

## Programarea Obiect Orientată (POO) în limbajul PHP

Pentru aplicațiile de mari dimensiuni, o dezvoltare structurată a codului (orientată pe funcții/proceduri), implicând existența unui număr foarte mare de linii de cod, prin modul de organizare a codului, conduce la mari dificultăți privind realizarea unor modificări ulterioare în cadrul aplicației. Programarea orientată pe obiecte a apărut ca o necesitate în contextul creșterii complexității codului aplicațiilor software.

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

1.2. Principii POO - Încapsulare

- Accesul la atributele unui obiect se face doar prin apelarea metodelor sale prin interfețe sau prin mesaje, orice alt obiect neștiind nici măcar de existența atributelor obiectului în cauză.
- Metodele formează interfața clasei, iar mesajele reprezintă cereri adresate obiectului pentru a returna o valoare sau pentru a-și schimba starea.

1

## ELEMENTELE DE BAZĂ ÎN POO

**Clasa și obiectul** – reprezintă noțiunile de bază în metoda OO.

**Obiectul** – este o reprezentare a unei entități din lumea reală *asupra căruia se poate întreprinde o acțiune sau care poate întreprinde o acțiune.*

Sau, altfel spus, *obiectul este o variabilă structurată*, care conține toată informația necesară referitoare la *o entitate fizică* sau *o noțiune realizată prin coduri de program.*

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

**Prin instanțiere a unei clase** este creat un obiect (*evident se pot face instanțieri multiple, construindu-se mai multe obiecte ale aceleiași clase*)

## PHP și POO

PHP dispune de posibilități OO foarte generale. PHP *nu este un limbaj de programare OO dedicat*, așa cum sunt de exemplu C++ sau Java. Dar principiile de bază ale POO sunt susținute și în PHP:

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

**Încapsularea** - înseamnă *gruparea datelor și a operațiilor* asupra acestor date *în același întreg (agregat)*, *ascunzându-se detaliile de implementare* (proiectare-realizare) ale acestui întreg.

**Moștenirea** - permite *unei clase* sa *moștenească atributele și metodele unei alte clase existente*.

**Polimorfismul** - *unele* dintre *proprietățile și metodele definite in superclasă pot fi redefinite (rescrise) in subclasele de obiecte derivate*.

## DEFINIREA CLASEI ÎN PHP

În PHP clasa se definește cu ajutorul cuvântului rezervat **class** și **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
{
  /*BLOC PROPRIETĂȚI*/
  var /vizibilitate $denumire_proprietate;
  /*listă proprietăți*/

  /*BLOC METODE*/
  function denumire_metodă()
  {
    /* Definire metodă */
  }
  /*listă metode*/
}
```

## DEFINIREA OBIECTULUI

Dacă despre **clasă** se poate spune că **are rol de șablon** în crearea obiectelor, atunci **obiectul reprezintă date structurate conform șablonului, 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, *instanță*, definit în script, i se pune în corespondență un **nume unic**, pentru a-l identifica. Acest nume este unic de-a lungul vieții obiectului.

## ELEMENTELE / MEMBRII CLASEI

### ÎN CADRUL CLASEI SUNT PRECIZATE:

A. **atributele sau proprietățile** – partea de date a *obiectului, reprezentate prin declarații de variabile, inclusiv* **posibile inițializări** ale acestora.

*Proprietățile se definesc folosind cuvântul cheie var (până la versiunea 5) sau* **specificându-se vizibilitatea proprietății** (după versiunea 5)

B. **metodele** – partea de cod a **clasei**, reprezentate prin **funcții** ce definesc comportamentul obiectelor. Totodată constituie și **interfața** obiectului *destinată manipulării datelor* acestuia.

**Pentru a obține acces la membrii clasei – proprietăți și metode (funcții) – în interiorul clasei, poate fi folosită referința \$this, care se referă mereu la elementul curent.**

## DEFINIREA PROPRIETĂȚILOR CLASEI

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

*Definirea proprietății în clasă se aseamănă cu definirea unei variabile, plasându-se în față cuvântul rezervat var sau, începând cu versiunea 5, în față 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 și din ce context.*

Cuvintele cheie **public, protected și private**, care *definesc domeniul de vizibilitate ale proprietăților și metodelor (funcției)*, au apărut în PHP 5.

În PHP 4 toate proprietățile trebuie definite folosind cuvântul cheie **var**, ceea ce este *identic cu folosirea cuvântului cheie public.*

**Nită:** Reieșind din principiul compatibilității inverse, *în PHP5 se admite utilizarea cuvântului cheie `var` la definirea proprietăților, în locul cuvântului rezervat `public`.*

**Domeniul de vizibilitate** definește **contextul funcției sau a proprietății**, în cadrul căruia poate fi utilizată variabila sau metoda.

Astfel,

- **variabila definită în interiorul corpului funcției, are un domeniu de vizibilitate local, iar**
- **variabila definită în afara funcției — posedă un domeniu de vizibilitate global.**

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

### **Exemplu definire proprietăți**

