Web2, curs10, Schita notiuni

I. OOP – Programarea orientata pe obiecte

1. OOP are la baza ideea organizarii unor concepte din lumea reala in anumite entitati separate - structuri de date definite de programator. O astfel de structura este denumita **clasa** si contine atat datele cat si actiunile legate de acel concept.
2. Termeni:
   - **clasa**
     - structura definita de catre programator; este "proiectul" care sta la baza construirii obiectelor
     - contine **atribute** sau **proprietati** (variabile) si **metode** (functii)
     - numele de clase nu sunt case-sensitive in PHP
   - **obiect**
     - o instanta a unei clase
     - numele de obiecte sunt case-sensitive in PHP
   - **atribute** sau **proprietati**
     - datele, sau variabilele definite in clasa respectiva. In general, fiecare obiect are propriile valori pentru aceste variabile. Similar cum diferite persoane au valori diferite pentru nume, prenume, varsta, etc.
   - **metode**
     - actiunile definite de clasa respectiva, adica functiile definite in clasa. Astfel, spre deosebire de programarea procedurala, aici functiile nu sunt de sine statatoare, ele sunt definite in interiorul unei clase si vor fi apelate doar prin intermediul unui obiect din clasa respectiva. (uneori prin intermediul clasei).
     - Pentru ca numele de functii sunt case-sensitive in PHP, si numele de metode sunt case-sensitive.
   - **Modificator de acces** poate fi
     - **public** – atributul/metoda este vizibil complet in afara clasei
     - **protected** - atributul/metoda este vizibil doar in clasele derivate (ce mostenesc) din aceasta clasa
     - **private** - atributul/metoda nu este vizibil decat in alte metode din cadrul aceleiasi clase. Acest membru este "ascuns" in interiorul clasei, nu poate fi accesat din exterior.
     - daca modificatorul de acces pentru o metoda nu este specificat, se considera implicit public.
3. Sintaxa de declarare a unei clase:

```
class ClasaMea{
        <modificator_acces> atribut_1 [=valoare];
        <modificator_acces> atribut_2 [=valoare];
        ...
        <modificator_acces> atribut_n [=valoare];
        [<modificator_acces>] function metoda_1([lista_parametri]){
        //instructiuni
        }
        ...
        [<modificator_acces>] function metoda_m([lista_parametri]){
        //instructiuni
        }
}
```

Atributele unei clase sunt definite inaintea metodelor.
Daca atributele primesc valori la declarare, aceste valori treuie sa fie constante, nu rezultatul unei expresii.

```
class ClasaMea{
public $var1 = 123; // asa este ok
public $var2 = $num * $num; // asa nu este ok
}
```

4. Crearea unui obiect sau instantierea clasei se face folosind cuvantul cheie ***new***. Pentru a crea un obiect din clasa *ClasaMea, s*e poate folosi una dintre urmatoarele variante:

```
$obiect1= new ClasaMea();
$obiect2=new ClasaMea;
```

5. Accesarea atributelor si metodelor unei clase, se face folosind operatorul sageata →.

$obiect1->atribut_1
$obiect1->metoda_1
Obs:

1. *Atributele si metodele unei clase nu pot fi accesate direct ci numai prin intermediul unui obiect al clasei respective.*

2. *Pentru ca o metoda a unei clase sa poata accesa un atribut din aceeasi clasa, atributul va fi accesat cu ajutorul unei variabile speciale,* ***$this***

```
class ClasaMea {

        public $var;

        function do() {

                echo $var;// Asa nu va merge:
                echo $this->var;//Asa va merge
        }

}
```

6. Exemplu de clasa:

```
class Persoana{

        public $nume;

        public $prenume;


        public function spuneNume(){

                echo $this->nume.' '.$this->prenume;

                }



}
```

7. Instantierea clasei:

```
//creez obiectul $persoana1

$persoana1=new Persoana();



//dau valori atributelor

$persoana1->nume="Popa";

$persoana1->prenume="Ion";



//afisez numele si prenumele
$persoana1->spuneNume();
```

Obs:

3. *Este recomandat sa se creeze un fisier separat care sa contina definitiile de clase. In proiecte mari,*

fiecare clasa va fi definita intr-un fisier separat.

