */
}
/*listă metode*/
}
```

DEFINIREA OBIECTULUI

Dacă despre **clasă** se poate spune că **are rol de şablon** în crearea obiectelor, atunci ***obiectul reprezintă date structurate conform şablonului, definit de clasă***.

Se mai spune că obiectul este un exemplar al clasei, tipul fiind definit de clasă.

De exemplu, **fie clasa:**

```
class produs
    {  Corpul clasei  }
```

Obiectele care pot fi definite de această clasă se creează astfel:

```
$produs1= new produs();
$produs2 = new produs();
```

Fiecărui obiect, definit în script, i se pune în corespondenţă un **nume unic**, pentru a-l identifica. Acest nume este unic de-a lungul vieţii obiectului.

ELEMENTELE / MEMBRII CLASEI

În cadrul clasei sunt precizate:

atributele sau proprietăţile – **partea de date** a *obiectului, reprezentate prin declaraţii de variabile, inclusiv posibile iniţializări ale acestora.*

Proprietăţile se definesc folosind cuvântul cheie var (până la versiunea 5) sau specificându-se vizibilitatea proprietăţii (după versiunea 5)

metodele – partea de cod a **clasei**, reprezentate prin **funcţii** definesc comportamentul obiectelor. Totodată constituie şi **interfaţa obiectului** destinată manipulării datelor acestuia.

Pentru a obţine acces la membrii clasei – proprietăţi şi funcţii – în interiorul clasei, poate fi folosită referinţa $this, care se referă mereu la elementul curent.

DEFINIREA PROPRIETĂŢILOR CLASEI

În interiorul clasei se definesc variabile speciale care sunt numite proprietăţi. **Proprietatea, numită şi variabilă-membru (member variable), conţine date care se modifică de la un obiect la altul.**

Definirea proprietăţii în clasă se aseamănă cu definirea unei variabile, plasându-se în faţă cuvântul rezervat **var** sau, începând cu versiunea 5, în faţă se plasează unul dintre cuvintele rezervate:

1. **public**
2. **protected**
3. **private,**

care definesc domeniul de vizibilitate a membrului.

Cuvintele cheie **public, protected sau private** stabilesc la ce variabilă este posibil accesul şi din ce context.

Cuvintele cheie **public, protected şi private**, care definesc domeniul de vizibilitate ale proprietăţilor, au apărut în PHP 5. În PHP 4 toate proprietăţile trebuie definite folosind cuvântul cheie **var, *ceea ce este identic cu folosirea cuvântului cheie public***.

Nită: Reieşind din ***principiul compatibilităţii inverse***, în PHP5 se admite utilizarea cuvântului **cheie var** la definirea proprietăţilor, în locul cuvântului rezervat **public.**

Domeniul de vizibilitate defineşte **contextul funcţiei sau a proprietăţii**, în cadrul căruia poate fi utilizată variabila sau metoda.

Astfel, variabila definită în interiorul corpului funcţiei, are un domeniu de vizibilitate local, iar variabila definită în afara funcţiei — posedă un domeniu de vizibilitate global.

De regulă, nu este posibilă accesarea datelor, plasate în locaţii de vizibilitate limitate în raport cu cea curentă. **Din acest motiv variabila definită în interiorul funcţiei, nu va putea fi accesată dinafara funcţiei.**

Exemplu definire proprietăţi

```
class produs
{
        public $denumireProdus = "Cactus";
        public $numeProducator = "Flora SA";
        public $dataProducerii = "12/12/2014";
        public $pretUnitar = 90;
}
```

ACCESUL LA PROPRIETĂŢI

Pentru a accesa membrii clasei în PHP se foloseşte OPERATORUL "->".
ADRESAREA SE FACE CUNOSCÂND NUMELE OBIECTULUI ŞI NUMELE PROPRIETĂŢII.

Dacă proprietăţile sunt definite ca și „publice" pot fi accesate valorile acestor proprietăţi sau pot fi atribuite noi valori proprietăţilor, modificând astfel valorile standard definite în clasă, **la iniţializare.**

Exemplu accesare proprietăţi

