Programare PHP orientată pe obiecte

Programarea orientată pe *obiecte* (OOP –*Object Oriented Programming*) a apărut ca o necesitate în contextul creşterii complexităţii codului aplicaţiilor software. Pentru aplicaţiile de mari dimensiuni, o dezvoltare structurată a codului (orientată pe funcţii/proceduri) implicând existenţa unui număr foarte mare de linii de cod (uneori puternic redundant), prin modul de organizare a codului conduce la o lizibilitate scăzuta a acestuia şi implicit, la mari dificultăţi privind realizarea unor modificări ulterioare în cadrul aplicaţiei.

OOP oferă o modalitate diferită de organizare a codului şi a datelor în cadrul unui program. Din acest punct de vedere, elementele constructive de cod specifice OOP sunt clasa, respectiv obiectul. Clasa reprezintă definiţia unui obiect ("planul obiectului"). Prin instanţierea unei clase este creat un obiect (evident se pot face instanţieri multiple, construindu-se mai multe obiecte ale aceleiaşi clase). În cadrul clasei (definiţia obiectului) sunt precizate:

- atribute sau proprietăţi – (practic partea de date a obiectului) reprezentate prin declaraţii de variabile, inclusiv posibile iniţializări ale acestora;

- metode – (partea de cod a obiectului) reprezentate prin funcţii (sauproceduri) constituind totodată şi interfaţa obiectului destinata manipulării datelor acestuia.

OOP este fundamentată pe 3 principii de bază:

- încapsulare – fiecare obiect este de sine stătător şi complet autonom (conţinând atât date -proprietăţi-, cât şi cod –metode). Un obiect are o interfaţă clară, bine definită, folosită pentru a manipula obiectul;

- moştenire;

- polimorfism.

Particularizat la PHP, în continuare vom trata următoarele aspecte:

- Crearea unei clase (proprietăţi, metode)

- Crearea unui obiect (instanţierea clasei)

- Utilizarea proprietăţilor obiectului

- Apelul metodelor obiectului

- Moştenirea

- Polimorfismul

În limbajul PHP, crearea unei clase se face utilizând *class*. O definire minimala a unei clase, este urmatoarea:

```
class denumire_clasa {
...
}
```

În vederea atribuirii unei funcţionalităţi clasei este necesară definirea unor proprietăţi şi metode. Proprietăţile se creează prin declararea unor variabilelor la începutul definiţiei unei clase folosind instrucţiunea *var*. Următoarea secvenţă PHP creează o clasă denumită *Clasa1* având două atribute (proprietăţi): *$message, $data* (ultimul fiind şi iniţializat).

```
class Clasa1 {
var $message;
var $data=”initializat”;
}
```

Metodele se creează prin declararea unor funcţii PHP în definiţia clasei. Codul metodei *setMessage*, având un parametru de intrare (*$param*) şi permiţând o setare a valorii proprietăţii *$message*. De remarcat că, în cadrul definiţiei clasei, referirea unei proprietăţi a acesteia se face folosind operatorul *$this* care precede numele proprietăţii (nume utilizat fără *$* în faţă). În cazul de fata: *$this->message*.

- metoda *getMessage*, fără parametrii de intrare, care afişează un mesaj, respectiv returnează valoarea proprietăţii *$message*.

```
<?php
class Clasa1
{
var $message;
var $data=”initializat”;
```

```
function setMessage($param)
{
$this->message = $param;
}
function getMessage()
{
echo "Mesajul pentru obiectul 1 este:<br>";
return $this->message;
}
}
// . . .cod PHP instanţiere clasă
?>
```

După ce a fost creata clasa, ne intereseaza modul de instanţiere a clasei în vederea creării unui obiect, precum şi modul de setare a proprietăţilor acestuia și de apel al metodelor (prin completarea secvenţei anterioare, după încheierea definiţiei clasei):

Exemplu

```
$Obiect1 =new Clasa1();
$Obiect1->setMessage("Setarea proprietate folosind o metoda");
echo $Obiect1->getMessage();
$var=$Obiect1->getMessage();
echo $var;
$Obiect1->message="Setare directa a proprietatii printr-o atribuire ";
echo $Obiect1->getMessage();
echo $Obiect1->data;
```

Rezultatul rulării scriptului este următorul:

Mesajul pentru obiectul 1 este:

Setarea proprietate folosind o metoda

Mesajul pentru obiectul 1 este:

Setarea proprietate folosind o metoda:

Mesajul pentru obiectul 1 este:

Setare directa a proprietatii printr-o atribuire

initializat

Ulterior declarării unei clase, este necesara crearea un obiect cu care acesta să opereze. Operaţia este denumita instanţierea unei clase sau crearea unei instanţe. În limbajul PHP pentru aceasta operaţie, se utilizează instrucţiunea, cuvântul cheie new.

- Deci prima linie din secvenţa de cod anterioară creează un nou obiect Obiect1 prin instanţierea clasei Clasa1.

- În continuarea se face un apel al metodei setMessage ($Obiect1->setMessage) pasându-i un parametru, metoda care permite setarea proprietăţii message. Linia următoare apelează metoda getMessage (fără parametrii), care afişează un mesaj şi returnează o valoare (a proprietăţii message, afişată pe ecran folosind echo).

- Valoarea returnată putea fi evident memorată într-o variabilă ($var în cazul de faţă).

- În acest caz, setarea proprietăţii message a fost realizată prin apelul unei metode. Setarea unei proprietăţi a unui obiect se poate face şi prin atribuirea direct a unei valori către proprietatea referită ($Obiect1->message="Setare directa a proprietatii") sau chiar printr-o iniţializare în definiţia clasei (vezi proprietatea $data). Toate comenzile de afişare (echo) au fost utilizate doar pentru a evidenţia modul de referire a atributelor/metodelor obiectului.

Observaţie: În cazul limbajul PHP nu se limitează accesul la proprietăţi. Implicit toate proprietăţile sunt de tip public, neputând fi declarate private sau protected.

Orice proprietate declarată în definiţia clasei poate fi referită în exteriorul ei printr-o construcţie de tipul: $Obiect->nume_proprietate;

O metodă specială a unei clase este aşa numita metodă constructor. O metoda constructor are acelaşi nume cu al clasei, fiind apelata automat la crearea unui obiect (realizând operaţii de iniţializare, crearea de alte obiecte necesare obiectului în cauza etc.). Un constructor se declară similar cu celelalte metode, singura deosebire fiind ca are acelaşi nume ca şi clasa. În PHP definirea unei metode constructor nu este obligatorie. Clasa anterior creata nu avea definit un constructor (unele proprietăţi fiind însă iniţializare direct odată cu declararea lor).