```
class produs
```

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



### **ACCESUL LA PROPRIETĂȚI**

**Pentru a accesa membrii clasei (proprietati și metode) în PHP se folosește OPERATORUL “->”.**

**ADRESAREA SE FACE CUNOSCÂND NUMELE OBIECTULUI ȘI NUMELE PROPRIETĂȚII.**

*Dacă proprietățile sunt definite ca și „publice” pot fi accesate valorile acestor proprietăți sau pot fi atribuite noi valori proprietăților, modificând astfel valorile standard definite în clasă, la inițializare.*

### **Exemplu accesare proprietăți**

```
class produs
```

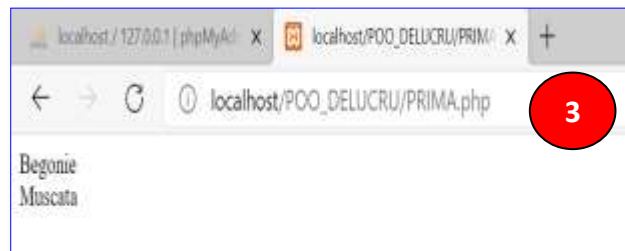
```
{
```

```

public $denumireProdus = "Cactus";
public $numeProducator = "Flora SA";
public $dataProducerii = "12/12/2021";
public $pretUnitar = 90;
}
$produs1 = new produs();
$produs2 = new produs();
ADRESAREA SE FACE CUNOSCÂND NUMELE OBIECTULUI "->". ȘI
NUMELE PROPRIETĂȚII.
// Modificăm valorile proprietăților
$produs1->denumireProdus = 'Begonie';
$produs2->denumireProdus = 'Muscata';
// Afișăm valorile proprietăților modificate
echo $produs1->denumireProdus."<br />";
echo $produs2->denumireProdus;

```

### Rezultat:



### Accesarea mai multor proprietăți:

```

$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 />";

```

### Afișarea rezultatelor:



## DEFINIREA METODELOR CLASEI

Așa cum proprietățile permit obiectelor stocarea datelor, metodele le permit obiectelor îndeplinirea sarcinilor/acțiunilor și definesc comportamentul obiectelor.

**Metodele (methods)** — sunt funcții speciale, care se definesc în interiorul clasei. Definirea metodei amintește de definirea unei funcții. 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 și proprietățile, metodele pot fi definite ca fiind publice, protejate și 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 '->' și numele metodei, adică *\$nume\_variabila\_obiect -> nume\_metoda()*.

La apelarea metodei se folosesc parantezele rotunde, la fel ca și la apelul unei funcții (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* și poate fi apelată și dinafara hotarelor clasei.

De asemenea a fost folosită *pseudovariabila \$this*. Un avantaj al OOP este ca nu mai este nevoie sa folosim *global* pentru a face *variabile (proprietati) globale* in functii, *le vom apela pe cele ale clasei* cu *\$this->nume\_variabila*

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

    function afiseazaDate()
```

```

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

```

### Afișarea rezultatelor:



## CONSTRUCTORI

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

*Constructorul definește valori pentru unele atribute sau poate apela alte metode ale clasei.*

De asemenea *constructorul* poate fi utilizat pentru a realiza toate activitățile preventive necesare lucrului cu obiectele.

