Pagini WEB cu PHP 5

Mihai Scorțaru, Claudiu Soroiu

Acest articol este ultimul din seria destinată modului de utilizare a limbajului PHP 4 și interacțiunea sa cu alte medii. De data aceasta nu vom mai prezenta funcții și interacțiuni, ci vom prezenta îmbunătățirile care au fost aduse de versiunea 5 a interpretorului PHP.

Noutăți

Cele mai multe îmbunătățiri care au fost aduse limbajului *PHP* sunt legate de paradigma programării orientate obiect.

În cadrul acestei versiuni au fost rescrise complet rutinele de manipulare a obiectelor din cadrul *script*-urilor *PHP* astfel încât codul să se execute mai eficient și să ofere mai multe facilități programatorilor.

În versiunile precedente versiunii 5 a interpretorului, obiectele erau tratate ca tipuri primitive de date. Astfel, în momentul atașării unei valori de tip obiect unei variabile, obiectul era mai întâi copiat în zona de memorie corespunzătoare variabilei respective. În această versiune obiectele sunt identificate prin referințe, similar limbajului *Java*.

De exemplu, dacă avem următorul cod *PHP*:

```
class Complex{
  var $re;
  var $im;

  function Complex($re = 0, $im = 0){
    $this->setRe($re);
    $this->setIm($im);
  }

  function getRe(){
    return $this->re;
  }
  function setRe($re){
    $this->re = $re;
  }
  function getIm(){
    return $this->im;
  }
  function setIm($im){
    $this->im = $im;
  }
}

function changeValue($complex, $re, $im){
  $complex->setRe($re);
  $complex->setIm($im);
}

$complex = new Complex(1, 0.5);
changeValue($complex, 2, 5);
echo $complex->getRe();
```

în urma executării ultimei linii de cod se va obține valoarea 1 în locul valorii 2, deoarece, în momentul apelării funcției `changeValue`, se creează o copie a obiectului `$complex` și toate modificările se vor efectua asupra copiei fără a fi observate din exteriorul funcției.

Această eroare de programare poate fi reparată folosind transmiterea prin referință a obiectului `$complex` din cadrul funcției, folosind de mai multe ori operatorul `&`.

În *PHP* 5 obiectele sunt transmise prin referință tot timpul.

În continuare vom prezenta modificările aduse modelului obiect de versiunea 5 a interpretorului *PHP*.

Modificatorii de acces public, private și protected

În noua versiune a interpretorului *PHP* au fost introduse cuvintele cheie **`public`**, **`private`** și **`protected`**. Cei care au folosit limbaje de programare ca *C++* sau *Java* cunosc semnificația acestor cuvinte cheie în programarea orientată obiect.

Aceste cuvinte cheie sunt modificatori de acces la metodele și proprietățile unor clase.

Dacă în cadrul declarației unei clase, o metodă sau proprietate este precedată de cuvântul cheie **public**, atunci acea metodă va putea fi accesată din exteriorul declarației clasei.

Dacă în cadrul declarației unei clase, o metodă sau proprietate este precedată de cuvântul cheie **protected**, atunci acea metodă va putea fi accesată doar în cadrul declarației clasei curente și a claselor derivate din aceasta.

Dacă în cadrul declarației unei clase, o metodă sau proprietate este precedată de cuvântul cheie **private**, atunci acea metodă nu va putea fi accesată din exteriorul declarației clasei curente.

Dacă pentru un membru al unei clase nu este specificat nici un modificator de acces, acel membru va avea implicit modificatorul **public**.

În continuare vă vom prezenta un exemplu de utilizare corectă a acestor modificatori de acces la membrii claselor.

```
class Punct{
  private $x;
  private $y;

  public function Punct($x,$y){
    setXCoord($x);
    setXCoord($y);
  }

  function setXCoord($x){
    $this->x = $x;
  }
  function getXCoord(){
    return $this->x;
  }
  function setYCoord($y){
    $this->y = $y;
  }
  function getYCoord(){
    return $this->y;
  }

  //funcția următoare este corectă deoarece $punct are tipul
  //Punct
  protected function setLocation($punct){
    $this->x = $punct->x;
    $this->y = $punct->y;
  }
}

$punct = new Punct(1, 0.5);
echo $punct->x;                    //nu afișează nimic
$punctnou = new Punct(3, 2);
$punct->setLocation($punctnou); //nu face nimic
$punct->setXCoord(3);
$punct->setXCoord(5);
echo $punct->x." ".$punct->y;
```

Modificatorul final

În limbajul *PHP* a fost introdus cuvântul cheie **final**. Acest cuvânt cheie poate fi folosit în cadrul declarării de membri ai unei clase sau în cadrul declarării clasei.