Pentru exemplificare, se va crea un constructor pentru Clasa1, care va afişa un mesaj şi va iniţializa proprietatea $message. Definiţia anterioară a clasei se va completa cu încă o metodă (având acelaşi nume cu al clasei):

```
function Clasa1()
{
echo "Obiect creat<br>";
$this->message = "initializare_obiect1";
}
```

O simplă secvenţă de creare a obiectului va apela imediat metoda constructor:

```
$Obiect1 =new Clasa1();
echo $Obiect1->message;
```

iar rezultatul va fi:

Obiect creat

initializare_obiect1

Pentru ca o clasă deja creată să poate fi instanţiată în mai multe scripturi PHP, fără a se face o replicare (copiere) a codului clasei în fiecare script, de regulă definiţiile claselor sunt păstrate în fişiere distincte de cele în care sunt create obiectele. Pentru cazul de faţă, spre exemplu, într-un fişier **clase.php** este salvată definiţia clasei, în timp ce într-un alt fişier PHP (obiect.php spre exemplu) se face instanţierea clasei (şi utilizarea obiectului):

```
<?php
// fişier obiect.php
include("clase.php");
$Obiect1 =new Clasa1();
```

```
echo $Obiect1->data;
?>
```

Evident fişierul clase.php poate conţine definiţiile mai multor clase.

O caracteristică importantă a OOP o reprezintă moştenirea care permite crearea unei relaţii ierarhice între clase, folosind subclase. O subclasa, practic moşteneşte proprietăţile şi metodele unei superclase. Prin intermediul moştenirii, pe lângă elementele moştenite (proprietăţi şi metode), în cadrul noii clase se pot construi şi adăuga noi elemente (proprietăţi sau metode). Astfel, pornind de la clase de baza simple se pot deriva clase mai complexe şi mai specializate pentru o anumită aplicaţie. Utilizarea moştenirii creşte gradul de reutilizare şi lizibilitate al codului sursă, avantaj important al OOP (reducându-se substanţial munca programatorului în cazul în care aceleaşi metode pot fi scrise doar o singură dată într-o superclasă, în loc să fie scrise de mai multe ori în subclase separate). Pe baza superclasei Clasa1 (în acelaşi fişier script), se construieşte prin derivarea acesteia o nouă subclasa Clasa2 (utilizând cuvântul cheie extends), care moşteneşte toate proprietăţile şi metodele superclasei, dar în acelaşi timp are şi alte noi proprietăţi şi metode:

```
class Clasa2 extends Clasa1
{
var $message2="gama";
function getMessage()
{
echo "mesajul nou pentru obiectul 2 este<br>";
return $this->message2;
}
function plus()
{
echo "<br>ceva nou<br>";
}
}
```

De remarcat că, în cadrul subclasei Clasa2 se defineşte o metodă având un nume similar cu al unei metode din superclasa (getMessage), realizându-se în cadrul subclasei o suprascriere a metodei moştenite, precum şi o nouă metodă (plus). De asemenea, noua clasă are şi o proprietate în plus ($message1). O secvenţă de instanţiere şi utilizare a obiectelor subclasei se poate face prin liniile de cod următoare:

```
$Obiect2 =new Clasa2();
$Obiect2->plus();
$Obiect2->setMessage("beta");
echo $Obiect2->message."<br>";
echo $Obiect2->getMessage();
```

are rezultatul următor:

Obiect creat

ceva nou
beta

mesajul nou pentru obiectul 2 este

gama

Crearea obiectului prin instanţierea subclasei Clasa2 conduce şi la moştenirea constructorului corespunzător (afişându-se Obiect creat). Apelul metodei plus conduce la afişarea mesajului ceva nou. Apelul metodei moştenite setMessage permite setarea proprietăţii moştenite message. Apelul metodei suprascrise getMessage (plasat într-un echo), conduce la afişarea ultimelor două rânduri.

Observaţie:

Mecanismul de moştenire funcţionează într-un singur sens, subclasa (copil) moşteneşte de la superclasa (părinte). Orice modificare a clasei părinte este automat preluată şi de clasa copil.

O altă caracteristică importantă a OOP o reprezintă polimorfismul, prin intermediul căruia clase diferite pot conţine metode cu acelaşi nume, dar cu comportamente diferite. Chiar în exemplul cu moştenire, superclasa respectiv subclasa conţin metode cu acelaşi nume (getMessage), dar funcţionalităţi diferite.

Se consideră încă un exemplu. În acelaşi script cu definiţia clasei Clasa1, fie definiţia unei clase Clasa3:

```
class Clasa3
{
var $message;
function setMessage($param)
{
$this->message = $param;
echo "Clasa 3";
return $this->message;
}
}
```

La o instanţiere a claselor şi folosire a obiectelor create:

```
$Obiect1 =new Clasa1();
$Obiect3 =new Clasa3();
$Obiect1->setMessage("Setarea proprietate folosind o metoda:");
echo $Obiect1->getMessage();
$Obiect3->setMessage("Setarea proprietate clasei 3");
```

rezultatele sunt:

Obiect creat

Mesajul pentru obiectul 1 este:

Setarea proprietate folosind o metoda:

Clasa 3

Observaţie:

În PHP 4 nu este permisă supraîncărcarea metodelor (supraîncărcare – overload- o aceeaşi metodă având în cadrul aceleaşi clase definiţii multiple, fiecare cu un număr diferit de parametrii de intrare). Începând cu PHP 5 s-a introdus o funcţie overload() pentru a putea beneficia de acest avantaj.

Un exemplu concret de utilizare a programării orientate pe obiecte pentru realizarea conexiunii la un server MySQL, respectiv la baza de date, este prezentat în continuare, în contextul în care aceste operaţii implică o refolosire repetată a aceleiaşi secvenţe de cod pentru fiecare script operând cu baza de date.

Ca punct de start, se defineşte o clasă (Clasa1) implementând o metodă (setServer) executând operaţiile dorite (clasă salvată într-un fişier script distinct -Clase.php-, care poate fi referit şi inclus în orice alt script este necesar utilizând comanda include):

- Fişier Clase.php:

```
<?php
class Clasa1
{
function setServer($var1, $var2, $var3, $var4)
{
$returnez = @mysql_connect ("$var1", "$var2", "$var3") or die ("Eroare SERVER");
mysql_select_db("$var4") or die ("Eroare BD");;
return $returnez;
}
}
?>
```

Metoda clasei are patru parametri de intrare (nume server, user, parolă, numele bazei de date), returnând în caz de reuşită un identificator de conectare (util pentru cazul conectării la mai multe servere SQL, pentru identificarea unei anumite conexiuni).

