

PHP – programowanie obiektowe

Beata Pańczyk - PHP (Wykład 5)

Plan wykładu

- Definicja klasy, konstruktory i destruktory
- Specyfikatory dostępu
- Składniki statyczne (static)
- Składniki stałe (const)
- Klasy i metody finalne (final)
- Klasy abstrakcyjne i interfejsy
- Obiektowy szablon strony
- Biblioteka PEAR

Obiektowość w PHP5

- W PHP5 wprowadzono zupełnie nowy model obiektowości (w PHP4 – bardzo prymitywny, <PHP4 – brak)
- model zdefiniowany praktycznie od początku ale oferujący szersze możliwości tworzenia programów w pełni obiektowych (jak C++ czy Java)
- większość cech dotyczących obiektowości opiera się na zasadach języka C++

Przykład - Klasa i obiekt

```
<?php
    class A
    { //deklaracje pól
        public $a = 'wartość domyślna';
        //deklaracje metod
        public function wyswietl_a()
        {      echo $this->a; }
    }

    $obiekt = new A();    //utworzenie obiektu klasy A
?>
```

Automatyczne dołączanie obiektów

- W praktyce programowania obiektowego zwykle tworzy się osobne skrypty z definicją klasy, które następnie dołącza do skryptu głównego za pomocą funkcji `include` (dla każdej klasy osobno). W wyniku uzyskuje się długą listę z wywołaniem `include`.
- w PHP 5 istnieje możliwość wykorzystania funkcji (`_autoload function`), która automatycznie wykona dołączanie odpowiednich klas do kodu skryptu głównego. Funkcja taka będzie automatycznie wywoływana w momencie próby tworzenia obiektu klasy nie znanej jeszcze w skrypcie (jest to ostatnia próba dołączenia klasy przed wysłaniem komunikatu o błędzie).

Przykład - Funkcja autoload

- Zakładamy, że skrypty **Klasa1.php** i **Klasa2.php** zawierają definicje odpowiednich klas
- Skrypt poniżej dołącza klasy z tych plików.

```
<?php
```

```
function __autoload($klasa)  
{ require_once $klasa.'.php';}
```

```
$ob1 = new Klasa1();
```

```
$ob2 = new Klasa2();
```

```
?>
```

Konstruktory i destruktory

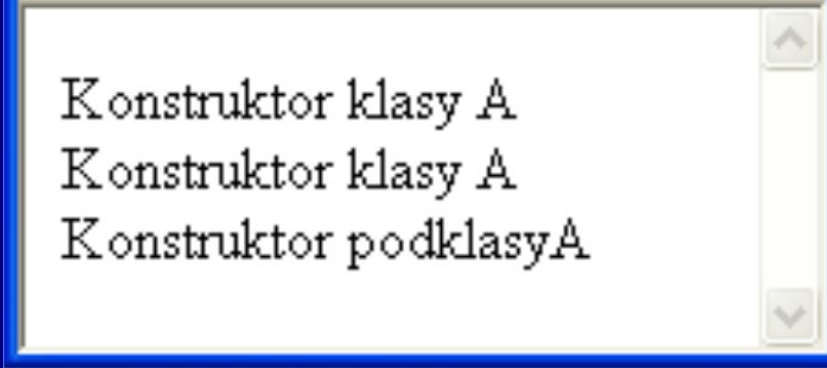
- PHP 5 pozwala deklarować konstruktory klas, wywoływane automatycznie w momencie tworzenia nowego obiektu klasy; konstruktor (i destruktor) klasy bazowej musi być jawnie wywoływany w klasie pochodnej.
- Konstruktor ma postać: `__construct ([argumenty [, ...]])`
- w celu zachowania kompatybilności z wersjami poprzednimi, jeśli PHP 5 nie może znaleźć funkcji `__construct()` w danej klasie, poszukuje konstruktora w starym stylu tzn. funkcji o nazwie takiej jak nazwa klasy.
- W PHP 5 istnieje destruktor, wywoływany w momencie kiedy usunięte są już wszystkie referencje do danego obiektu – obiekt jest wówczas likwidowany.
- Destruktor jest postaci: `__destruct ()`