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

**Începând cu PHP5 numele constructorului trebuie definit folosind construcția**

***\_\_construct()***.

Numele acestei metode începe cu două linii de subliniere. În PHP5 au apărut și ***destructorii*** – funcții care *se apelează automat înaintea distrugerii obiectului*.

În PHP5 funcția-destructor se apelează prin

***\_\_destruct()***.

### Exemplu definire constructor

```

function __construct($denumireProdus, $numeProducator, $dataProductiei,
$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 />";
```

**Rezultat:**



**Explicații 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 și prețul.

**Astfel, în metoda-constructor se folosește pseudovariabila \$this pentru atribuirea valorilor proprietăților corespunzătoare obiectului.**

**Nota:**

**REAMINTIM!!!** Un avantaj al OOP este ca nu mai este nevoie sa folosim global pentru a face variabile (proprietati) globale in functii, le vom apela pe cele ale clasei cu `$this->nume_variabila`

În PHP, în continuare este susținută denumirea constructorului cu numele clasei.

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

**MOȘTENIREA (EXTENDS)**

**Deseori este necesară definirea unor clase cu proprietăți și metode asemănătoare.**

Este comod să fie definită o clasă generală, care poate fi utilizată simultan în câteva proiecte și adaptată conform cerințelor pentru fiecare proiect. Pentru a ușura definirea unor astfel de clase a fost introdus conceptul **de extindere (derivare) a claselor**.



***O clasă derivată va păstra toate proprietățile și metodele clasei pe care o extinde și poate conține diferite proprietăți și metode noi.***

**Notă:** Nu exista nici o posibilitate de a elimina din clasa derivată anumite proprietăți sau metode ale clasei de bază – **CLASA URMAȘ POATE FI DOAR EXTINSĂ.**

**Notă:** În PHP o anumită clasă-urmaș poate avea o singură clasă părinte. **Astfel, în PHP nu este permisă moștenirea multiplă.**

Pentru a extinde o anumită clasă în PHP se utilizează cuvântul cheie **extends**.

## **EXEMPLU EXTINDERE**

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

**Dacă descriem omul ca și programator,** atunci el cunoaște 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 afirmațiile de mai sus în noțiuni OO, atunci se poate spune că au fost descrise două clase:

- **clasa oamenilor și**
- **clasa programatorilor**

– fiecare clasă cu proprietățile și metodele proprii.

Este evident că **clasa programatorilor** posedă toate proprietățile specifice **clasei „om”**, dar suplimentar mai posedă și alte proprietăți 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 știe un limbaj de programare, aceasta nu înseamnă că orice om trebuie să cunoască acest limbaj de programare (prin aceasta se explica extinderea!!!).**

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

**Mai jos**

## **OPERATORUL ::**

**Uneori în descrierea clasei-urmaș apare necesitatea referirii la funcțiile 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 folosește un **operator** special - **<<::>>**.

În exemplul următor va fi definită extensia **floare** și vom defini pentru ea **funcția** `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-urmas pot fi apelate orice proprietăți sau metode, definite în clasa-părinte cu ajutorul lui \$this,** numai dacă clasa-urmas nu redefinește aceste proprietăți sau metode.

### **Exemplu**

```
<?php
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

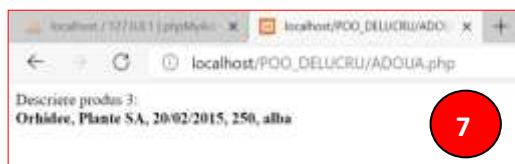
function set_color($new_color)
{
$this->culoare = $new_color;
// apelăm funcția din clasa-părinte
produs::afiseazaDate();
```

```

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

```

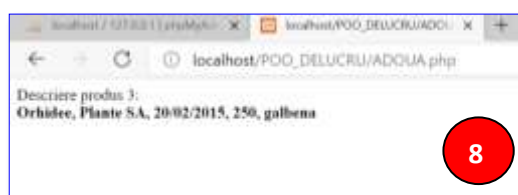
### **Rezultat**



```

// CREAREA UNEI INSTANTE
$produs3 = new floare('Orhidee', ' Plante SA', '20/02/2015', 250);
echo "Descriere produs 3: <b>". "<br/>";
$produs3->set_color("galbena");
echo "</b>";

```



## **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, și atunci va fi necesară rescrierea tuturor descrierilor claselor.