```
class produs
{
        public $denumireProdus = "Cactus";
        public $numeProducator = "Flora SA";
        public $dataProducerii = "12/12/2021";
        public $pretUnitar = 90;
}
$produs1 = new produs();
$produs2 = new produs();
// Modificam valorile proprietatilor
$produs1->denumireProdus = 'Begonie';
$produs2->denumireProdus = 'Muscata';
// Afişăm valorile proprietăţilor
echo $produs1->denumireProdus."<br />";
echo $produs2->denumireProdus;
```

Afişarea rezultatelor:

??????????????????????

Accesarea mai multor proprietăţi

```
$produs1->denumireProdus = 'Begonie';
$produs1->numeProducator = 'Plante SA';
$produs1->dataProducerii = '01/01/2021';
$produs1->pretUnitar = 70;
echo "Descriere produs: <b>".$produs1->denumireProdus.", ".$produs1->numeProducator.", ".$produs1->dataProducerii.", ". $produs1->pretUnitar."</b><br />";
```

Afişarea rezultatelor:

??????????????????????

DEFINIREA METODELOR CLASEI

Aşa cum **proprietăţile permit obiectelor stocarea datelor**, **metodele le permit obiectelor îndeplinirea sarcinilor/ acţiunilor şi definesc comportamentul obiectelor.**

Metodele (methods) — **sunt funcţii speciale, care se definesc în interiorul clasei.** Definirea metodei aminteşte de definirea unei funcţiei. După cuvântul rezervat function urmează numele metodei, după care poate urma o listă cu argumente incluse între paranteze rotunde:

Corpul metodei se include între acolade:

```
public  function numeMetoda($arg1, $arg2, ... argN)
{
// ...
}
```

Ca şi proprietăţile, metodele pot fi definite ca fiind publice, protejate şi private.

La definirea metodei ca fiind publică, se va asigura posibilitatea accesării acesteia dinafara obiectului curent. Dacă la definirea metodei se va omite definirea domeniului de vizibilitate, atunci **implicit metoda va fi definită publică.**

ACCESAREA METODEI

În majoritatea cazurilor metoda se apelează prin intermediul variabilei-obiect, după care urmează simbolul '->' şi numele metodei.

La apelarea metodei se folosesc parantezele rotunde, la fel ca şi la apelul unei funcţii (*chiar dacă metodei nu i se transmite nici un argument*).

În exemplul următor la definirea metodei nu a fost utilizat un cuvânt-cheie pentru a specifica domeniul de vizibilitate. Aceasta înseamnă că metoda va fi de **tip public** şi poate fi apelată şi dinafara hotarelor clasei. De asemenea a fost folosită pseudovariabila **$this.**

```
class produs {
        public $denumireProdus = "Cactus";
        public $numeProducator = "Flora SA";
        public $dataProducerii = "12/12/2021";
        public $pretUnitar = 90;

function afiseazaDate()
          {

    Return "{$this->denumireProdus}, {$this->numeProducator}, {$this->dataProducerii}, {$this->pretUnitar}";
          }
}
$produs1 = new produs();
$produs1->denumireProdus = 'Begonie';
$produs1->numeProducator = 'Plante SA';
$produs1->dataProducerii = '01/01/2021';
$produs1->pretUnitar = 70;
// accesarea metodei
echo "Descriere produs: <b>".$produs1->afiseazaDate()."</b><br />";
```

Afişarea rezultatelor:

??????????????????

CONSTRUCTORI

Un constructor este o metodă a **unei clase care este apelată automat în momentul în care este creată o nouă instanţă a clasei** (cu ajutorul operatorului **new**).

Constructorul defineşte valori pentru unele atribute sau poate apela alte metode ale clasei. De asemenea constructorul poate fi utilizat **pentru a realiza toate activităţile** preventive necesare lucrului cu obiectele.

În versiunea 4 PHP numele constructorului trebuia să coincidă cu numele clasei din care face parte.

Începând cu PHP5 numele constructorului trebuie definit folosind construcţia

***__construct*().**

Numele acestei metode începe cu două linii de subliniere. În PHP5 au apărut şi ***destructorii*** – funcţii care ***se apelează automat înaintea distrugerii obiectului***. În PHP5 funcţia-destructor se apelează prin

***__destruct*()**.

Exemplu definire constructor