Dacă un membru este declarat ca fiind **final**, atunci acel membru al clasei nu va putea fi suprascris (nu i se va putea modifica funcționalitatea) în cadrul claselor derivate.

Dacă o clasă este declarată ca fiind **final**, atunci nu se pot crea clase derivate din aceasta.

Poate vă întrebați care este utilitatea modificatorului **final**. Modificatorul **final** indică programatorului că elementul care are această proprietate reprezintă o operație sau structură *atomică* și o structură atomică nu mai poate fi divizată.

De exemplu, la nivel abstract, un punct este o structură atomică. În momentul în care acel punct trebuie desenat pe o suprafață de desen, atunci nu mai este o operație atomică deoarece poate fi desenat folosind mai multe culori.

În continuare vă prezentăm un exemplu de utilizare a acestui modificator.

```
final class ClasaAtom{
  function ClasaAtom(){
  }
}

//declarația următoare nu este validă
class ExtindeAtom extends ClasaAtom{
}

class Factory{
  function Factory(){
  }

  //metoda următoare nu poate fi suprascrisă de clasele derivate
  //din clasa Factory
  final function createInstance($a){
    ...
  }
}
```

Constructori

În versiunea 5 a interpretorului *PHP* au fost unificați constructorii. De această dată, pentru a construi un obiect, trebuie implementată metoda `__construct()`.

Clasele pe care le-am prezentat anterior în cadrul articolului sunt corecte pentru noul interpretor *PHP*, deoarece s-a păstrat compatibilitatea cu versiunea 4, astfel că dacă în cadrul definiției unei clase interpretorul nu găsește metoda `__construct()` atunci acesta va căuta o metodă a cărui nume este identic cu cel al clasei și o va folosi pe post de constructor.

De exemplu, clasa **Punct**, ca să fie compatibilă numai cu limbajul PHP 5, ar trebui implementată astfel:

```
class Punct{
  private $x;
  private $y;
```

```
  function __construct($x,$y){
    setXCoord($x);
    setXCoord($y);
  }
...
```

Am omis din acest exemplu celelalte metode ale clasei `Punct` deoarece numai constructorul este important în acest moment.

În *PHP* în momentul construirii unui obiect, nu este apelat constructorul clasei părinte, acesta trebuind apelat, dacă este cazul, folosind instrucțiunea:
`parent::__construct()`.

Destructori

În cadrul claselor definite în limbajul *PHP* 5 pot fi declarați destructori de clase.

Destructorii sunt utilizați în momentul distrugerii (eliberării memoriei alocate) unui obiect. Un obiect este distrus (eliminat din memorie) în momentul în care nu mai există referințe către el. Necesitatea destructorilor a apărut în momentul în care s-a pus problema transmiterii obiectelor prin referință și nu prin valoare cum se efectua transmiterea obiectelor în versiunile anterioare ale interpretorului.

Un destructor este dat de o metodă al cărui nume este `__destruct()` și care nu are parametri.

Ca și în cazul constructorilor, în momentul apelului unui destructor pentru o clasă nu este apelat automat destructorul părintelui, acesta trebuid apelat folosind instrucțiunea:
`parent::__destruct()`.

În continuare vă prezentăm un exemplu de utilizare a destructorilor.

```
class Linie{
  private $capat1;
  private $capat2;

  function __construct($p1, $p2){
    $this->capat1 = $this->p1;
    $this->capat2 = $this->p2;
  }

  function __destruct($p1, $p2){
    $this->capat1 = null;
    $this->capat2 = null;
  }
}
```

Operatorul delete

Deși pare ciudat, există momente din timpul execuției unui script *PHP* în care anumite variabile, care nu mai sunt utilizate, să refere niște obiecte. Din această cauză obiectele respective nu sunt distruse. Pentru a evita astfel de situații a fost introdus operatorul **`delete`**. Acest operator apelează metoda **`__destruct()`** a obiectului care va fi distrus și eliberează memoria asociată acestuia.

