Web programiranje

Vežbe 6 - OOP u PHP-u 5

Dok su neki jezici od početka projektovani kao objektno-orijentisani, sa PHP-om to nije bio slučaj. U svojim počecima, PHP je bio proceduralni skript jezik namenjen isključivo web primenama. Kako je vreme prolazilo, zajednica je tražila uvođenje objektno-orijentisanih koncepata u jezik, što je i učinjeno izdanjem PHP-a 4.

Međutim, ta prva implementacija je imala mnoge slabosti, kao što su problematično referenciranje objekata, nedostatak postavljanja *scope*-a (*public, private, protected, abstract*) za atribute i metode, nedostatak destruktora, kloniranja objekata i interfejsa. Srećom, PHP5 implementacija popravlja ove nedostatke i donosi još mnogo novih mogućnosti, na taj način stvrstavajući PHP u grupu jezika sa potpuno implementiranim objektnim modelom. Sav dalji tekst se odnosi upravo na PHP5 implementaciju koja je danas aktuelna.

Osnovne osobine tj. prednosti OOP-a kao koncepta su:

- apstrakcija
- enkapsulacija
- nasleđivanje
- polimorfizam

U daljem tekstu biće definisani osnovni pojmovi i operacije u OOP-u kod PHP5, kao što su klase, objekti, nasleđivanje, kloniranje objekata itd.

Definisanje klase i instanciranje objekata

```
<?php

class Zaposleni {
    // Atributi
    protected $ime;
    protected $plata;

    // Metode
    function  __construct($ime, $plata) {
        $this->ime = $ime;
        $this->plata = $plata;
        date_default_timezone_set('Europe/Belgrade');
    }

    public function stigao() {
        echo "Zaposleni $this->ime stigao je u ".date("h:i:s")."<br />";
    }

    public function otisao() {
        echo "Zaposleni $this->ime otisao je u ".date("h:i:s")."<br />";
    }
```

```
    public function  __toString() {
        return "Zaposleni: $this->ime sa platom $this->plata";
    }
}

$mile = new Zaposleni("Mile", 2000);
$mile->stigao();
sleep(1);
$mile->otisao();
echo $mile;
?>
```

Klasa **Zaposleni** ima dva *protected* atributa, konstruktor i tri metode od kojih je jedna specijalna (**__toString()**).

Konstruktor u jeziku PHP5 mora imati naziv **__construct** i može imati proizvoljno mnogo argumenata, kao i podrazumevane argumente. Za razliku od Jave, u PHP-u je moguće imati samo jedan konstruktor, ali se taj nedostatak lako nadomešta dinamičkim tipiziranjem, podrazumevanim argumentima ili tzv. fabričkim statičkim metodama.

Kada se obraćamo bilo kojem nestatičkom (vezanom za objekat) atributu ili metodi unutar koda same klase, moramo koristiti pokazivač **$this->,** što, kao i u Javi, predstavlja referenciranje objekta, tj. instance te klase. Za razliku od Jave, u PHP-u je $this obavezno!

Public, private i protected imaju isto značenje kao u Javi, sa izuzetkom *protected* koji dozvoljava pristup samo klasama koje datu klasu nasleđuju (u Javi ceo paket može da pristupi **protected** objektu). U OOP praksi uobičajeno je da atributi budu **private** ili **protected** i da im se pristupa preko metoda, tzv. *gettera* i *settera*. Gornjoj klasi bi se npr. mogle dodati za postavljanje i čitanje atributa **$plata**:

```
    public function postaviPlatu($plata) {
        $this->plata = $plata;
    }

    public function citajPlatu() {
        return $this->plata;
    }
```

koje bi se koristile na sledeći način:

```
$mile = new Zaposleni("Mile", 2000);
$mile->postaviPlatu(2500);
echo $mile->citajPlatu();
```