Utilizarea clasei, în orice script este necesară o conexiune la MySQL, presupune o instanţiere a ei (creând un nou obiect), urmată de un apel al metodei ei, setată cu parametri adecvaţi:

```
<?php
include("clase.php");
$Obiect1 =new Clasa1();
$db=$Obiect1->setServer("localhost","root", "parola","baza");
echo 'Conectare OK!';
$query = "SELECT * FROM test";
$interoghez=mysql_query($query);
...
```

O soluţie şi mai compactă ca şi cod, presupune folosirea unui constructor şi astfel, la o instanţiere a clasei se face şi un apel automat al metodei realizând conectarea la server, respectiv la baza de date. Fişierul conţinând clasa cu codul următor:

- Fişier Clase.php:

```
<?php //folosind constructor
class Clasa2
{
var $returnez;
function Clasa2($var1, $var2,$var3,$var4)
{
$this->returnez = mysql_connect("$var1", "$var2", "$var3") or die ("Error connecting to mysql");
mysql_select_db("$var4");
return $this->returnez;
}
}
?>
```

Scriptul în care se face un apel la această clasă pentru realizarea unei conexiuni la o bază de date MySQL poate conţine următorul cod:

```
<?php
include("clase.php");
$Obiect2=new Clasa2("localhost","root","parola", "baza");
$db=$Obiect2->returnez; // referire identificator conectare
//cod interogare
. . .
?>
```

Aplicatii

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
```

```php
$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();
?>
```

Restrictionarea accesului

Dupa cum am precizat mai sus, PHP5 permite marcarea proprietatilor si metodelor unei clase ca private, ceea ce inseamna ca ele nu pot fi modificate sau afisate in afara definitiei clasei respective. Acest lucru este util a impiedica manipularea proprietatilor unei clase de catre o instanta a acesteia. Sa consideram urmatorul exemplu, care ilustreaza acest lucru prin adaugarea unei noi variabile private **$_ultimeleUnitatiConsumate** in clasa **Urs()**:

```php
<?php
// PHP 5
// definitia clasei
class Urs {
    // definitia proprietatilor
    public $nume;
    public $greutate;
    public $varsta;
    private $_ultimeleUnitatiConsumate;
    // constructor
    public function __construct() {
        $this->varsta = 0;
        $this->greutate = 100;
        $this->_ultimeleUnitatiConsumate = 0;
    }
     // definitia metodelor
    public function mananca($unitati) {
        echo $this->nume." mananca ".$unitati." unitati de mancare... ";
        $this->greutate += $unitati;
        $this->_ultimeleUnitatiConsumate = $unitati;
    }
    public function afiseazaUltimaMasa() {
        echo "Unitatile consumate la ultima masa
        au fost ".$this->_ultimeleUnitatiConsumate." ";
    }
}
?>
```

Deoarece variabila **$_ultimeleUnitatiConsumate** este declarata ca private, orice tentativa de a o modifica va genera o eroare. Iata un exemplu:

```php
<?php
$bob = new Urs;
$bob->nume = "Ursul Bobby";
$bob->mananca(100);
$bob->mananca(200);
echo $bob-> afiseazaUltimaMasa();
// urmatoarea linie va genera o eroare
```

```
$bob->_ultimeleUnitatiConsumate = 1000;
?>
```

In mod similar, si metodele unei clase pot fi declarate ca private, ceea ce inseamna ca ele nu pot fi apelate decat din interiorul clasei respective.

Mostenirea unei clase
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 = 0;
        $this->greutate = 100;
    }
    // definitia metodelor
    public function mananca($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 = "alb";
        $this->greutate = 600;
    }
    // definitia metodelor
    public function inoata() {
        echo $this->nume." inoata... ";
    }
}
?>
```

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:

```php
<?php
// creeaza o noua instanta a clasei Urs()
$tom = new Urs;
$tom->nume = "Ursul Tommy";
// creeaza o noua instanta a clasei UrsPolar()
$bob = new UrsPolar;
$bob->nume = "Ursul Bobby";
/* $bob poate apela toate metodele claselor Urs() si UrsPolar() */
$bob->alearga();
$bob->vaneaza();
$bob->inoata();
// $tom poate apela doar metodele clasei Urs()
$tom->alearga();
$tom->vaneaza();
$tom->inoata();
?>
```

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

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

Definitia clasei UrsPolar:

```php
<?php
// PHP 4
// UrsPolar extinde definitia clasei Urs
class UrsPolar extends Urs {
    // constructor
    function UrsPolar() {
        parent::Urs();
        $this->culoare = "alb";
        $this->greutate = 600;
    }
    // definitia metodelor
    function inoata() {
        echo $this->nume." inoata... ";
    }
}
?>
```

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
// 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 */
```

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

Clase abstract si interface

Abstract si **Interface** (*interfata*) sunt tipuri de **clase** mai speciale in **POO**, pentru lucru mai avansat in programarea orientata pe obiecte.

1. Clase si Metode abstract

Clasele abstracte se declara folosind cuvantul **abstract** inaintea lui "class".
La aceste tipuri de clase nu se poate crea instanta de obiect, ele pot fi doar mostenite de alte clase extinse din ele.
In clasele abstracte se definesc si **metode abstracte**, acestea se declara cu acelasi cuvant "*abstract*".
Exemplu de clasa cu o proprietate "protected" o metoda abstracta si una accesor:

```php
<?php
// Definire clasa abstracta
abstract class Fructe {
  protected $color;        // Proprietate

  // Definire metoda abstracta
  abstract function Stoc($luna);

  // Metoda accesor pt. "color"
  public function setColor($c) { $this->color = $c; }
}
?>
```

- Daca se creaza o instanta de obiect la aceasta clasa (de ex.: *$obj = new Fructe();*), va genera eroare de tipul:

```
Fatal error: Cannot instantiate abstract class
```

- Metodele abstracte se creaza doar in clase abstracte.
- Rolul claselor si metodelor abstracte este acela de a crea un model minim de metode obligatorii care trebuie definite in sub-clase normale derivate din ele (cu *extends*). Metodele abstracte definite in cea parinte trebuie create in orice clasa copil extinsa din cea abstracta, cu acelasi numar de parametri (numele poate fi difereit).
De exemplu, orice sub-clasa extinsa din clasa Fructe (definita mai sus) trebuie sa contina metoda Stoc() cu un parametru, cum ar fi cea din urmatorul exemplu, denumita Mere.

