APLIKACJE INTERNETOWE
PHP

TEMAT 4-04: Metody magiczne.

Autor dokumentu: Wojciech Galiński	czwartek, 17 listopada 2016 r.	351203 Technik informatyk
ŹRÓDŁA WIEDZY:	http://www.php.net/manual/pl, http://webmaster.helion.pl/index.php/kurs-php, http://pl.wikipedia.org/, http://pl.wikibooks.org/wiki/PHP, http://phpkurs.pl/, http://kursphp.com/.	

Zagadnienia obowiązkowe

1. **Pojęcie metody magicznej** – to metoda wywoływana przez interpreter w odpowiedzi na różne zdarzenia, a nie przez programistę.

   Można je poznać po tym, że ich nazwy zaczynają się od dwóch znaków podkreślenia („__”).

   W niniejszym temacie zostaną omówione następujące metody magiczne: „**__call**”, „**__get**” i „**__set**”, „**__toString**”, „**__isset**” i „**__unset**”, „**__autoload**”.

2. **Dostęp do nieistniejących metod klasy** – jeśli odwołujemy się do metody, która nie istnieje, odpowiada za to metoda magiczna „**__call**” wywoływana z dwoma parametrami: nazwa metody oraz tablica przekazanych do niej parametrów.

   PRZYKŁAD:

```
class TCena
{
   public function __call($nazwa, $param)
   {
      echo '<p>BŁĄD: brak metody "'. $nazwa. '" z parametrami: '. join(', ',$param).'. </p>';
   }
}
$cena = new TCena;
$cena->nieistniejaca_metoda(2, 3.5, true);
```

3. **Dostęp do nieistniejących pól klasy** – jeśli odwołujemy się do pola, które nie istnieje, odpowiadają za to następujące metody magiczne:

   ➔ ***__get($pole)*** – wywoływana automatycznie, gdy próbujemy odczytać wartość z nieistniejącego pola klasy;

   ➔ ***__set($pole, $wartosc)*** – wywoływana automatycznie, gdy próbujemy przypisać wartość do nieistniejącego pola klasy.

   PRZYKŁAD:

```
class TCena
{  private  $cena = array('netto'=>0.0, 'vat'=>0.23, 'brutto'=>0.0);
   …
   public function __get($symulowane_pole)
   {  switch ($symulowane_pole)
      {  case 'netto':  case 'vat':  case 'brutto':  return $this->cena[$symulowane_pole];  }
   }
   public function __set($symulowane_pole, $wartosc)
   {  switch ($symulowane_pole)
      {
         case 'netto':   $this->cena['netto'] = $wartosc;
                         $this->cena['brutto'] = $wartosc * (1 + $this->cena['vat']);
                         break;
         case 'vat':     if ($wartosc<0.0 || $wartosc >= 1.0)  break;
                         $this->cena['vat'] = $wartosc;
                         $this->cena['brutto'] = $this->cena['netto'] * (1 + $wartosc);
                         break;
         case 'brutto':  $this->cena['brutto'] = $wartosc;
                         $this->cena['netto'] = $wartosc / (1 + $this->cena['vat']);
      }
   }
   …
}
$cena_2 = new TCena(5);
$cena_2->netto = 20;    echo '<p>netto: '. $cena_2->netto. ', vat: '. $cena_2->vat.
                             ', brutto: '. $cena_2->brutto. '</p>';
$cena_2->brutto = 20;   echo '<p>netto: '. $cena_2->netto. ', vat: '. $cena_2->vat.
                             ', brutto: '. $cena_2->brutto. '</p>';
```

4. **Automatyczna konwersja obiektu do tekstu** – zajmuje się tym metoda magiczna „***__toString***”.

PRZYKŁAD:

```
class TCena
{  …
   public function __construct($cena = 0.0, $vat = 0.23)
   {
      if ($vat>=0.0 && $vat < 1.0)  $this->cena['vat'] = $vat;
      if ($cena>=0)
      {
         $this->cena['netto'] = $cena;   $this->cena['brutto'] = $cena * (1 + $vat);
      }
   }
   public function __toString()
   {
      return  'Cena: <b>'. number_format($this->cena['netto'], 2, ',', ' ').
              ' zł</b> (brutto: <b>'. number_format($this->cena['brutto'], 2, ',', ' ').
              ' zł</b>; w tym VAT: <b>'. 100*$this->cena['vat']. '%</b>)';
   }
   …
}
$cena_3 = new TCena(1, 0.07);
echo '<p>'. $cena_3. '</p>';
$cena_4 = new TCena(5);
echo '<p>'. $cena_4. '</p>';
```

5. **Symulowanie istnienia oraz usuwania niesitniejącego pola klasy** – gdy używamy metod magicznych „**__get**” i „**__set**”, to powinno się symulować także istnienie oraz usuwanie takiego pola. Robimy to za pomocą metod magicznych: „***__isset***” i „***__unset***”.

PRZYKŁAD 1: (kontynuacja definiowania klasy TCena)

```
class TCena
{  …
   public function __isset($zmienna)  {  return in_array($zmienna, array_keys($this->cena));  }
   …
}
$c_5 = new TCena(5);   var_dump(isset($c_5->netto), isset($c_5->vat), isset($c_5->brutto));
```

PRZYKŁAD 2:

```
class Symulator
{  private  $ma_nie_istniec = array();
   public function __isset($pole) { return !in_array($pole, $this->ma_nie_istniec); }
   public function __unset($pole) { $this->ma_nie_istniec[] = $pole; }
   public function __get($pole)
   {  return (isset($this->$pole)? 'jakaś wartość': 'BŁĄD: wartość nie istnieje');  }
}
$s = new Symulator;   unset($s->b);   echo '<pre>a = '. $s->a. "\nb = ". $s->b. '</pre>';
```

6. [*] **Automatyczne ładowanie** – umożliwia automatyczne ładowanie kodu z definicją klasy po napotkaniu pierwszej instrukcji tworzenia obiektu takiej klasy. Stosuje się je w celu zwolnienia programisty z obowiązku używania funkcji „**require**” albo „**require_once**”.

   Wyróżniamy 2 metody automatycznego ładowania:

   - ➔ ***za pomocą metody magicznej „__autoload”*** – poniższy kod działa pod warunkiem, że definicja klasy znajduje się w pliku „**.php**” o takiej samej nazwie, jak nazwa klasy;

     PRZYKŁAD: `function __autoload($className)  { require('./'.$className.'.php'); }`

   - ➔ ***za pomocą funkcji „spl_autoload_register” i „spl_autoload_unregister” (zlecane w PHP 5)*** – ta metoda pozbawiona jest mankamentu poprzedniego rozwiązania.

     PRZYKŁAD:

```
function Moj_loader($className)
{  echo 'Ładuję klasę '.$className.'<br />';
   require('lib/'.$className.'.class.php');
   return true;
}   spl_autoload_register('Moj_loader');  // rejestracja autoładowania
// Można od razu korzystać z klas
$obiekt1 = new Klasa1;   $obiekt2 = new Klasa2;   $obiekt3 = new Klasa1;
spl_autoload_unregister('Moj_loader');  // wyrejestrowanie autoładowania
```

Zadania

1. Przeanalizuj poszczególne fragmenty definicji klasy „TCena” łącząc je po kolei w 1 całość.