În continuare vă prezentăm un exemplu de utilizare a acestui operator.

```
$punct1 = new Punct(1, 1);
$punct2 = new Punct(2, 2);
$linie = new Linie($punct1, $punct2);
$linie_noua = $linie;
delete $linie;
```

Modificatorul static

În limbajul *PHP*, ca de altfel și în alte limbaje moderne de programare există modificatorul **`static`**.

Haideți să vedem ce înseamnă faptul că un membru al unei clase a fost declarat **`static`**.

În momentul creării unui obiect, se alocă o zonă de memorie pentru obiectul nou creat. Pentru membrii statici ai unei clase se creează o altă zonă de memorie în care vor fi stocați. Astfel, toate obiectele de un anumit tip partajează aceeași zonă de memorie pentru membrii statici.

Se recomandă folosirea modificatorului **`static`** pentru membrii unei clase care sunt independenți de instanțele acesteia, adică tot timpul vor avea aceeași valoare în cadrul tuturor instanțelor.

Vă recomandăm să nu deduceți, din faptul că un membru are aceeași valoare în cadrul tuturor instanțelor unei clase, că acel membru este o constantă deoarece nu este adevărat, și aceasta din cauză că în timpul execuției unui *script* valoarea membrului respectiv se poate modifica, dar se va modifica în același fel în toate instanțele.

În versiunea 5 a limbajului *PHP* modificatorul static poate fi aplicat metodelor unei clase, iar proprietățile statice ale unei clase pot fi acum inițializate în momentul declarării lor.

Membrii statici ai unei clase nu pot fi folosiți prin apel indirect (folosirea operatorului `->`) ci folosind construcția: `NumeClasă::numeMembru`.

În continuare vă prezentăm un exemplu de utilizare a modificatorului **`static`**.

```
class Locale{
  private static defaultLocation = "Romania";
  private location;

  function __construct($location = null){
    if ($location == null)
      setLocation(Locale::$defaultLocation);
    else
      setLocation($location);
  }

  function getDefaultLocation(){
    return Locale::$defaultLocation;
  }
  function setDefaultLocation($location){
    Locale::$defaultLocation = $location;
  }
```

```
  function getLocation(){
    return $this->location;
  }
  function setLocation($location){
    $this->location = $location;
  }
}
```

Clase și metode abstracte

În cadrul programării orientate obiect, pentru a reutiliza clase, se recomandă crearea de ierarhii de clase și clasele să fie specializate pe anumite operații în sensul că operațiile comune sau chiar identice ale unor clase să fie definite în cadrul unui părinte.

Există cazuri în care două sau mai multe clase care sunt similare din punctul de vedere al unor proprietăți și comportament, dar în cadrul unui posibil părinte nu pot fi definite toate funcțiile comune claselor. În acest caz, ar trebui definite în părinte metodele comune claselor chiar dacă implementarea lor depinde de clasele în care sunt construite.

Pentru a ușura munca programatorilor, a fost introdus conceptul de *clasă abstractă* și *metodă abstractă*. Acest concept se regăsește și în *PHP* 5 și se poate profita de el prin utilizarea cuvântului cheie **abstract** înaintea declarației unei clase sau a unei metode.

O metodă este abstractă dacă îi lipsește implementarea.

O clasă este abstractă dacă nu a fost construită complet sau conține cel puțin o metodă abstractă.

Clasele abstracte nu pot fi instațiate deoarece în cadrul implementării anumitor metode din cadrul unei clase abstracte se pot apela metode care au fost declarate ca fiind abstracte.

Din punct de vedere logic o clasă nu poate fi declarată ca fiind **final** și **abstract** în același timp deoarece aceasta nu ar putea fi utilizată deloc în programare.

În continuare vă prezentăm un exemplu de utilizare al acestui cuvânt cheie **abstract**:

```
abstract class CoadaDePrioritati{
  abstract public function push($obj);
  public function pop(){
    $obj = $this->peek();
    $this->remove($obj);
    return $obj;
  }
  abstract public function peek();
  abstract protected function remove($obj);
  abstract public function isEmpty();
}

final class Heap extends CoadaDePrioritati{
  public function push($obj){
    ...                    //aici urmează implementarea
  }
  public function peek(){
    ...                    //aici urmează implementarea
  }
  public function remove($obj){
    ...                    //aici urmează implementarea
  }
  public function isEmpty(){
    ...                    //aici urmează implementarea
  }
}
```

Interfețe

În limbajul *PHP* 5 a fost introdus conceptul de interfață. O *interfață* este o clasă abstractă ai cărei membri sunt publici și care nu are nici o metodă implementată. De fapt, o interfață este un schelet pentru clasele care o vor implementa.