```php
<?php
// Definire clasa copil, extinsa din cea abstracta
class Mere extends Fructe {
  private $kg;        // Proprietate

  // Metoda obligatorie (seteaza valoarea proprietatii "Kg")
```

```
  public function Stoc($kg) {
    $this->kg = $kg;
  }

  // Alta Metoda - optionala (returneaza valoarea proprietatii "kg")
  public function getKg() {
    return $this->kg. ' kg';
  }
}
?>
```

- Deoarece clasa Mere extinde Fructe, trebuie sa contina, pe langa alte elemente, si metodele abstracte declarate in aceea (anume Stoc(), cu un parametru). Daca aceasta sub-clasa nu ar avea metoda Stoc(), va genera eroare de genul:

Fatal error: Class Fructe contains 1 abstract method and must therefore be declared abstract or implement the remaining methods (Mere::Stoc) in...

- Sub-clasele care extind o clasa abstracta pot fi utilizate normal, se pot crea si folosi instante de obiect la ele, mostenesc si pot folosi elementele cu atribut "public" si "protected" din cea parinte.
De exemplu:

```
<?php
// Creare instanta de obiect la clasa Mere si apelare metode
$obj = new Mere();
$obj->Stoc(78);
echo $obj->getKg();        // Afiseaza:  78 kg
?>
```

2. Interfaces

Ca rol, se poate spune ca **Interface** este asemanatoare cu clasa "abstract".
Clasa **Interface** este folosita ca tipar, sau template pentru clase cu functii similare, care trebuie sa respecte o anumita structura de metode.
Sintetizat, Interface este o clasa cu o lista de metode obligatorii ce trebuie sa fie create in clasele unde este implementata. Toate metodele specificate in "Interface" sunt cu atribut **public** si trebuie sa fie definite in clasele in care e aplicata, avand acelasi numar de parametri cati sunt indicati in "Interface".

Creare interface

Interface se creaza similar cu celelalte tipuri de clase. Diferenta e aceea ca la definirea ei, in loc de cuvantul "***class***" se foloseste cuvantul "**interface**"; in plus, in corpul ei se scrie doar o lista cu metode publice fara, alt cod.
Sintaxa generala este urmatoarea:

- **interface numeInterfata {**
  public function numeMetoda();
  public function altaMetoda()
  .....
  }

- La declararea metodelor in Interface nu se adauga acoladele sau codul lor, si nici alt atribut diferit de "public".

Exemplu cu o Interface, denumita "ITest", in care sunt definite 2 metode: "Links()" si "Tutoriale()".

```
<?php
// Definire Interface ITest
interface ITest {
  // Lista cu metode
  public function Links();
  public function Tutoriale($str, $nota);
}
?>
```

Implementare interface

Dupa ce a fost definit tiparul "interface", se pot crea clase care implementeaza metodele stabilite in acel tipar, respectand si numarul de parametri.
Implementarea se face adaugand cuvantul **implements** si numele Interfatei la definirea claselor, dupa numele lor.

class NumeClasa implements numeInterfata {
// Instructiuni
}

Acestea trebuie sa contina in corpul lor toate metodele definite in "interface", cu atribut "public", si numarul de parametri stabiliti pt. fiecare (numele la parametri poate fi diferit). Pe langa acestea pot contine si alte metode.

Exemplu cu o clasa care implementeaza interfata ITest creata mai sus.

```
<?php
// Creare clasa care aplica Interfata ITest
class WebDevelopment implements ITest {
  // Definire proprietate 'link' (cu atribut "protected")
  protected $link = 'upit.net';

    /* Definire metodele obligatorii (Links si Tutoriale), din interface */

  // Returneaza valoarea proprietatii 'site'
  public function Links() {
    return $this->link;
  }

  // Returneaza valoarea unei variabile din ea ($re), ce preia argumentele transmise
  public function Tutoriale($gen, $nota) {
    $re = $gen. '-'. $nota;
    return $re;
  }

  // Se pot crea si alte metode, suplimentare
  // Aceasta modifica valoare proprietatii "link"
  public function setLink($link) {
    $this->link = $link;
  }
}
?>
```

- Metodele obligatorii (aici "Links()" si "Tutoriale()") respecta exact numarul de parametri stabiliti in "interface" "ITest". Alte metode (aici "setLink()") si proprietati sunt optionale, in functie de rolul fiecarei clase.
- Numele parametrilor nu conteaza (*observati ca in loc de $str s-a folosit $gen*), dar numarul lor trebuie sa fie aceleasi ca in "interface".

- Daca vreuna din conditii nu ar fi respectata in clasa, cum ar fi: nedefinirea unei metode, adaugarea de parametru in plus sau minus; scriptul genereaza eroare.
Astfel, implementarea de "interface" este utila mai ales cand sunt create mai multe clase cu roluri similare si dorim ca acestea sa aibe toate o anumita ordonare si structura minima de metode, mai usor de retinut.

3. Interface ca tip de date

Interfata poate fi utilizata si ca tip de data la parametri de functii, astfel, acel parametru poate fi utilizat ca instanta de obiect la orice clasa din cele ce folosesc acea "interface".
Exemplu in care este creata si folosita inca o clasa (LimbiStraine) ce aplica Tiparul din "ITest"; contine o proprietate si metodele obligatorii stabilite.

```
<?php
// Creare clasa care aplica Interfata ITest
class LimbiStraine implements ITest {
  // Definire proprietate
  protected $adr = 'upit.net/';

          /* Definire metodele obligatorii (Links si Tutoriale), din interface */

  // Returneaza expresia 'Cale buna'
  public function Links() {
      return'Cale buna';
  }

  // Returneaza valoarea unei variabile din ea (re), ce preia argumentele "nr", "gen" si
proprietatea "adr"
  public function Tutoriale($gen, $nr) {
    $re = $nr.'-'. $this->adr. $gen;
    return $re;
  }
}
?>
```

Intr-un script PHP se poate scrie urmatorul cod:

```
<?php
// Se includ fisierele cu clasele create mai sus (daca sunt in fisiere externe): ITest
(Interfata), WebDevelopment si LimbiStraine
include('interf.ITest.php');       // Interface
include('class.WebDevelopment.php');
include('class.LimbiStraine.php');

// Creare functie care accepta doar argument cu obiect la clasele care au implementata Interfata
"ITest"
function cursuri(ITest $obj) {
  // Apeleaza metodele comune (stabilite in ITest) prin parametru $obj
  echo '<br />'. $obj->Links();
  echo '<br />'. $obj->Tutoriale('php-mysql', 4);
}

// Creare instante de obiect la clasele folosite
$web_development = new WebDevelopment();
$limbi_straine = new LimbiStraine();

// Apeleaza functia cu instantele de obiect ale claselor ce au aplicat ITest
cursuri($web_development);        // "upit.net" si "php-mysql-4"
cursuri($limbi straine);          // "Cale buna" si "4-upit.net/php-mysql"
?>
```

- Functia "cursuri()" creata cu aceasta formula intre acolade "*function cursuri(ITest $obj)*" face ca ea sa accepte ca argument doar obiect care are implementat "ITest".
- Apeland functia cu argumente diferite, reprezentand numele instantelor la clase, foloseste parametru

$obj ca instanta la clasa respectiva, si poate apela aceleasi metodele ("Links()" si "Tutoriale()") pt. fiecare deoarece aceste fiind specificate in "interface" ele trebuie sa existe in fiecare clasa ce apartine acelei Interfate, cu acelasi numar de parametri.
- Prin aceasta tehnica nu mai e nevoie de a crea aceeasi functie pt. fiecare instanta.
Acest exemplu va afisa in browser urmatorul rezultat:

```
upit.net
php-mysql-4
Cale buna
4-upit.net/php-mysql
```

PHP OOP - metode Accesor si Destructor

1. Metoda Accesor

Variabilele (proprietatile) create intr-o clasa pot avea de la inceput o valoare sau pot fi doar simplu declarate, urmand ca valoarea lor sa fie atribuita prin intermediul unei functii (metode). Aceasta functie e denumita generic **Metoda Accessor**, e la fel ca oricare alta metoda, doar ca scopul ei este de a atribui valori proprietatilor; in rest, se construieste si se apeleaza la fel ca celelalte.
Ca sa vedeti cum functioneaza "*metoda accessor*", incercati urmatorul exemplu, in care este creata o clasa "SiteClas" cu doua proprietati ('site' si 'categorie'), ambele fara valoare, iar metoda Constructor e folosita si ca **accesor**, atribuie valori proprietatilor prin parametri ei.