Konstante i statički atributi i metode

Klasne konstante se definišu koristeći ključnu reč **const** i imaju oblast važenja na nivou klase (ne objekta!). Treba naglasiti da se u PHP-u koriste različiti separatori za instance, tj. objekte i klase, tj. "**->**" i "**::**" respektivno, za razliku od Jave gde se koristi isključivo separator ".". Takođe, ključna reč **self**, se koristi kao oznaka aktuelne klase.

```
class matematicke_funkcije
{
    const PI = '3.14159265';
    const E = '2.7182818284';
    const OJLER = '0.5772156649';

    public function pi_kvadrat() {
        echo self::PI * self::PI."<br />";
    }

    public static function pi_kvadrat_static() {
        echo self::PI * self::PI."<br />";
    }
}
echo matematicke_funkcije::PI."<br />";
$m = new matematicke_funkcije();
$m->pi_kvadrat();
matematicke_funkcije::pi_kvadrat_static();
```

U gornjem primeru dat je i način na koji se definišu i pozivaju i statičke metode koje, naravno, smeju da pristupaju isključivo atributima i metodama sa klasnom oblasti važenja (konstante, statički atributi...).

Evo još jednog uobičajenog primera čiji je zadatak brojanje novoformiranih instanci klase:

```
class Posetilac {
    private static $posetioci = 0;

    function __construct() {
        self::$posetioci++;
    }
    static function citaj_Posetioce() {
        return self::$posetioci;
    }
}

// Instanciranje klase Posetilac
$pos1 = new Posetilac();
echo Posetilac::citaj_Posetioce()."<br />";
// Jos jedna instanca klase Posetilac
$pos2 = new Posetilac();
echo Posetilac::citaj_Posetioce()."<br />";
```

Destruktori

Iako se u PHP-u, kao i u Javi, instancirani objekti automatski uništavaju po izlazu iz oblasti važenja ili po izlazu iz samog skripta, ponekad je potrebno dodatno prilagoditi proces destrukcije objekta, npr. zatvoriti otvorene fajlove, obraditi greške itd... Destruktor je metoda bez argumenata, a koristi se na sledeći način:

```
<?php
    class Knjiga
    {
        private $naziv;
        private $isbn;
        private $broj_kopija;

        function __construct($isbn)
        {
           echo "<p>Kreirana instanca klase Knjiga.</p>";
        }

       function __destruct()
        {
           echo "<p>Instanca klase Knjiga unistena.</p>";
        }
   }

   $knjiga = new Knjiga("1893115852");
?>
```

Ključna reč instanceof

Isto kao i u Javi, **instanceof** dozvoljava da se proveri da li je objekat instanca tražene klase. Treba primetiti i da je objekat instanca klase **Klasa1**, i kada je instanciran kao objekat klase **Klasa2** koja je izvedena iz klase **Klasa1.**

```
$m = new Zaposleni();
...
if ($m instanceof Zaposleni) echo "Da";
```

a takođe je i u sledećem skriptu odgovor potvrdan:

```
// Rukovodilac je izvedena klasa klase Zaposleni - videti dole
$m = new Rukovodilac();
...
if ($m instanceof Zaposleni) echo "Da";
```

Nasleđivanje

Kao većina OO jezika, i PHP5 podržava nasleđivanje klasa, po zakonima sličnim ili čak istim kao u programskom jeziku Java. Naredni primer pokazuje kako se klasa **Zaposleni** nasleđuje klasom **Rukovodilac**, koja poseduje dodatni atribute **$zvanje** i **$bonus** i metode koje rukuju ovim atributima.

```
<?php

class Zaposleni {
    // Atributi
    protected $ime;
    protected $plata;

    // Metode
    function  __construct($ime, $plata) {
        $this->ime = $ime;
        $this->plata = $plata;
        date_default_timezone_set('Europe/Belgrade');
    }

    public function stigao() {
        echo "Zaposleni $this->ime stigao je u ".date("h:i:s")."<br />";
    }

    public function otisao() {
        echo "Zaposleni $this->ime otisao je u ".date("h:i:s")."<br />";
    }

    public function postaviPlatu($plata) {
        $this->plata = $plata;
    }

    public function citajPlatu() {
        return $this->plata;
    }

    public function  __toString() {
        return "Zaposleni: $this->ime sa platom $this->plata";
    }
}

class Rukovodilac extends Zaposleni {
    // Atributi
    private $zvanje;
    private $bonus;

    // Metode
    function  __construct($ime, $plata, $zvanje, $bonus) {
        parent::__construct($ime, $plata);
        $this->zvanje = $zvanje;
```

```
        $this->bonus = $bonus;
    }

    public function puna_plata() {
        return $this->plata + $this->bonus;
    }

    public function  __toString() {
        return "Rukovodilac: $this->zvanje $this->ime sa punom platom".
                $this->puna_plata().".<br />";
    }
}

$pera = new Rukovodilac("Pera", 4000, "dr", 200);
$pera->stigao();
// Cekaj 2 sekunde
sleep(2);
$pera->otisao();
echo $pera;
?>
```