În *PHP* 5 o interfață se creează asemenea unei clase cu excepția faptului că metodele pot fi implementate, iar în locul cuvântului cheie **class** trebuie utilizat cuvântul cheie **interface**.

Spre deosebire de cazul în care o clasă este derivată din altă clasă și pentru a indica acest lucru, se folosește cuvântul cheie **extends**. În cazul în care o clasă implementează metodele prezente într-o interfață se va folosi cuvântul cheie **implements**, urmat de o listă de interfețe separate între ele prin virgulă.

În continuare vom prezenta un exemplu de utilizare a interfețelor în *PHP* 5.

```
interface Lista{
  public function add($obj);
  public function get($i);
  public function set($obj,$i);
  public function remove($i);
  public function size();
}

class Vector implements Lista{
  private $vector;

  public function __contruct(){
    $this->vector = array();
  }

  public function add($obj){
    $this->vector[] = $obj;
  }
  public function get($i){
    return $this->vector[$i];
  }
  public function set($obj,$i){
    if ($i <= $this->size)
      $this->vector[$i] = $obj;
  }
  public function remove($i){
    for($j = $i; $j < $this->size()-1; $j++)
      $this->vector[$j] = $this->vector[$j+1];
    array_pop($this->vector);
  }
```

```
  public function size(){
    return sizeof($this->vector);
  }
}
```

Operatorul instanceof

Acest operator este prezent și în limbajul *Java* și funcționalitatea sa este echivalentă cu cea a funcției `is_a()`. Acest operator este binar și are ca operatori un obiect (în partea stângă) și un nume de clasă sau interfață (în partea dreaptă) și efectul folosirii lui este valoarea logică **`TRUE`** dacă obiectul este o instanță a clasei sau este derivat din aceasta.

În continuare vă prezentăm un exemplu de utilizare a acestui operator.

```
$a = new Punct(1, 1);
if ($a instanceof Punct)
  echo "a este instanță a clasei Punct";
```

Clonarea explicită

Datorită faptului că în *PHP* 5 obiectele sunt transmise prin referință și nu prin valoare așa cum se transmiteau în versiunile anterioare, a apărut problema replicării obiectelor.

Pentru aceasta s-a introdus metoda `__clone()`. Pe lângă faptul că această metodă se implementează în cazul în care se dorește replicarea unui obiect, această replicare este controlată de programator și se poate face optim în funcție de fiecare clasă în parte.

În continuare vă vom prezenta un exemplu de replicare a unui obiect.

```
class Punct{
  public $x,$y;

  public function __construct(){
    $this->x = 0;
    $this->y = 0;
  }

  public function __clone(){
    $res = new Punct();
    $res->x = $this->x;
    $res->y = $this->y;
    return res;
  }
}

$p1 = new Punct();
$p1->x = 4;
$p1->y = 5;
$copie_p1 = $p1->__clone();
```

Metoda __toString()

În *PHP* 5 există posibilitatea ca orice obiect să aibă o reprezentare textuală. Dacă un obiect se folosește în operații cu șiruri de caractere, atunci se va folosi o reprezentare textuală a acestuia. În *PHP* 5, prin implementarea metodei `__toString()` programatorul poate controla modul în care un obiect va fi reprezentat sub forma unui șir de caractere.

În continuare vă prezentăm modul de utilizare a metodei `__toString()`.

```
class Punct{
  public $x,$y;
  public function __construct(){
    $this->x = 0;
    $this->y = 0;
  }

  public function __toString(){
    return "(".$this->x.", ".$this->y.")";
  }
}

$p1 = new Punct();
$p1->x = 4;
$p1->y = 5;
echo $p1;                    //se va tipări (4, 5)
```

Constante în interiorul claselor

În această versiune a limbajului *PHP* 5 se pot defini constante care să fie membri ai unor clase.

Constantele se declară similar proprietăților, dar sunt precedate de cuvântul cheie **`const`** și trebuie să fie inițializate la declarare.