```
<?php
// Se defineste clasa SiteClas
class SiteClas {
  public $site;          // Proprietate declarata fara valoare
  private $categorie;          // Proprietate privata, fara valoare

  // Constructor
  public function   construct($site, $categorie) {
    // Atribuie proprietatilor valoarea din parametri
    $this->site = $site;
  $this->categorie = $categorie;
  }

  // Metoda pt. afisare adresa pagini
  public function pagini($pag) {
    // Afiseaza un sir cu adresa URL, www., valoarea celor 2 proprietati si argumentul primit
  echo '<br />www.'. $this->site.'/'. $this->categorie.'/'. $pag;
  }
}
?>
```

- Cand este creata instanta de obiect la clasa, metoda constructor (care se executa automat) atribuie proprietatilor "*site*" si "*categorie*" valorile din parametri, care trebuie adaugate la crearea instantei.
- Metoda *pagini()*, cand este apelata, afiseaza o adresa URL formata din valoarea proprietatilor "*site*", "*categorie*" (atribuite prin constructor) si argumentul ei *$pag*.
- Salvam clasa intr-un fisier denumit "*class.SiteClass.php*", iar pentru test, se adauga urmatorul cod intr-un fisier .php salvat in acelasi director unde e si clasa.

```php
<?php
include('class.SiteClas.php');        // Include clasa

// Creare instanta de obiect la clasa SiteClas, cu argumentele necesare
$objSite = new SiteClas('upit.net', 'php-mysql');

$objSite->pagini('oop-clase-obiecte.html');        // Apelare metoda pagini()
?>
```

- Dupa executie, in browser va afisa:

upit.net/php-mysql/oop-clase-obiecte.html

Valoarea proprietatii "*site*", avand atribut "public", poate fi modificata si pe parcurs in script, cu expresia:

$objSite->site = valoare;

2. Accesare si modificare proprietati prin metode Accesor

Variabilele in PHP nu au un tip de date stabilit precis la declararea lor, de exemplu, o variabila poate sa contina initial ca valoare un numar, iar pe parcursul scriptului sa i-se atribuie ca valoare un sir sau un Array. Aceasta flexibilitate este folositoare, dar in unele situatii poate prezenta probleme in anumite contexte din codul unei metode.
De exemplu, daca intr-o metoda se parcurg datele dintr-o proprietate de tip Array iar in script acea proprietate primeste o valoare de tip Sir, apar erori.
Pentru a fi siguri ca o proprietate primeste doar tipul de date care poate fi corect utilizat, se declara ca **private** si se folosesc metode accesor pentru accesarea ei, cu functii PHP de verificare a tipului de date. Aceste functii sunt:

- **is_bool()** - ***Boolean*** - una din valorile speciale: *true* sau *false*
- **is_integer()** sau **is_int()** - ***Integer*** - numere intregi (fara virgula)
- **is_float()** sau **is_double()** - ***Float /Double*** - numere cu zecimale (cu virgula)
- **is_numeric()** - ***Number*** - orice numar sau sir ce reprezinta un numar
- **is_string()** - ***String*** - siruri de caractere si cuvinte
- **is_array()** - ***Array*** - Array
- **is_object()** - ***Object*** - Obiect
- **is_resource()** - ***Resource*** - un identificator pentru lucru cu date din sure externe (fisier, baza de date)
- **is_null()** - ***Null*** - Valoare NULL sau nedefinita

Iata o alta versiune a clasei TestClas. Ambele proprietati sunt declarate "private", ca sa nu fie modificate in mod direct in afara clasei. Valorile initiale le primesc la crearea instantei (prin constructor). Pentru a putea accesa si modifica proprietatea "*categorie*" in script, se creaza special cate o metoda accesor: *getCategorie()* si *setCategorie()* (vedeti si explicatiile din cod).

```php
<?php
// Se defineste clasa SiteClas
class SiteClas {
  // Definire proprietati private, fara valoare
  private $site;
  private $categorie;

  // Constructor
  public function __construct($site, $categorie) {
    // Atribuie proprietatilor valoarea din parametri
    $this->site = $site;
    $this->categorie = $categorie;
  }
```

```
  // Metoda accesor - returneaza valoarea proprietatii 'categorie'
  public function getCategorie() {
    return $this->categorie;
  }

  // Metoda accesor pt. setare valoare la "categorie"
  public function setCategorie($val) {
    // Daca $val e de tip Sir (string) si are cel putin un caracter
    // Atribuie valoarea lui proprietatii 'categorie'
    // Altfel, returneaza eroare
    if(is_string($val) && strlen($val)>0) {
      $this->categorie = $val;
    }
  else throw new Exception('Valoare incorecta pt. categorie');
  }

  // Metoda pt. afisare adresa pagini
  public function pagini($pag) {
    // Afiseaza un sir cu adresa URL, www., valoarea celor 2 proprietati si argumentul primit
    echo '<br />www.'. $this->site.'/'. $this->categorie.'/'. $pag;
  }
}
?>
```

- Prin declararea proprietatilor ca "*private*", se respinge accesul direct la ele in afara clasei, iar pt. proprietatea "*categorie*" s-a creat posibilitatea de a fi accesata si modificata valoarea ei prin metodele accesor: *getCategorie()* si *setcategorie($val)*.
- In *setCategorie($val)* se atribuie valoarea transmisa pt. $val la "categorie" (prin formula *$this->categorie = $val;*) doar daca $val e de tip Sir (String) si are cel putin un caracter; altfel, utilizand formula speciala "*throw new Exception('Eroare')*" returneaza un mesaj de eroare.
- Pentru a testa efectul acestor metode, se executa scriptul urmator, in care e utilizata o instanta la aceasta clasa.