```
function __construct($denumireProdus, $numeProducator, $dataProducerii, $pretUnitar) {
        $this->denumireProdus = $denumireProdus;
        $this->numeProducator = $numeProducator;
        $this->dataProducerii = $dataProducerii;
        $this->pretUnitar = $pretUnitar;
        }
//Cream obiecte folosind constructorul
$produs1 = new produs('Begonie', 'Plante SA', '01/01/2021', 70);
$produs2 = new produs('Muscata', ' Flori SA', '15/02/2021', 50);
// Apelul metodei
echo "Descriere produs 1: <b>".$produs1->afiseazaDate()."</b><br />";
echo "Descriere produs 2: <b>".$produs2->afiseazaDate()."</b><br />";
```

Explicaţii la exemplul de mai sus

Metoda ***__construct()*** va fi apelată atunci când se va crea obiectul cu operatorul **new.** Valorile tuturor argumentelor enumerate la crearea obiectului vor fi transmise constructorului. Astfel, în exemplu, constructorului i se transmit **denumirea produsului, numele producătorului, data producerii şi preţul.**

Astfel, în metoda-constructor se foloseşte pseudovariabila $this pentru atribuirea valorilor proprietăţilor corespunzătoare obiectului. În PHP, în continuare este susţinută denumirea constructorului cu numele clasei.

Şi dacă nu există necesitatea compatibilităţii cu versiunile mai vechi ale PHP-ului atunci **metoda-constructor se recomandă a fi numită cu *__construct()*.**

MOŞTENIREA (EXTENDS)

Deseori este necesară definirea unor clase cu proprietăţi şi metode asemănătoare. Este comod să fie definită o clasă generală, care poate fi utilizată simultan în câteva proiecte şi adaptată conform cerinţelor pentru fiecare proiect. Pentru a uşura definirea unor astfel de clase a fost introdus conceptul **de extindere (derivare) a claselor.**

***O clasă derivata va păstra toate proprietăţile şi metodele* clasei pe care o extinde şi poate conţine diferite proprietăţi şi metode noi.**

Notă: Nu exista nici o posibilitate de a elimina din clasa derivată anumite proprietăţi sau metode ale clasei de bază – **CLASA URMAŞ POATE FI DOAR EXTINSĂ**.
În PHP o anumită clasă-urmaş poate avea o singură clasă părinte. **Astfel, în PHP nu este permisă moştenirea multiplă.**
Pentru a extinde o anumită clasă în PHP se utilizează cuvântul cheie **extends.**

EXEMPLU EXTINDERE

Vom încerca să descriem omul. Această descriere poate fi făcută diferit, în funcţie de domeniul de utilizare a acestei descriere. **Omul poate fi descris ca fiind student sau ca fiind programator.**

Dacă descriem omul ca şi programator, atunci el cunoaşte anumite limbaje de programare, limbaje de modelare, sisteme de operare etc.

Dar odată ce omul este programator, el nu încetează de a fi om. Adică el are un loc de trai, are un nume, un prenume etc.

Dacă e să încercăm să traducem afirmaţiile de mai sus în noţiuni OO, atunci se poate spune că au fost descrise două clase:

- **clasa oamenilor** şi
- **clasa programatorilor**

– fiecare clasă cu proprietăţile şi metodele proprii.

Este evident că clasa programatorilor posedă toate proprietăţile specifice clasei „om”, dar suplimentar mai posedă şi alte proprietăţi specifice. Adică clasa „programator” este subclasa clasei „om”.

Astfel, dacă un om posedă un nume, prenume – programatorul tot le are. Invers, însă nu este adevărat – dacă un programator ştie un limbaj de programare, aceasta nu înseamnă că orice om trebuie să cunoască acest limbaj de programare.

Înafara clasei programatorilor, în mod analogic, poate fi definită clasa contabililor, economiştilor, managerilor etc. Şi toate aceste clase vor fi subclase ale clasei „om”.

OPERATORUL ::

Uneori în descrierea clasei-urmaş apare necesitatea referirii la funcţiile sau variabile din clasa-părinte.

Această adresare este necesară a fi făcută înaintea creării unui anumit exemplar al clasei părinte. În PHP4 pentru acest tip de referire se foloseşte un operator special - **«::».**

În exemplul următor va fi definită extensia **floare** şi vom defini pentru ea **funcţia set_color()** – pentru a fixa culoarea florii.

În exemplu, va fi utilizat operatorul «::» pentru a apela din clasa părinte produs, metoda afiseazaDate().

În interiorul clasei-urmaş pot fi apelate orice proprietăţi sau metode, definite în clasa-părinte cu ajutorul lui $this, numai dacă clasa-urmaş nu redefineşte aceste proprietăţi sau metode.

Exemplu