Iz priloženog koda je očigledno da objekat klase **Rukovodilac** pored specifičnih osobina za tu klasu, nasleđuje i atribute i metode iz roditeljske klase. Često je potrebno pre izvršavanja specifičnog koda za datu klasu izvršiti istoimenu metodu iz roditeljske klase. Kako se to radi na primeru konstruktora klase **Rukovodilac**, vidi se iz gornjeg skripta. Ključna reč **parent** označava roditeljsku klasu, ali se isti efekat postiže i navođenjem imena klase:

```
parent::__construct($ime, $plata);
```

ima isti efekat kao i

```
Zaposleni::__construct($ime, $plata);
```

Specijalna metoda **__toString()** klase **Zaposleni** je preklopljena istoimenom metodom izvedene klase **Rukovodilac**.

Odvajanje koda u različite fajlove i automatsko učitavanje klasa

Norma programskog jezika Java nalaže da svaka javna klasa bude smeštena u posebnom fajlu. Jezik PHP po ovom pitanju nije imperativan, ali je svakako dobra praksa razdvojiti funkcionalnosti u posebne fajlove i po potrebi ih učitavati. Na primer:

Fajl *Zaposleni.php*:

```
<?php
class Zaposleni {
    protected $ime;
    protected $plata;

    function  __construct($ime, $plata) {
        $this->ime = $ime;
        $this->plata = $plata;
        date_default_timezone_set('Europe/Belgrade');
    }
```

```
    public function stigao() {
        echo "Zaposleni $this->ime stigao je u ".date("h:i:s")."<br />";
    }

    public function otisao() {
        echo "Zaposleni $this->ime otisao je u ".date("h:i:s")."<br />";
    }

    public function postaviPlatu($plata) {
        $this->plata = $plata;
    }

    public function citajPlatu($plata) {
        return $this->plata;
    }

    public function  __toString() {
        return "Zaposleni: $this->ime sa platom $this->plata";
    }
}
?>
```

Fajl *Rukovodilac.php*:

```
<?php
require_once 'Zaposleni.php';

class Rukovodilac extends Zaposleni {
    // Atributi
    private $zvanje;
    private $bonus;

    // Metode
    function  __construct($ime, $plata, $zvanje, $bonus) {
        parent::__construct($ime, $plata);
        $this->zvanje = $zvanje;
        $this->bonus = $bonus;
    }

    public function puna_plata() {
        return $this->plata + $this->bonus;
    }

    public function  __toString() {
        return "Rukovodilac: $this->zvanje $this->ime sa punom platom ".
                $this->puna_plata().".<br />";
    }
}
?>
```

Fajl *proba.php*:

```
<?php
require_once 'Zaposleni.php';
require_once 'Rukovodilac.php';

$pera = new Rukovodilac("Pera", 4000, "dr", 200);
$pera->stigao();
// Cekaj 2 sekunde
sleep(2);
$pera->otisao();
echo $pera;
?>
```

Funkcija PHP-a **require_once** omogućava da se tekst nekog spoljašnjeg fajla uključi u skript koji se izvršava. Ukoliko navedeni fajl ne postoji, prijavljuje se greška i prekida se izvršavanje skripta. Sufiks **_once** označava da se traženi fajl sme uključiti samo jednom. Ukoliko bi se u skripti *proba.php* umesto **require_once** upotrebio izraz **require** došlo bi do nepotrebnog redefinisanja klase **Zaposleni**.