În continuare vă prezentăm un exemplu de utilizare a constantelor.

```
class Clasa{
  const constant = "constant";
}

echo "Clasa::constant = ".Clasa::constant."\n";
```

Dereferențierea obiectelor

În versiunile precedente ale limbajului *PHP* nu erau posibile apeluri de forma `$obiect->metoda1()->metoda2()` în cazul în care `$obiect->metoda1()` returnează un obiect care are metoda `metoda2()`.

În *PHP* 5 acest lucru a fost făcut posibil prin dereferențierea obiectelor rezultate în urma apelurilor de metode și funcții.

Tipuri de date pentru parametri de tip obiect

În cadrul semnăturilor metodelor prezente în definiția unei clase, în fața parametrilor de tip obiect ale acestora se poate preciza tipul clasei pe care trebuie să îl aibă.

În momentul execuției codului *PHP*, dacă un obiect transmis ca parametru unei metode nu are tipul cerut, atunci este generată o eroare.

Nu pot fi impuse restricții asupra tipurilor variabilelor care au tipuri primitive (întreg, șir de caractere, real, vector etc.).

În continuare vă prezentăm un exemplu de utilizare a acestei facilități oferite de limbajul *PHP* 5.

```
class Linie{
  private $capat1;
  private $capat2;

  function __construct(Punct $p1, Punct $p2){
    $this->capat1 = $this->p1;
    $this->capat2 = $this->p2;
  }
```

Moștenirea multiplă

În *PHP* 5 a fost făcută posibilă moștenirea multiplă. O clasă acum poate fi derivată din una sau mai multe clase, deci poate să aibă mai mulți părinți. Acest lucru a afectat modul de funcționare al funcției `get_parent_class()`.

Pentru a crea o clasă care să fie derivată din mai multe clase este nevoie ca după cuvântul cheie **`extends`** să se găsească lista numelor claselor moștenite separate între ele prin virgulă. Pentru a crea o clasă care are mai mulți părinți sintaxa este următoarea:

```
class Fiu extends Parinte1, Parinte2, ... {
  ...
};
```

Iteratori

În această versiune a limbajului *PHP* pot fi definite clase care enumeră elementele altei clase. Aceste clase se numesc *iteratori*.

Obiectele se pot folosi în cadrul ciclurilor repetitive **`foreach`** la fel ca și tablourile unidimensionale create cu funcția `array()` cu condiția ca acestea să implementeze interfața `Traversable` care nu conține nici o metodă și nici o proprietate. Pentru obiecte cheile sunt reprezentate de numele proprietăților publice, iar valorile cheilor sunt chiar valorile proprietăților respective.

În continuare vă prezentăm un exemplu de enumerare a proprietăților unei clase:

```
class Punct2D implements Traversable{
  public $x = 4;
  public $y = 5;
}

$obj = new Punct2D();
foreach ($obj as $nume => $valoare){
  echo $nume."=".$valoare."\n";
}
```

Există posibilitatea ca o clasă să implementeze interfața `IteratorAggregate` și să definească metoda `getIterator()` care trebuie să returneze un obiect care implementează interfața `Iterator`. Cu ajutorul acestor două interfețe se specifică interpretorului *PHP* modul în care se va face enumerarea elementelor.

Interfața `Iterator` are următoarele metode:

- `rewind()` - trebuie să realizeze derularea la primul element al unui obiect (de exemplu, primul element al unui vector);
- `current()` - trebuie să returneze elementul curent;
- `key()` - trebuie să returneze cheia elementului curent;
- `next()` - trebuie să realizeze trecerea la următorul element dintr-o posibilă listă de elemente;
- `hasMore()` - trebuie sa returneze o valoare logică care indică dacă mai sunt elemente de enumerat sau nu.

În continuare vă prezentăm un exemplu de utilizare a acestei modalități de enumerare a elementelor unei clase.

```
class IteratorElemente implements Iterator{
  private $obj;
  private $nr;

  function __construct($obj){
    $this->obj = $obj;
  }
  function rewind(){
      $this->nr = 0;
  }
  function hasMore(){
    return $this->num < sizeof($this->obj);
  }
  function key(){
     return $this->nr;
  }
  function current(){
    return $this->obj[$this->nr];
  }
  function next(){
    $this->num++;
  }
}

class Obiect implements IteratorAggregate{
  function getIterator(){
    $a = array(1,2,3,4,5,"a","c");
    return new IteratorElemente($a);
  }
}

$obj = new Obiect;
foreach($obj as $cheie => $valoare){
  echo $cheie."=".$valoare."\n";
}
```

Supraîncărcarea accesului la membrii unei clase

O altă facilitate interesantă a limbajului *PHP* este aceea că oferă supraîncărcarea membrilor unei clase în sensul că se poate realiza accesul la diferite metode și proprietăți ale clasei care nu există în momentul creării unui obiect ci pot apărea pe parcursul execuției codului sau modificarea / aflarea valorilor unor proprietăți implică și alte calcule.