Przykład - Konstruktor klasy bazowej i pochodnej

```
<?php
class KlasaA {
    function __construct()
    {
        print "Konstruktor klasy A <br />";
    }
}

class PodklasaA extends klasaA {
    function __construct()
    {
        parent::__construct();
        print "Konstruktor podklasyA<br />";
    }
}
```

```
$obj = new KlasaA();
$obj = new PodklasaA();
?>
```



Konstruktor klasy A
Konstruktor klasy A
Konstruktor podklasyA

Przykład - Destruktor

```
<?php
class KlasaA {
    protected $name="abc";
    function __construct()
    {   print "Konstruktor klasy A <br />";
    }
    function __destruct()
    {   echo "Destruktor klasy A:". $this->name."<br />";
    }
}
```

```
$obj = new KlasaA();
```

```
?>
```

```
Konstruktor klasy A
Destruktor klasy A:abc
```

Specyfikatory dostępu

- Dostęp do pól i metod klasy można definiować za pomocą specyfikatorów:
 - **public** (nieograniczony dostęp do składników klasy)
 - **protected** (dostęp do składników klasy mają tylko klasy od niej pochodne)
 - **private** (niemożliwy dostęp spoza klasy)
- Pola i metody klasy muszą być definiowane za pomocą jednego z trzech powyższych specyfikatorów. Brak specyfikatora dostępu oznacza dostęp **public**.
- Deklaracja pól poprzedzona słowem **var** z PHP4 nie jest już stosowana w PHP5. W celu zachowania kompatybilności z PHP4 – zmienne poprzedzone **var** są traktowane jako **public**.

Składniki static

- Deklaracja składników klasy jako statyczne umożliwia odwołanie się do tych składników za pomocą nazwy klasy w przypadku gdy nie istnieje jeszcze żaden obiekt tej klasy
- Składniki statyczne nie mogą być modyfikowane na zewnątrz klasy ani w jej klasach pochodnych
- Deklaracja `static` umieszczana jest po specyfikatorze dostępu
- Jeśli nie istnieje specyfikator dostępu (PHP 4) wtedy statyczne składniki klasy traktowane są jako `public`
- Ponieważ metody statyczne są wywoływane na rzecz samej klasy to autoreferencja `$this` nie może być stosowana w metodach statycznych
- **UWAGA!** - Do pól statycznych nie można odwoływać się z obiektu za pomocą operatorów: `->` i `::`

Przykład - Składniki statyczne

```
<?php
class A {
    public static $pole_stat='A';
    public static function funkcja_stat() { }
    public function fa()
    { return self::$pole_stat;}
}
class B extends A{
    public function fb()
    { return parent::$pole_stat;}
}
print A::$pole_stat."<br />";
$a = new A();
print $a->fa()."<br />";
print $a->pole_stat."- blad <br />";
//print $a::pole_stat."- bład! <br />";
```

```
print B::$pole_stat."<br />";
$b = new B();
print $b->fa()."<br />";
A::funkcja_stat();    ?>
```

A
A

Strict Standards: Accessing static property A::\$pole_stat as non static in C:\xampp\htdocs\TestAjax\php2.php on line 16

Notice: Undefined property: A::\$pole_stat in C:\xampp\htdocs\TestAjax\php2.php on line 16
- blad

A
A

Przykład - Składniki const

- Nazwa składnika stałego nie zawiera symbolu \$. Tak jak w przypadku składników statycznych – składniki stałe są dostępne poprzez nazwę klasy i nie są dostępne z obiektu poprzez operator ->.

```
<?php
class A {
    const N=10;
    function wyswietl_N()
    { echo self::N."<br />"; }
}

echo A::N."<br />";
$a = new A();
$a->wyswietl_N();
echo $a::N; //- poprawne od wersji 5.3
echo $a->N; //- zabronione!
?>
```

```
10
10
10
```

Notice: Undefined property: A::\$N in
C:\xampp\htdocs\TestAjax\php2.php
on line 11