```
<?php
include('class.SiteClas.php');        // Include clasa

// Creare instanta de obiect la clasa SiteClas
$objSite = new SiteClas('upit.net', 'php-mysql');

// Afiseaza valoarea returnata de getCategorie() (reprezentand valoarea proprietatii "categorie")
echo $objSite->getCategorie();

// Modifica prin setCategorie() valoarea proprietatii "categorie"
$objSite->setCategorie('html');

$objSite->pagini('introducere.html');        // Apelare metoda pagini()
?>
```

- In browser scriptul va afisa:

```
php-mysql
upit.net/html/introducere.html
```

- "*php-mysql*" este valoarea initiala data proprietatii "categorie" prin crearea instantei la clasa. Dar prin modificarea ei cu *$objSite->setCategorie('html');*, metoda pagini() va utiliza proprietatea "categorie" cu aceasta valoare (*html*).

3. Metoda Destructor

Metoda Destructor se creaza cu numele **__destruct** (*doua caractere '_' la inceput*). Aceasta metoda este opusul lui *__construct*. Daca metoda constructor e apelata automat cand se creaza o instanta de obiect la clasa, metoda Destructor se apeleaza automat cand e stearsa, cu **unset()**, instanta la acea clasa.
Se intelege mai bine din urmatorul exemplu. Clasa Test de mai jos contine o proprietate privata, "nume", o metoda accesor "setNume()", metoda constructor si destructor.

```
<?php
// Clasa Test
class Test {
  private $nume;

  // Constructor
  public function   construct($nume) {
    // Atribuie proprietatii 'nume' valoarea din parametru
    // si afiseaza un mesaj
    $this->nume = $nume;
    echo 'Bine ai venit '. $this->nume. '<br />';
  }

  // Metoda accesor - schimba valoarea proprietatii 'nume'
  public function setNume($nume) {
    $this->nume = $nume;
  }

  // Metoda Destructor
  function __destruct() {
    echo 'Cu bine '. $this->nume;
  }
}
?>
```

- Ca sa testam efectul metodei **__destruct**, se foloseste urmatorul script (*vedeti explicatiile din cod*).

```
<?php
// Creare instanta de obiect la clasa Test
$obj = new Test('UPIT');

// Apeleaza metoda setNume(), care seteaza alta valoarea la prop. 'nume'
$obj->setNume('DPI');

// Sterge instanta de obiect $obj
unset($obj);
?>
```

- Prin argumentul 'Mar' adaugat la crearea instantei, metoda constructor atribuie aceasta valoare proprietatii "nume", pe care o afiseaza in mesajul returnat.
- Apelarea metodei setNume() modifica valoarea acestei proprietati, iar cand instanta de obiect e stearsa, cu *unset($obj)*, se autoapeleaza metoda "*__destruct*" si determina executia codului din ea, care va afisa alt mesaj, cu proprietatea "nume" avand ultima valoare setata.
- Rezultatul afisat in browser este:

```
Bine ai venit UPIT
Cu bine DPI
```

Metoda destructor este utila cand se doreste executarea automata a unor ultime instructiuni cand instanta la clasa e stearsa din memorie prin apelarea functiei PHP unset().

Conectare si utilizare FTP cu PHP

FTP (File Transfer Protocol) este un protocol standard care realizeaza o conexiune client-server pentru transfer si manipulare de fisiere prin Internet.
Cu ajutorul FTP puteti adauga si copia fisiere (si directoare) de pe server sau sa le schimbati permisiunile CHMOD. PHP are functii speciale pentru conectarea la server si lucrul cu FTP.

1. Conectarea la server

Pentru conectarea la server prin FTP sunt necesare 3 date importante: **adresa serverului (*a domeniului*)**, **numele de utilizator** si **parola de conectare prin FTP**.
La inceput se deschide o conexiune cu serverul, folosind functia **ftp_connect()**, aceasta preia ca argument necesar numele serverului, care de obicei are forma "***ftp.server***" ("server" poate fi numele domeniului sau adresa IP a acestuia). Daca deschiderea conexiunii s-a efectuat, functia va returna un ID a conexiunii, in caz contrar returneaza FALSE.
Dupa deschiderea conexiunii se face autentificarea pe server, pentru aceasta se foloseste functia **ftp_login()** care are urmatoarea sintaxa:

ftp_login(conn_id, nume_user, parola)

- unde "**conn_id**" este id-ul conexiunii returnat de "ftp_connect()"
- "**nume_user**" este numele de utilizator pentru conectare la FTP, iar "**parola**" este parola utilizatorului.

Pentru inchiderea conectarii se foloseste functia **ftp_close()**, aceasta preia ca argument id-ul conexiunii.

Iata un exemplu de conectare prin FTP la server-ul unui site:

```
<?php
// Datele pt. conectare la server
$ftp_server = 'ftp.un_site.net';              // Poate fi de exemplu
'ftp.upit.net'
$ftp_user = 'utilizator';
$ftp_pass = 'parola';

// Seteaza Id-ul conexiunii
$conn_id = ftp_connect($ftp_server);

// Executa autentificarea pe server (mod pasiv ca sa nu fie blocat de
firewall)
$login_result = ftp_login($conn_id, $ftp_user, $ftp_pass);
ftp_pasv($conn_id, true);

// Verifica daca s-a reusit sau nu conectarea la server
if ((!$conn_id) || (!$login_result)) {
echo "Conectarea prin FTP la serverul $ftp_server pentru utilizatorul
$ftp_user a iesuat.";
exit;
}
else {
echo "Conectare reusita la serverul $ftp_server, pentru utilizatorul
$ftp_user";
}

// Inchide conexiunea
```

```
ftp_close($conn_id);
?>
```

- Daca s-a efectuat cu succes conectarea si autentificarea pe server, va afisa mesajul de confirmare, in caz contrar apare mesajul de eroare.
- La urma, "ftp_close($conn_id);" inchide conexiunea si elibereaza memoria alocata.

Conectarea la server folosind codul de mai sus este primul pas, dupa aceasta puteti folosi functiile PHP specifice FTP pentru a efectua diferite operatii, cum sunt: preluarea continutului directoarelor, incarcarea sau descarcarea de fisiere, crearea de noi directoare sau modificarea permisiunilor CHMOD.

2. Preluarea continutului directoarelor

Pentru a prelua continutul unui director (numele fisierelor si directoarelor din el) se foloseste functia

ftp_nlist(conn_id, director)

- unde "**conn_id**" este id-ul conexiunii returnat de "ftp_connect()"
- "**director**" este numele directorului a carui continut va fi preluat

Aceasta functie returneaza o matrice secventiala a carei elemente sunt numele fiecarui director si fisier din catalogul cautat.
Mai jos este un exemplu practic.