4. *Variabila $this se foloseste pentru a accesa proprietati sau pentru a apela metode ale clasei curente. Atentie la semnul $: $this->nume nu ~~$this->$nume~~*

5. *Cand creati un obiect, nu puneti intre ghilimele numele clasei: ~~$pers="new 'Persoana';~~*

8. Metodele getter si setter
   Deseori, in programarea orientata obiect, este de preferat sa stabilim metode separate pentru citirea sau modificarea anumitor atribute din clasa. Astfel, codul client (partea din cod din afara clasei ce foloseste aceasta clasa) nu va avea acces direct la atributele respective, care in general sunt private, ci prin intermediul acestor metode standard, numite "getter" si "setter".
   - metode de tip **getter** - denumite si **accesor** - sunt destinate citirii unui atribut declarat *private*, care bineinteles nu poate fi accesat in mod direct din afara clasei. Denumim aceste metode **getAtribut()** si in general nu primesc argument, si returneaza valoarea atributului respectiv
   - metode de tip **setter** - denumite si **mutator** - sunt destinate modificarii unui atribut *private*. Aceste metode se denumesc **setAtribut($valoare)**, primind ca parametru valoarea cu care modificam atributul respectiv
9. Setarea acestor metode se face de catre programator, atunci cand simte nevoia sa ofere un acces controlat la anumite atribute ale clasei respective. La accesarea unui astfel de atribut, cu o metoda tip *getAtribut()*, programatorul poate alege sa formateze, verifice sau sa schimbe forma atributului inainte sa-l returneze codului client. Metodele de tip setAtribut($valoare) realizeaza o filtrare a valorii primite pentru setarea atributului respectiv. Programatorul clasei are posibilitatea prin **setters** sa verifice validitatea datelor primite, sa le respinga, sa genereze o eroare, sau sa le modifice.
10. Clasa *Persoana* cu metode de tip **getter, setter**

```
class Persoana{

        public $nume;

        public $prenume;


        function getNume(){

                return $this->nume;

                }


        function getPrenume(){

                return $this->prenume;

                }


        function setNume($nume){

                $this->nume=$nume;

                }


        function setPrenume($prenume){

                $this->prenume=$prenume;
```

```
                    }
}
//creez obiectul $persoana1
$persoana1=new Persoana();

//dau valori atributelor
$persoana1->setNume("Popa");
$persoana1->setPrenume("Ion");

//afisez valorile atributelor
echo 'Salut '.$persoana1->getNume().' '.$persoana1->getPrenume();
```

11. Metoda constructor
    - trebuie denumita *__construct* (precedata de doua caractere underline),
    - se executata **la crearea unui obiect**
    - in general initializeaza obiectele nou create
    - poate seta si valori implicite pentru atribute ale obiectului
    - daca folosesc constructor, scriu mai putin cod pentru ca pot renunta la **setter**
12. Metoda destructor
    - este optioala
    - se executata in momentul cand se sterg referintele catre un obiect (cu functia unset() de exemplu)

13. Clasa *Persoana* cu metoda constructor:

```
class Persoana{
        public $nume;
        public $prenume;

        function __construct($nume,$prenume){
                $this->nume=$nume;
                $this->prenume=$prenume;
                }

        function getNume(){
                return $this->nume;
```

```
                }


        function getPrenume(){

                return $this->prenume;

                }

}

//creez obiectul $persoana1

$persoana1=new Persoana('Popa','Ion');


//afisez valorile atributelor

echo 'Salut '.$persoana1->getNume().' '.$persoana1->getPrenume();
```