Međutim, kako nazivi fajlova u kojima se nalaze klase obično slede neku internu konvenciju programera, PHP pruža još jedan zgodni mehanizam za automatsko uključivanje potrebnih klasa, tzv. **autoload**. Naime, umesto višestrukih require_once direktiva, u gornjem slučaju je dovoljna jedna specijalna funkcija **__autoload($ime_klase)**:

```
<?php
function __autoload($ime_klase) {
    require_once "$ime_klase.php";
}

$pera = new Rukovodilac("Pera", 4000, "dr", 200);
$pera->stigao();
// Cekaj 2 sekunde
sleep(2);
$pera->otisao();
echo $pera;
?>
```

Zadatak:

Napisati *PHP/JavaScript* skriptu koja prikazuje formu u koju je moguće uneti podatke o jednom zaposlenom, kao i podatke o jednom rukovodiocu, gde se prikaz dodatnih polja za zvanje i bonus kontroliše *checkbox* poljem "Rukovodilac". Skript zatim treba da odštampa unete podatke koristeći metode klasa Zaposleni ili Rukovodilac.

```
<html>
    <head>
        <title>Zaposleni/Rukovodilac</title>
        <script type="text/javascript">
            function rukovodilac_status() {
                var rukovodilac = document.getElementById("rukovodilac");
                var zvanje = document.getElementById("zvanje");
                var bonus = document.getElementById("bonus");

                if (rukovodilac.checked) {
                    zvanje.style.visibility = 'visible';
                    bonus.style.visibility = 'visible';
                }
                else {
                    zvanje.style.visibility = 'hidden';
                    bonus.style.visibility = 'hidden';
                }
            }
        </script>
    </head>
    <body>
        <?php
        function __autoload($ime_klase) {
            require_once "$ime_klase.php";
        }

        if (!isset ($_GET['posalji'])) {
        ?>
        <h1>Podaci o zaposlenom/rukovodiocu</h1>
        <form action="<?php echo $_SERVER['PHP_SELF'] ?>" method="GET">
          Rukovodilac <input type="checkbox" name="rukovodilac"
                id="rukovodilac" onchange="rukovodilac_status()"/><br />
          Ime: <input type="text" name="ime" /><br />
          Plata: <input type="text" name="plata" /><br />
          Zvanje: <input type="text" style="visibility:hidden"
                   name="zvanje" id="zvanje" /><br />
          Bonus: <input type="text" style="visibility:hidden"
                   name="bonus" id="bonus" /><br />
            <input type="submit" name="posalji" value="Posalji">
        </form>
        <?php
        }
        else {
            $ime = $_GET['ime'];
```

```
            $plata = $_GET['plata'];
            $zvanje = $_GET['zvanje'];
            $bonus = $_GET['bonus'];

            // Konstruisi instancu klase Zaposleni
            if (!isset ($_GET['rukovodilac'])) {
                $z = new Zaposleni($ime, $plata);
                echo $z;
            }
            // Konstruisi instsancu klase Rukovodilac
            else {
                $z = new Rukovodilac($ime, $plata, $zvanje, $bonus);
                echo $z;
            }
        }

        ?>
    </body>
</html>
```

-- Još malo nasleđivanja

```php
<?php
class A
{
    // more code here
}

class B extends A
{
    // more code here
}

class C extends B
{
    // more code here
}

$someObj = new A();
$someOtherObj = new B();
$lastObj = new C();
?>
```

Interfejsi

```
interface Nameable{
   public function getName();
   public function setName($name);
}

class Book implements Nameable{
   private $name;

   public function getName(){
      return $this->name;
   }

   public function setName($name){
      return $this->name=$name;
   }
}
```

