APLIKACJE INTERNETOWE
PHP

TEMAT 4-05: Dziedziczenie i polimorfizm.

Autor dokumentu: Wojciech Galiński	piątek, 18 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. **Dziedziczenie** – to wykorzystanie funkcjonalności zdefiniowanej klasy w nowotworzonej klasie.

   Klasa może dziedziczyć po innej klasie, co oznacza, że oprócz swoich własnych atrybutów oraz zachowań, uzyskuje także te pochodzące z klasy, z której dziedziczy. Klasa dziedzicząca nazywana jest klasą pochodną lub potomną (ang. subclass, derived class), zaś klasa, z której następuje dziedziczenie — klasą bazową (ang. superclass).

2. **Słowo kluczowe „protected" (chroniony)** – służy do udostępniania składników klas wewnątrz definicji klas pochodnych. Taki składnik dostępny jest także wewnątrz definicji klasy, w której został utworzony, ale nie jest dostępny dla użytkownika żadnej z tych klas.

   PRZYKŁAD:

```
class Pojazd
{   protected $marka = 'Pociąg Tuwima';   protected $cena = 20000000;   };
class Pociag extends Pojazd
{ public $liczba_wagonow;
  public function __construct($liczba_wagonow)
  {
    $this->liczba_wagonow = $liczba_wagonow; }
  public function wyswietl_info()
  { echo 'Marka: '. $this->marka. '<br />Cena: '. $this->cena.
         '<br />Liczba wagonów: '. $this->liczba_wagonow. '<br />'; }
};
$poc_1 = new Pociag(40);   $poc_1->wyswietl_info();
```

3. **Konstruktory i destruktory w dziedziczeniu** – podlegają dziedziczeniu tak samo jak i inne funkcje składowe klas (metody). Dostęp do konstruktora klasy bazowej uzyskujemy poprzez odwołanie się do nazwy klasy bazowej, co zostało pogrubione w poniższym przykładzie.

   PRZYKŁAD:

```
class Pojazd
{ private $marka;   private $cena;
  public function __construct($marka, $cena)
  {   $this->marka = $marka;   $this->cena = $cena;   }
};
class Pociag extends Pojazd
{ public $liczba_wagonow;
  public function __construct($marka, $cena, $liczba_wagonow)
  { Pojazd::__construct($marka, $cena);
    $this->liczba_wagonow = $liczba_wagonow;
  }
  public function wyswietl_info()
  { echo 'Marka: '. $this->marka. '<br />Cena: '. $this->cena.
         '<br />Liczba wagonów: '. $this->liczba_wagonow. '<br />'; }
};
```

4. **Instrukcja „instanceof"** – sprawdza, czy obiekt pochodzi bezpośrednio lub pośrednio od podanej klasy.

   PRZYKŁAD: `var_dump($poc_1 instanceof Pociag, $poc_1 instanceof Pojazd);`

5. **Przysłanianie (unieważnianie) metod w klasach pochodnych** – to ponowne definiowanie metody o tej samej nazwie w klasie pochodnej. Taka metoda przysłania metodę z klasy bazowej i metoda z klasy bazowej nie jest już dostępna.

   PRZYKŁAD:

```
class A   {   public function xyz() { return 1; }   };
class B extends A   {   public function xyz() { return 2; }   };
$a = new A();   $b = new B();   echo $a->xyz(). ' '. $b->xyz();
```

6. **Klasa lub metoda abstrakcyjna (abstract)** – to klasa lub metoda, która musi zostać zdefiniowana (przysłonięta) w klasie pochodnej.
7. **Klasa lub metoda finalna (final)** – to klasa lub metoda, która nie może zostać zdefiniowana (przysłonięta) w klasie pochodnej.

   PRZYKŁAD:
```
abstract class A
{
  abstract public function abc();
  final public function xyz() { return 2; }
};
class B extends A
{
  public function abc() { return 11; }
  // public function xyz() { return 12; } // nie można przysłonić tej metody
};
$b = new B();   echo $b->xyz(). ' '. $b->abc();
```
8. **Statyczne elementy w dziedziczeniu** – z poziomu klasy pochodnej mamy dostęp do statycznych elementów klasy bazowej. Możemy je także przysłaniać.

   PRZYKŁAD:
```
class X
{   static protected $w_1 = 5;   static protected $w_2 = 10;   };
class Y extends X
{
  static public $w_2 = 15;
  static public $w_3 = 20;
  static public function pokaz_w()
  {   return parent::$w_1. ' '. parent::$w_2; }
};
$y_1 = new Y();   $y_2 = new Y();
echo Y::pokaz_w(). ' '. y::$w_2. ' '. Y::$w_3;
```
9. **Polimorfizm** – to automatyczne wybieranie metody z właściwej klasy pochodnej.

   PRZYKŁAD:
```
function niech_ruszy($pojazd) {  $pojazd->rusz(); }
class Tpojazd
{ private $marka, $model; public function rusz() { echo 'Pojazd rusza.<br />'; } }
class TBus extends TPojazd { public function rusz(){ echo 'Bus rusza.<br/>'; }};
$bus = new TBus;  niech_ruszy($bus);
class TTir extends TPojazd { public function rusz() { echo 'Tir rusza. <br />'; } };
$tir = new TTir;  niech_ruszy($tir);
```
10. [*] **Interfejs** – to rodzaj klasy zawierającej tylko nazwy i parametry metod (bez treści). Powstał jako częściowe rozwiązanie problemu braku obsługi dziedziczenia po wielu klasach bazowych (wielodziedziczenia).

Zadania

1. Zapoznaj się z przedstawionymi przykładami. Wykonaj je i przeanalizuj ich działanie.
2. Sprawdź w materiałach z języka C++, czym jest polimorfizm.