14. Constantele unei clase
    - similar cu constantele de sine statatoare, contin valori ce nu se schimba.
    - se declara astfel: *const numeConstanta; Numele constantei nu este precedat de $!*
    - sunt initializate obligatoriu in momentul cand sunt definite, cu o valoare constanta (nu expresie, variabila, etc) si de un tip scalar (deci nu array)
    - nu sunt precedate de modificatorii de acces, sunt automat publice
    - constantele clasei exista intr-un singur exemplar, avand aceeasi valoare (si loc de memorie) pentru fiecare obiect al clasei.
    - pot fi accesate cu operatorul ::
        - Din afara clasei prin *NumeClasa::numeConstanta*.
        - Din interiorul clasei cu *self::numeConstanta*
          self este un cuvant cheie ce reprezinta clasa curenta

15. Atribute si metode statice
    - Cuvantul cheie **static** poate fi folosit atat in fata atributelor cat si in fata metodelor, acestea devenind "statice".
    - Atributele statice mai sunt numite si "class variables" (variabilele clasei),
    - Metodele statice mai sunt denumite "class functions" (functiile clasei), asta pentru ca termenul de "static" se refera la apartenenta la clasa, si nu la instantele (obiectele) create din clasa respectiva.
    - **Atributele statice** apartin clasei, nu obiectelor create, si sunt deci apelate prin intermediul numelui clasei, astfel:
        - ***NumeClasa::$numeAtribut*** - din afara clasei
        - ***self::$numeAtribut*** - din interiorul clasei. **self** fiind un cuvant cheie ce desemneaza clasa curenta
    - **Metodele statice** pot fi apelate atat prin sintaxa asemanatoare atributelor statice, (***NumeClasa::metoda()*** sau ***self::metoda()*** din interiorul clasei) , insa pot fi apelate si prin intermediul unui obiect. Metodele statice nu au acces la atribute si metode ce apartin obiectului curent (folosind sintaxa $this), dar au acces la alte atribute si metode statice.

Web2, curs10, Schita notiuni

OOP. Incapsulare. Mostenire. Polimorfism

1. ***Incapsularea*** (engleza: *encapsulation*) este proprietatea obiectelor de a-si ascunde o parte dintre proprietati si metode.
    - Din exteriorul obiectului sunt accesibile ("vizibile") numai proprietatile si metodele ***publice (public)***.
    - Proprietatile si metodele ***private (private)*** sunt accesibile numai din interiorul clasei.
    - Proprietatile si metodele ***protejate (protected)*** sunt accesibile din interiorul clasei, dar si din interiorul claselor derivate.

```
<?php

class Persoana {

        public $nume;

        public $inaltime;

        protected $asigurareViata;

        private $codPin;


        function __construct($numePersoana) {

                $this->nume = $numePersoana;

        }


        function setNume($numeNou) {

                $this->nume = $numeNou;

        }


        function getNume() {

                return $this->nume;

        }

}

?>
```

2. **Mostenirea** (engleza: *inheritance*)
    - este o caracteristica OOP prin care se poate folosi o clasa (***clasa de baz***a) ca baza pentru definirea altei/altor clase (***subclase***)
    - trecerea de la clasa la subclasa se face prin adaugarea de atribute si/sau de metode
    - se spune ca o subclasa ***mosteneste*** clasa de baza sau este ***derivata*** din aceasta
    - de exemplu, clasa *Cursant* si clasa *Angajat* au ambele toate atributele si metodele clasei *Persoana*, dar fiecare din acestea are si atribute si/sau metode specifice.
    - De ce sa folosesti mostenirea?
        - Iti permite sa reutilizezi eficient codul clasei de baza: un *Cursant* este o *Persoana.* Clasa *Cursant*

va mosteni atributele si metodele public si protected ale clasei *Persoana;* vor fi descrise numai atributele si/sau metodele specifice.

- Codul clasei de baza va fi scris o singura data, intr-un singur loc si poate fi reutilizat de mai multe ori (mai eficient decat *include*)