```
<?php
// Datele pt. conectare la server
$ftp_server = 'ftp.domeniu_site';             // Poate fi de exemplu
'ftp.upit.net'
$ftp_user = 'utilizator';
$ftp_pass = 'parola';

// Seteaza Id-ul conexiunii
$conn_id = ftp_connect($ftp_server);

// Executa conectare pe server (mod pasiv ca sa nu fie blocat de firewall)
$login_result = ftp_login($conn_id, $ftp_user, $ftp_pass);
ftp_pasv($conn_id, true);

// Verifica daca s-a reusit sau nu conectarea la server
if ((!$conn_id) || (!$login_result)) {
echo "Conectarea prin FTP la serverul $ftp_server pentru utilizatorul
$ftp_user a iesuat.";
exit;
}
else {
// Obtine continutul directorului (numele de fisiere si directoare din el)
$continut = ftp_nlist($conn_id, ".");

// Parcurge matricea si afiseaza numele directoarelor si fisierelor returnate
for ($i=0; $i<count($continut); $i++) {
echo '<br>'.$continut[$i];
}
}

// Inchide conexiunea
ftp_close($conn_id);
?>
```

- Dupa setarea conexiunii si autentificarea pe server, daca conectarea a reusit este apelata functia "ftp_nlist()", care va stoca in variabila "$continut" matricea cu numele cataloagelor si fisierelor din directorul radacina (precizat de argumentul ".").
Daca doriti obtinerea datelor din alt director, de exemplu "www", puteti apela functia asa: ***ftp_nlist($conn_id, '/www')***.
- Pentru afisarea (sau preluarea) numelui fiecarui fisier si director returnat se parcurge matricea "$continut".
Rezultatul afisat de acest script va fi ceva similar cu cel de jos:

```
cgi-bin
private
public_html
statistics
www
```

3. Incarcare fisier pe server

Pentru incarcarea (upload) unui fisier pe server prin FTP cu PHP se foloseste functia **ftp_put()**, aceasta returneaza TRUE daca incarcarea s-a efectuat cu succes, in caz contrar returneaza FALSE. Are urmatoarea sintaxa:

ftp_put(conn_id, destinatie, fisier, tip_transfer)

- unde "**conn_id**" este id-ul conexiunii returnat de "ftp_connect()"
- "**destinatie**" este directorul (si numele fisierului) pe server unde va fi incarcat fisierul
- "**fisier**" este numele fisierului care va fi copiat
- "**tip_transfer**" reprezinta modul incarcarii pe server, acesta poate lua valoarea **FTP_ASCII** sau **FTP_BINARY**

Iata un exemplu in care se copie un fisier de pe calculatorul client pe server, in directorul 'www':

```
<?php
// Datele pt. conectare la server
$ftp_server = 'ftp.domeniu_site';
$ftp_user = 'utilizator';
$ftp_pass = 'parola';

// Seteaza Id-ul conexiunii
$conn_id = ftp_connect($ftp_server);

// Executa conectare pe server (mod pasiv ca sa nu fie blocat de firewall)
$login_result = ftp_login($conn_id, $ftp_user, $ftp_pass);
ftp_pasv($conn_id, true);

// Verifica daca s-a reusit sau nu conectarea la server
if ((!$conn_id) || (!$login_result)) {
echo "Conectarea prin FTP la serverul $ftp_server pentru utilizatorul
$ftp_user a iesuat.";
exit;
}
else {
// Datele cu numele fisierului, sursa unde se afla si directorul unde va fi
incarcat pe server
```

```
$fisier = 'test.txt';
$sursa = 'c:/wamp/www/site/'. $fisier;
$destinatie = '/www/'. $fisier;

// Apeleaza functia "ftp_put()" pt. efectuarea transferului
// Afiseaza mesaj de reusita sau nu
if (ftp_put($conn_id, $destinatie, $sursa, FTP_BINARY)) echo "Fisierul
$fisier a fost incarcat pe server";
else echo "Eroare, fisierul $fisier nu a putut fi incarcat pe server";
}

// Inchide conexiunea
ftp_close($conn_id);
?>
```

- Daca datele pentru conectare sunt corecte si PHP are permisiuni de transfer prin FTP, fisierul "test.txt" (aflat pe calculatorul client in directorul "c:/wamp/www/site/") va fi copiat pe server, in directorul "www/".
- Fisierul va fi incarcat cu acelasi nume, daca doriti sa fie transferat pe server cu alt nume, modificati la "$destinatie".

4. Copiere fisier de pe server

Pentru copierea (download) unui fisier de pe server pe calculatorul client se foloseste functia **ftp_get()**, aceasta returneaza TRUE daca transferul s-a efectuat cu succes, in caz contrar returneaza FALSE. Are urmatoarea sintaxa:

ftp_get(conn_id, client, fisier, tip_transfer)

- unde "**conn_id**" este id-ul conexiunii returnat de "ftp_connect()"
- "**client**" este locatia de pe calculatorul client unde va fi copiat fisierul
- "**fisier**" este numele fisierului (si locatia) de pe server unde acesta se gaseste
- "**tip_transfer**" reprezinta modul incarcarii pe server, acesta poate lua valoarea **FTP_ASCII** sau **FTP_BINARY**

Iata un exemplu in care se descarca prin FTP un fisier de pe server pe calculatorul client:

```
<?php
// Datele pt. conectare la server
$ftp_server = 'ftp.domeniu_site';
$ftp_user = 'utilizator';
$ftp_pass = 'parola';

// Seteaza Id-ul conexiunii
$conn_id = ftp_connect($ftp_server);

// Executa conectare pe server (mod pasiv ca sa nu fie blocat de firewall)
$login_result = ftp_login($conn_id, $ftp_user, $ftp_pass);
ftp_pasv($conn_id, true);

// Verifica daca s-a reusit sau nu conectarea la server
if ((!$conn_id) || (!$login_result)) {
echo "Conectarea prin FTP la serverul $ftp_server pentru utilizatorul
$ftp_user a iesuat.";
exit;
}
else {
```

```
// Datele cu numele fisierului, sursa unde se afla si directorul unde va fi
copiat
$fisier = 'test.txt';
$server = '/www/'. $fisier;
$client = 'c:/download/'. $fisier;

// Apeleaza functia "ftp_get()" pt. efectuarea transferului
// Afiseaza mesaj de reusita sau nu
if (ftp_get($conn_id, $client, $server, FTP_BINARY)) echo "Fisierul $fisier a
fost descarcat";
else echo "Eroare, fisierul $fisier nu a putut fi descarcat";
}

// Inchide conexiunea
ftp_close($conn_id);
?>
```

- Daca datele pentru conectare sunt corecte si PHP are permisiuni de transfer prin FTP, fisierul "test.txt" (aflat pe server in directorul "www/") va fi copiat pe calculatorul client, in directorul "c:/download/".
- Fisierul va fi copiatat cu acelasi nume, daca doriti sa fie descarcat pe calculator cu alt nume, modificati la "$client". Daca in directorul unde va fi copiat se mai afla un fisier cu acelasi nume, acesta va fi inlocuit cu cel descarcat de pe server.

5. Creare director pe server

Inainte de a crea un director nou prin FTP, setati directorul curent de lucru care reprezinta locatia unde doriti sa fie creat noul catalog. Pentru aceasta se foloseste functia **ftp_chdir(conn_id, director)**; "conn_id" este id-ul conexiunii iar "director" este numele directorului care va fi setat pentru lucru.

Pentru crearea unui director pe server prin FTP se foloseste functia **ftp_mkdir()**, in caz de succes returneaza numele noului director creat, iar in caz contrar returneaza FALSE. Are urmatoarea sintaxa:

ftp_mkdir(conn_id, nume_dir)

- unde "**conn_id**" este id-ul conexiunii returnat de "ftp_connect()"
- "**nume_dir**" este numele noului director

Iata un exemplu in care se creaza un director "test" pe server, in "www/":