```
class produs
{
        public $denumireProdus = "Cactus";
        public $numeProducator = "Flora SA";
        public $dataProducerii = "12/12/2021";
        public $pretUnitar = 90;

    function __construct($denumireProdus, $numeProducator,
$dataProducerii, $pretUnitar)
{
                $this->denumireProdus = $denumireProdus;
                 $this->numeProducator = $numeProducator;
                $this->dataProducerii = $dataProducerii;
                $this->pretUnitar = $pretUnitar;
            }
function afiseazaDate()
{
echo $this->denumireProdus ", " . $this->numeProducator ", " . $this-
>dataProducerii ", " . $this->pretUnitar;
}
            }
class floare extends produs
{
// definim clasa-urmas
var $culoare = "rosu";
// Adaugam culoarea florii

function set_color($new_color)
{
$this->culoare = $new_color;
// apelăm funcţia din clasa-părinte
produs::afiseazaDate();
echo "<b>, " . $new_color. "</b>";
```

```
}
}
$produs3 = new floare('Orhidee', ' Plante SA', '20/02/2015', 250);
echo "Descriere produs 3: <b>";
$produs3->set_color("alba");
echo "</b>";
```

Afişarea rezultatelor:
??????????????????

Pentru definirea unei alte culori: …$produs3->set_color(“galbena");…

OPERATORUL PARENT

În exemplul anterior, adresându-ne la clasa-părinte am folosit numele clasei-părinte - produs::afiseazaDate().

Uneori nu este comodă această adresare, deoarece se poate modifica numele clasei părinte sau ierarhia claselor, şi atunci va fi necesară rescrierea tuturor descrierilor claselor.

Pentru a evita aceste situaţii în locul numelui clasei-părinte, începând cu versiunea PHP5, se recomandă folosirea cuvântului rezervat **parent** - de exemplu, **parent::afiseazaDate().**

Parent face referire la acea clasă, care este specificată după cuvântul rezervat **extends** din definirea clasei urmaş. Şi de aceea dacă ierarhia claselor se va modifica, se va corecta doar numele de după cuvântul **extends**.

Extinderea claselor. Exemplu

```
<?php
class Om
{
	public $numeOm;
	public $locatieOm;
	public $virstaOm;
	public $specialitateOm;

	function __construct($numeOm, $locatieOm, $virstaOm)
	{
	$this->numeOm=$numeOm;
	$this->locatieOm=$locatieOm;
	$this->virstaOm=$virstaOm;
```

```
	}
// aici a fost utilizat this-> este in interiorul clasei Om

function afisareOm($numeOm, $locatieOm, $virstaOm)
	{
		echo "<br>"."Nume  ".$this->numeOm."Localitate
".$this->locatieOm. "Virsta  ".$this->virstaOm;
	}
// Si aici a fost utilizat this-> este in interiorul clasei Om
Fara aceasta referinta nu se transmite nimic
}

class specialistOm extends Om
{
	public $specialitate;
	public $institutiaTerminata;
	public $anulterminstit;

	function set_specialitatea($specialitate)
	{
		$this->specialitate=$specialitate;
		Om::afisareOm($numeOm, $locatieOm, $virstaOm);
		echo "<br>"."Specialitatea ".$specialitate;
	}
}
echo "</b>";

// adresare si formare obiect din subclasa
$Omspecialist = new specialistOm("Ion","Aneni","30"); - aici
datele initiale se transmit Clasei de baza Om care mai apoi sunt
folosite in subclasa..
echo "Descriere Om specialist: <b>";
// adresare la metoda din subclasa
$Omspecialist->set_specialitatea("Programator");
echo "</b>";
?>
```

REZULTATUL

Descriere Om specialist:
Nume Ion Localitate Aneni Virsta 30
Specialitatea Programator