Napomene:

1. Klasa ne može da implementira dva interfejsa koji imaju funkcije sa istim imenima, jer dovode do dvosmislenosti.
2. Interfejsi se mogu naslediti kao što je to slučaj sa klasom, korišćenjem ključne reči ***extends***.
3. Klasa koja implementira interfejs mora da koristi isti potpis metode koji je definisan u interfejsu.
4. Sve metode su public tipa
5. Interfejsi mogu da sadrže konstante

Primer 1:

```
<?php
interface a
{
    public function foo();
}

interface b extends a
{
    public function baz(Baz $baz);
}
// This will work
class c implements b
{
    public function foo()
    {
    }

    public function baz(Baz $baz)
    {
    }
```

```
}

?>
```

Primer 2

```
<?php
interface a
{
    const b = 'Interface constant';
}

// Prints: Interface constant
echo a::b;

This will however not work because its not allowed to
// override constants.
class b implements a
{
    const b = 'Class constant';
}
?>
```

Primer 3

```
<?php
interface ElectricalDevice{
  public function power_on();
  public function power_off();
}

interface FrequencyTuner{
  public function get_frequencey();
  public function set_frequency($f);
}

class ElectricFan implements ElectricalDevice{
  // define ElectricalDevice...
}

class MicrowaveOven implements ElectricalDevice{
  // define ElectricalDevice...
}

class StereoReceiver implements ElectricalDevice, FrequencyTuner{
  // define ElectricalDevice...
  // define FrequencyTuner...
}

class CellPhone implements ElectricalDevice, FrequencyTuner{
  // define ElectricalDevice...
  // define FrequencyTuner...
}

?>
```

Primer 4

```php
<?php
interface water
{
    public function makeItWet();
}

class weather
{
    public function start()
    {
        return 'Here is some weather';
    }
}
class rain extends weather implements water
{
    public function makeItWet()
    {
        return 'It is wet';
    }
}

$a = new rain();
echo $a->start();
echo $a->makeItWet();
?>
```

Izrada apstraktnih klasa

- Klasa koja se ne može direktno instancirati, a deluje kao zajednička osnova za klase potomke.
- Apstraktna klasa mora imati definiciju barem jedne apstraktne metode.
- Apstraktna klasa može sadržati i neapstraktne metode.
- Apstraktne metode se ne realizuju unutar apstraktnih klasa, kao što se ni metode definisane u interfejsu ne realizuju u njemu. Zato apstraktne metode treba realizovati u klasi potomku, koja proširuje apstraktnu roditeljsku klasu.

```php
<?php

abstract class person
{
    public $LastName;
    public $FirstName;
    public $BirthDate;

    abstract protected function write_info();

    public function get_Age($today=NULL){
        //age computation function
    }
}
```

```php
final class employee extends person{
    public $EmployeeNumber;
    public $DateHired;

    public function write_info(){
        echo "Writing ". $this->LastName . "'s info to emloyee dbase table";

    }
}

final class student extends person{
    public $StudentNumber;
    public $CourseName;

    public function write_info(){
        echo "Writing ". $this->LastName . "'s info to student dbase table";

    }
}

///----------
$personA = new employee;
$personB = new student;

$personA->FirstName="Joe";
$personA->LastName="Sbody";

$personB->FirstName="Ben";
$personB->LastName="Dover";

$personA->write_info();
?>
```

PHP dozvoljava postojanje samo jednog konstruktrora. Za slučaj da nam je potrebno više konstruktora problem se može rešiti na sledeće načine:

http://stackoverflow.com/questions/1699796/best-way-to-do-multiple-constructors-in-php
http://php.net/manual/en/language.oop5.traits.php
http://alfonsojimenez.com/post/30377422731/multiple-constructors-in-php

Magične metode:

http://php.net/manual/en/language.oop5.magic.php

Don't Call the Destructor Explicitly:
http://www.stoimen.com/blog/2011/11/14/php-dont-call-the-destructor-explicitly/