```
<?php
// Datele pt. conectare la server
$ftp_server = 'ftp.domeniu_site';
$ftp_user = 'utilizator';
$ftp_pass = 'parola';

// Seteaza Id-ul conexiunii
$conn_id = ftp_connect($ftp_server);

// Executa conectare (mod pasiv ca sa nu fie blocat de firewall)
$login_result = ftp_login($conn_id, $ftp_user, $ftp_pass);
ftp_pasv($conn_id, true);

// Verifica daca s-a reusit sau nu conectarea la server
if ((!$conn_id) || (!$login_result)) {
```

```
echo "Conectarea prin FTP la serverul $ftp_server pentru utilizatorul
$ftp_user a iesuat.";
exit;
}
else {
$dir = 'un_director';          // Numele noului director

// Daca e setat directorul de lucru "www/"
if(ftp_chdir($conn_id, '/www/')) {
// Apeleaza functia "ftp_mkdir()" pt. crearea directorului
// Afiseaza mesaj de reusita sau nu
if (ftp_mkdir($conn_id, $dir)) echo "Directorul $dir a fost creat";
else echo "Eroare, directorul $dir nu a putut fi creat";
}
}

// Inchide conexiunea
ftp_close($conn_id);
?>
```

- Daca datele pentru conectare sunt corecte si directorul curent de lucru (www/) este setat cu succes iar PHP are permisiuni de scriere pe server, in catalogul "www/" va fi creat directorul "un_director".

*Pentru stergerea unui director se foloseste functia **ftp_rmdir(conn_id, del_dir)**, unde "conn_id" este id-ul conexiunii iar "del_dir" este numele directorului care va fi sters.*

6. Transferarea unei portiuni de fisier pe server

Pe langa transferul unui fisier complet, PHP ofera si functii pentru transferul unei anumite parti din continutul unui fisier.
O functie care face acest lucru este **ftp_fput()**, aceasta incarca pe server continut dintr-un fisier deschis cu "fopen()", returneaza TRUE in caz de succes, iar in caz contrar returneaza FALSE.
Are urmatoarea sintaxa:

ftp_fput(conn_id, destinatie, pointer_fisier, tip_transfer)

- unde "**conn_id**" este id-ul conexiunii returnat de "ftp_connect()"
- "**destinatie**" este directorul de pe server unde va fi incarcat fisierul
- "**pointer_fisier**" reprezinta identificatorul (pointer-ul) de fisier deschis cu "fopen()". Retine continutul de la acest pointer pana la sfarsitul fisierului.
- "**tip_transfer**" reprezinta modul incarcarii pe server, acesta poate lua valoarea **FTP_ASCII** sau **FTP_BINARY**

Iata un exemplu in care este preluata o parte din continutul unui fisier si transferata intr-un fisier pe server:

```
<?php
$fisier = "test.txt";          // Fisierul care va fi deschis pt. citire

// Daca fisierul a fost deschis pt. citire
if ($fp = fopen($fisier, rb)) {
fseek($fp, 40);        // Seteaza pozitia pointer-ului in fisier

// Datele pt. conectare la server
$ftp_server = 'ftp.domeniu_site';
$ftp_user = 'utilizator';
```

```
$ftp_pass = 'parola';

// Seteaza Id-ul conexiunii
$conn_id = ftp_connect($ftp_server);

// Executa conectare (mod pasiv ca sa nu fie blocat de firewall)
$login_result = ftp_login($conn_id, $ftp_user, $ftp_pass);
ftp_pasv($conn_id, true);

// Verifica daca s-a reusit sau nu conectarea la server
if ((!$conn_id) || (!$login_result)) {
echo "Conectarea prin FTP la serverul $ftp_server pentru utilizatorul
$ftp_user a iesuat.";
exit;
}
else {
$destinatie = '/www/test2.txt';               // Directorul destinatie si
numele fisierului

// Transfera continutul intr-un fisier nou pe server
// Afiseaza mesaj de reusita sau nu
if (ftp_fput($conn_id, $destinatie, $fp, FTP_BINARY)) echo 'Continutul a fost
transferat';
else echo 'Eroare, continutul nu a putut fi transferat';
}

// Inchide conexiunea
ftp_close($conn_id);

// Elibereaza memoria ocupata de deschiderea fisierului
fclose($fp);
}
?>
```

- Daca fisierul "test.txt" a fost deschis pentru citire, functia "fseek($fp, 40)" muta pointer-ul la pozitia 40, astfel va fi retinut continutul din fisier care incepa de la caracterul 41 pana la sfarsit (*primul caracter are pozitia 0*).
- In continuare, daca s-a efectuat cu succes conexiunea si autentificarea prin FTP, se transfera continutul in fisierul "test2.txt" din directorul "www" de pe server (*daca fisierul "test2.txt" exista deja pe server, continutul lui va fi inlocuit cu cel transferat*).

In cazul in care doriti sa transferati intreg continutul unui fisier deschis cu "fopen()", nu este necesara utilizarea funtiei "fseek()".

Aceasta metoda este utila cand doriti sa adaugati un sir sau continut, de exemplu preluat dintr-un form, pe server prin FTP. Adaugati sirul intr-un fisier temporar pe care apoi il deschideti cu "fopen()" si transferati continutul cu "ftp_fput()".

*Pentru stergerea unui fisier se foloseste functia **ftp_delete(conn_id, 'cale/fisier')**, unde "conn_id" este id-ul conexiunii iar "cale/fisier" este calea si numele fisierului care va fi sters.*

7. Alte functii utile pentru FTP

Pe langa functiile prezentate in exemplele de mai sus, PHP are si alte functii utile pentru lucrul cu fisiere si directoare prin FTP.
Iata cateva din acestea:

- **ftp_cdup(conn_id)** - Schimba ca director curent de lucru directorul parinte
- **ftp_chmod(conn_id, octal_CHMOD, fisier)** - Seteaza permisiunile CHMOD a unui fisier prin FTP
- **ftp_mdtm(conn_id, fisier)** - returneaza timpul, in format Unix, cand a fost ultima data modificat un fisier
- **ftp_pwd(conn_id)** - Returneaza numele directorului curent de lucru
- **ftp_rename(conn_id, nume_actual, nume_nou)** - Modifica numele unui director sau fisier
- **ftp_size(conn_id, fisier)** - Returneaza marimea fisierului specificat, in bytes (*unele servere nu suporta aceasta functie*)