Dacă în cadrul unui obiect este implementată metoda `__get()` care primește ca parametru numele unei proprietăți, atunci interpretorul *PHP* apelează la această metodă pentru a afla valoarea proprietății și în acest fel se poate

controla modul în care valoarea unei proprietăți va fi returnată.

Dacă în cadrul unui obiect este implementată metoda `__set()` care primește ca parametri numele unei proprietăți și o valoare, atunci interpretorul *PHP* apelează la această metodă pentru a seta valoarea proprietății și în acest fel se poate controla modul în care valoarea unei proprietăți se modifică.

Dacă în cadrul unui obiect este implementată metoda `__call()` care primește ca parametri numele unei proprietăți și o listă de argumente, atunci interpretorul *PHP* apelează la această metodă în momentul apelului unei metode a obiectului respectiv, și în acest fel se poate controla modul și condițiile de execuție ale metodelor.

În continuare vă prezentăm un exemplu de utilizare a acestor facilități oferite de limbajul *PHP*.

```
class Setter{
  public $n;
  public $x = array("a" => 1, "b" => 2, "c" => 3);

  function __get($nm){
    echo "Returnare [$nm]\n";
    if (isset($this->x[$nm])) {
      $r = $this->x[$nm];
      echo "Am gasit proprietatea!\n";
      return $r;
    } else {
      echo "Nu am gasit proprietatea!\n";
  }
}

  function __set($nm, $val){
    echo "Setez valoarea pentru [$nm]\n";
    if (isset($this->x[$nm])) {
      $this->x[$nm] = $val;
      echo "Setare reusita!\n";
    } else {
      echo "Setare esuata!\n";
    }
  }
}

class Caller{
  var $x = array(1, 2, 3);

  function __call($m, $a){
    echo "A fost apelată metoda $m:\n";
    var_dump($a);
    return $this->x;
  }
}

$obj = new Setter();
$obj->n = 1;
$obj->a = 100;
$obj->a++;
$obj->z++;
var_dump($obj);

$foo = new Caller();
$a = $foo->test(1, "2", 3.4, true);
var_dump($a);
```

Alte elemente noi

Funcția __autoload()

Pentru a ușura munca programatorilor și codul *PHP* să nu conțină prea multe directive **include**, interpretorul *PHP* 5, în momentul în care găsește un nume de clasă care nu a fost încărcată în memorie, apelează funcția `__autoload()` dacă este definită. Această funcție este apelată cu un singur parametru de tip șir de caractere care reprezintă numele clasei care nu a fost găsită în memorie, deoarece se obișnuiește ca fiecare clasă să fie definită într-un script *PHP*.

În continuare vă vom prezenta un exemplu de utilizare a funcției `__autoload()`.

```
function __autoload($className){
  include_once $className.".php";
}

$object = new Punct;  //se va încărca fișierul Punct.php
```

Tratarea excepțiilor

În *PHP* 5 a fost introdus un mecanism de tratare a excepțiilor similar celui din *Java*, singura diferență fiind aceea că nu a fost implementată clauza **finally**.

Pentru a putea intercepționa o excepție care este posibil să fie generată de un bloc de cod, atunci acel bloc trebuie să se afle în interiorul unei clauze **try**. Clauza **try** poate fi urmată de una sau mai multe clauze **catch**. O clauză **catch** poate captura numai un anumit tip de excepție.

Tratarea excepțiilor se face în felul următor:

```
try{
  … //secvență de cod
}
catch (TipExceptie1 exceptie1){
  … //tratare TipExceptie1 reprezentată de exceptie1
}
catch (TipExceptie2 exceptie2){
  … //tratare TipExceptie2 reprezentată de exceptie2
}
…
catch (TipExceptieN exceptieN){
  … //tratare TipExceptieN reprezentată de exceptieN
}
```

unde `exceptie1`, `exceptie2`, …, `exceptieN` sunt obiecte care au tipurile `TipExceptie1`, `TipExceptie2`, …, respectiv `exceptieN`.

