Web programiranje

Vežbe 5 - Funkcije i OOP

Funkcije

Funkcije se definišu koristeći ključnu reč **function** slično kao u nekim drugim programskim jezicima, recimo C-u ili Javi, kao:

```
function ime_funkcije(parametri)
{
       telo funkcije
       return povratna_vrednost;
}
```

Za razliku od Jave, parametri fukcije se mogu prenositi i po vrednosti i po referenci. Primer prenošenja parametara po vrednosti je:

```
<?php
function puna_cena($cena, $porez) {
    $total = $cena + $cena*$porez/100;
    return $total;
}

function obim_kruga($poluprecnik) {
    return 2*$poluprecnik*pi();
}

echo "Puna cena je ". puna_cena(100, 18);
echo "<br />Obim prvog kruga je ".sprintf("%.3f",obim_kruga(6));
echo "<br />Obim drugog kruga je ".sprintf("%.3f",obim_kruga(7));
?>
```

Zadatak: Napisati funkciju koja promeni_case($tekst, $kod) koja u zavisnosti od koda koji može biti “V” ili “M”, menja tekst u velika slova ili mala slova, respektivno.

```
<?php

function promeni_case($str, $kod='') {
    switch(strtoupper($kod)) {
        case 'V':
            return strtoupper($str);
            break;
        case 'M':
            return strtolower($str);
            break;
        case 'N':
            return $str;
            break;
    }
}

print promeni_case("Prirodno-matematicki fakultet Kragujevac<br/>", "M");
print promeni_case("Prirodno-matematicki fakultet Kragujevac<br/>", "V");
print promeni_case("Prirodno-matematicki fakultet Kragujevac<br/>");
?>
```

U gornjem primeru vidi se i kako se u PHP-u tretiraju podrazumevane vrednosti argumenata. Kao i u programskom jeziku Java, svi argumenti desno od prvog podrazumevanog argumenta takođe moraju imati podrazumevane vrednosti.

Funkcija može da vrati i niz, a ne samo jednu vrednost. Sledeći primer to demonstrira.

Zadatak: Korisnik unosi 8 e-mail adresa u 8 tekst polja unutar HTML forme. Napisati funkciju koja daje listu jedinstvenih domena izvučenih iz polja forme.

```
<html>
    <head>
        <title>Adrese i domeni</title>
    </head>
    <body>
    <?php
    // Definicija funkcije
    function jedinstveni_domeni($lista) {
        $domeni = array();
        foreach ($lista as $adresa) {
            $arr = explode("@", $adresa);
            if (isset ($arr[1])) $domeni[] = trim($arr[1]);
        }
        return array_unique($domeni);
    }

    // Ako forma jos nije poslata
    if (!isset ($_POST['posalji']))
    {
        echo "<form method='POST' action=".$_SERVER['PHP_SELF'].">";
        for ($i=1; $i<=8; $i++)
            echo "Adresa $i: <input type='text' name='adrese[]' /> <br/>";
        echo "<input type='submit' name='posalji' value='Posalji' />";
        echo "</form>";
    }
    // Obrada poslatih vrednosti
    else
    {
        $adrese = $_POST['adrese'];
        if ( is_array($adrese) )
        {
            foreach (jedinstveni_domeni($adrese) as $domen) {
                echo "<p>$domen</p>";
            }
        }
    }
    ?>
    </body>
</html>
```

Prosleđivanje parametara po referenci

U PHP-u je moguće proslediti parametar po referenci i na taj način ga učiniti mutabilnim. Mehanizam je sličan navođenju ključne reči VAR ispred parametra u programskom jeziku Pascal. Znak koji PHP koristi za referencu je &:

```
<?php
function zameni(&$a, &$b) {
    $temp = $a;
    $a = $b;
    $b = $temp;
}
```

```
$a = "Mika";
$b = "Pera";
echo "Pravi redosled: $a i $b.<br />";
zameni($a, $b);
echo "Obrnuti redosled: $a i $b.<br />";
?>
```

Drugi primer radi sortiranje niza korišćenjem jednostavnog algoritma *Bubble sort*, pri čemu se niz sortira "u mestu". Pravo sortiranje niza naravno treba raditi ugrađenom funkcijom **sort()**, koja je višestruko brža.

```
<?php

function zameni(&$a, &$b) {
    $temp = $a;
    $a = $b;
    $b = $temp;
}

function sortiraj(&$niz) {
    for($i=0; $i<count($niz)-1; $i++)
        for($j=$i+1; $j<count($niz); $j++)
            if ($niz[$i] > $niz[$j]) zameni($niz[$i], $niz[$j]);
}

$niz = array(1,5,1,7,4,6,8,2,4,1);
echo "Nesortirani niz je: ".implode(', ', $niz)."<br />";
sortiraj($niz);
echo "Sortirani niz je: ".implode(', ', $niz)."<br />";
?>
```

Pomoćne funkcije za rad sa argumentima

Skript jezici kao što je PHP omogućavaju i neke dodatne funkcionalnosti u odnosu na jezike sa statičkim tipiziranjem. Jedna od mogućnosti je da se funkcija definiše bez argumenata, a da joj se pošalje proizvojno mnogo:

```
<?php
// definicija funkcije
function neka_funkcija() {
// uzmi broj argumenata
    $numArgs = func_num_args();

    // uzmi argumente
    $args = func_get_args();

    // stampaj argumente
    print "Poslati su mi sledeci argumenti: ";
    for ($x = 0; $x < $numArgs; $x++) {
        print "<br />Argument $x: ";
        // ako je poslat niz, iterisi kroz njega i stampaj sadrzaj
        if (is_array($args[$x])) {
            print " NIZ ";
            foreach ($args[$x] as $index => $element) {
                print " $index => $element ";
            }
        }
        else {
            print " $args[$x] ";
```

```
        }
    }
}

// pozovi funkciju sa raznorodnim argumentima
neka_funkcija("red", "green", "blue", array(4,5), "yellow");
?>
```

Globalne promenljive

Sledeći primer demonstrira kako se iz funkcije operiše nekom globalnom promenljivom:

```
<?php

$danas = "Utorak";

function dan() {
    // varijabla je globalna
    global $danas;
    $danas = "Subota";
    // stampaj varijablu
    print "Unutar fukcije je $danas. <br />";
}

// print the variable
print "Pre poziva funkcije je $danas. <br />";
dan();
print "Posle poziva funkcije je $danas. <br />";
?>
```

Ukoliko bi se izostavila linija "**global $danas**" promenljiva **$danas** bila bi lokalna za funkciju.

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
- kapsulacija
- 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($plata) {
        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 oblasi 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 seu 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($plata) {
        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.klasa.php*:

```
<?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($plata) {
        return $this->plata;
    }

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

Fajl *Rukovodilac.klasa.php*:

```
<?php
require_once 'Zaposleni.klasa.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.klasa.php';
require_once 'Rukovodilac.klasa.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_ondce direktiva, u gornjem slučaju je dovoljna jedna specijalna funkcija **__autoload($ime_klase)**:

```
<?php
function __autoload($ime_klase) {
    require_once "$ime_klase.klasa.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.class.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>
```