Pentru a evita aceste situații 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 definiția clasei urmaș.

Și de aceea dacă ierarhia claselor se va modifica, se va corecta doar numele de după cuvântul **extends**.

## Extinderea claselor.

### EXEMPLU 1 TREI.PHP

```
<?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 $anultermininstit;

    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>";

```

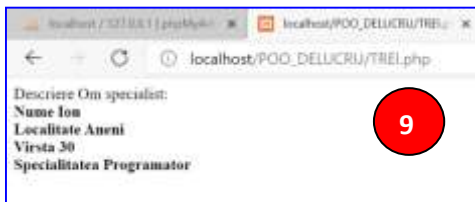
**// adresare la metoda din subclasa**

```

$Omspecialist->set_specialitatea("Programator");
echo "</b>";
?>

```

**REZULTATUL**



**Merge? A mers?**

**<?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."<br/>". "Localitate ".$this->locatieOm.
    "<br/>". "Virsta ".$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 $institutieTerminata;
    public $anulTerminat;

    function set_specialitatea($specialitate)
    {
        $numeOm=$this->numeOm;
        $locatieOm=$this->locatieOm;
        $virstaOm=$this->virstaOm;
        $this->specialitate=$specialitate;
        parent::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>";
?>
```

**Practic, \$XXX acționează ca ceva local, variabilă care urmează a fi utilizată în metoda curentă. \$ this-> XXX acționează ca o variabilă de instanță, care este disponibilă oriunde în clasă.**

## **EXEMPLU 2 /DOIOM.PHP/**

```
<?PHP
class OM
{
    public $name="Ghita";
    public $age = 27;
// metoda construct
    public function __construct()
```

```

{
    echo "Adresarea la constructor in clasa de baza";
    echo "<br>";
}
//definirea numelui
public function setName($name)
{
    $this->name = $name;
    return $this->name;
}
// Lansarea destructorului
public function __destruct()
{
    echo "Lansarea destructorului";
    echo "<br>";
}
}
//Extinderea clasei
class Student extends OM
{

    public $limbaj_de_programare = "Cunoaste limbajul de programare PHP";
    public $nume;
// Constructorul in clasa extinsă
    public function __construct($name)
    {
        echo "Lansarea constructorului în clasa extinsă" . "<br>";
        echo "<br>";
        parent::__construct();
        $this->nume = $name;
    }

    public function setName1($name)
    {
        $this->name = "Iata-l pe studentul".$name;
        $this->nume=$this->name;
        return $this->nume;
    }
}
//instantierea unui obiect Om1
$Om1 = new OM();
$Om1->setName("Petru");
echo "<br>";

```

```
echo "Este vorba de Omul <b>".$Om1->setName("Petru").", ".$Om1-  
>age."</b><br />";  
echo "<br>";  
//instantierea unui obiect Om2  
$Om1 = new OM();  
$Om1->name;  
echo "<br>";  
echo "Este vorba de Omul <b>".$Om1->name.", ".$Om1->age."</b><br />";  
echo "<br>";  
//instantierea unui obiect al clasei Student, extindere a clasei OM  
$student = new Student("Petru");  
echo "<br>";  
echo $student->nume."<br />".$student->limbaj_de_programare."</b><br />";  
echo "Studentul are virsta =", $student->age;  
echo "<br>";  
//instantierea unui obiect al clasei Student, extindere a clasei OM  
$student = new Student("Petru");  
echo "<br>";  
echo $student->setName1("Ghita")."<br />".$student->  
>limbaj_de_programare."</b><br />";  
echo "Studentul are virsta =", $student->age;  
echo "<br>";  
?>
```