O excepție este reprezentată printr-o clasă definită de utilizator sau predefinită. În interiorul unui bloc de cod, o excepție este generată prin intermediul clauzei **throw** care este urmată de un obiect.

O excepție este *aruncată* la primul nivel care se află în interiorul unei clauze **try** și se verifică în ordine clauzele **catch** până în momentul în care tipul obiectului care constituie excepția este cel prezent în clauză (sau derivat din acesta). Dacă nici un tip de obiect prezent într-o clauză **catch** a unui bloc **try** nu se potrivește cu cel al excepției generate, atunci excepția generată este aruncată la nivelul următor.

Dacă s-a ajuns la cel mai de sus nivel, atunci excepția este tratată de interpretorul *PHP*.

În continuare vă prezentăm un exemplu de tratare a excepțiilor în limbajul *PHP* 5.

```
class ExceptieProprie{
  function __construct($exception){
    $this->exception = $exception;
  }

  function Display(){
   print "ExceptieProprie: $this->exception\n";
  }
}
class ExceptieProprieExt extends
                                ExceptieProprie{
  function __construct($exception){
    $this->exception = $exception;
  }

  function Display(){
   print "ExceptieProprie: $this->exception\n";
  }
}

try{
  throw new ExceptieProprieExt('Eroare');
}
catch (ExceptieProprie $exception){
  $exception->Display();
}
catch (Exception $exception){
  echo $exception;
}
```

Se recomandă ca excepțiile definite de utilizator să extindă clasa `Exception` deoarece aceasta oferă mai multe informații legate de generarea acestora.

Ciclul repetiv foreach cu referințe

În această versiune a interpretorului *PHP* 5 este posibil ca în cadrul ciclui repetitiv **foreach** sa apară referințe.

Astfel că în *PHP* 5 este posibilă execuția unui cod de forma:

```
foreach ($vector as &$valoare){
  if ($valoare === "NULL"){
    $value = null;
  }
}
```

Valori implicite pentru parametri transmiși prin referință

În limbajul *PHP* 5 pot fi declarate metode și funcții ai căror parametri care sunt transmiși prin referință pot avea valori implicite, și, în concluzie pot fi opționali.

În continuare vă prezentăm un exemplu de utilizare a acestei facilități.

```
function my_function(&$var = null){
  if ($var === null){
    echo "$var trebuie sa aiba o valoare";
}
```

Constanta __METHOD__

În limbajul *PHP* 5 a fost introdusă constanta `__METHOD__` a cărei valoare este dată de numele metodei în care constanta este folosită.

În continuare vă prezentăm un exemplu care ilustrează valoarea constantei `__METHOD__`.

```
class Clasa{
  function Afiseaza(){
    echo __FILE__.'('.__LINE__.')'.__METHOD__;
  }
}

function Test(){
  echo __FILE__.'('.__LINE__.')'.__METHOD__;
}

$obj = new Clasa();
$obj->Afiseaza();
Test();
```

În loc de încheiere...

În limbajul *PHP* 5 au fost introduse mult mai multe elemente decât cele prezentate în cadrul acestui articol.

Din păcate documentația disponibilă pe piață legată de acest limbaj este puțină și așteptăm și noi cu nerăbdare prezentări mai complete ale noilor facilități oferite de acest limbaj.

Versiunea 5 a limbajului *PHP* aduce cu ea biblioteci de funcții noi, dar mare parte din cele prezente și în versiunea 4 au fost optimizate din punct de vedere al vitezei de execuție a codului.

Bibliotecile nou introduse au fost realizate folosind facilitățile programării orientate obiect oferite de această versiune.

Mai mult, limbajul *PHP* 5 oferă suport pentru introspecție și reflexie, astfel că se pot afla foarte multe informații legate de clasele și membrii acestora și se pot afla informații legate de comentariile prezente în codul *PHP*.

Dacă sunteți interesați de această versiune a interpretorului *PHP* puteți să o descărcați de pe site-ul **www.php.net** și să testați voi înșivă câteva elemente din cele prezentate de-a lungul acestui articol, unele dintre ele fiind foarte interesante din foarte multe puncte de vedere.