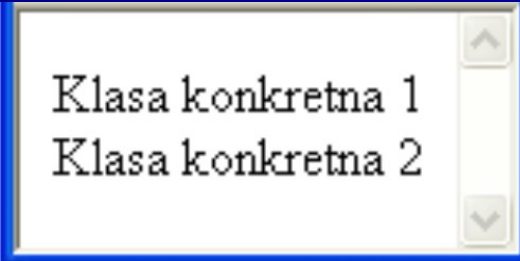
Klasy i metody abstrakcyjne

- PHP 5 udostępnia klasy i metody abstrakcyjne
- Nie można tworzyć instancji (obiektów) klasy abstrakcyjnej
- Każda klasa zawierająca nawet jedną metodę abstrakcyjną musi być zdefiniowana jako abstrakcyjna
- Metody zdefiniowane jako abstrakcyjne deklarują tylko nazwę metody i nie mogą jej implementować
- Klasa implementująca metodę abstrakcyjną musi ją definiować z tym samym poziomem dostępu lub **public**
- Jeśli klasa abstrakcyjna deklaruje metodę jako **protected** to jej konkretna implementacja powinna ją definiować jako **protected** lub **public**.

Przykład - Klasy i metody abstrakcyjne

```
<?php
abstract class Klasa_abstrakcyjna{
    // klasa pochodna musi zdefiniować tę metodę:
    abstract protected function wartosc();
    // metoda nieabstrakcyjna:
    public function drukuj()
    { print $this->wartosc()."<br />";}
}
class Klasa_konkretna1 extends Klasa_abstrakcyjna
{ protected function wartosc()
  { //implementuje metodę abstrakcyjną
    return "Klasa konkretna 1";}
}
class Klasa_konkretna2 extends Klasa_abstrakcyjna
{ protected function wartosc()
  { return " Klasa konkretna 2"; }
}
```

```
$o1 = new Klasa_konkretna1;
$o1->drukuj();
$o2 = new Klasa_konkretna2;
$o2->drukuj();
?>
```



Klasa konkretna 1
Klasa konkretna 2

Interfejsy

- Interfejs obiektowy umożliwia grupowanie metod, które następnie muszą być zaimplementowane w klasach korzystających z interfejsu
- Interfejs jest definiowany za pomocą słowa kluczowego **interface**, w ten sam sposób jak zwykła klasa z podaniem tylko deklaracji metod (bez ciała)
- Klasa, która implementuje dany interfejs (implements) musi zawierać definicję wszystkich metod tego interfejsu
- Klasa może implementować jeden lub więcej interfejsów (oddzielonych przecinkami)
- Wszystkie metody interfejsu muszą być publiczne (co wynika z natury samego interfejsu)
- Brak implementacji metody z interfejsu powoduje pojawienie się błędu (fatal error)

Interfejsy - uwagi

- Klasa nie może implementować dwóch interfejsów, które posiadają tak samo nazwane metody
- Interfejsy mogą po sobie dziedziczyć tak jak zwykłe klasy (extends)
- Klasa implementująca interfejs musi definiować dokładnie takie same metody jak zadeklarowano w interfejsie – w przeciwnym razie pojawi się błąd (fatal error)
- Interfejs może zawierać składniki stałe, które zachowują własności stałych zwykłych klas – nie mogą być jednak przesłanianie przez klasy/interfejsy po nich dziedziczące

Przykład - Interfejs

```
interface a
{
    public function fa();
}
```

```
interface b
{
    public function fb();
}
```

```
interface c extends a, b
{
    public function fc();
}
```

```
class d implements c
{
    public function fa()
    {
    }

    public function fb()
    {
    }

    public function fc()
    {
    }
}
```

Przykład - metoda final

- Wprowadzone w PHP 5 słowo kluczowe **final** zabrania predefiniowania tak określonej metody w klasach pochodnych. Jeśli cała klasa jest zdefiniowana jako **final**, to nie można po niej dziedziczyć

<?php

```
class A {  
    public function test1() { echo "Wywołano A::test() \n"; }  
    final public function test2() { echo "Wywołano A::test2()\n"; }  
class B extends A {  
    public function test2() { echo "Wywołano B::test2()\n"; }  
}
```

?>

Fatal error: Cannot override final method A::test2()
in C:\xampp\htdocs\TestAjax\php2.php on line 9

Przykład - klasa final

```
<?php
final class A {
    public function test()
    { echo "Wywolano A::test()\n"; }
    // dalej nie ma już znaczenia określenie metody – może być final lub nie
    final public function test2()
    { echo "Wywolano A::test2()\n"; }
}
class B extends A {
//Błąd – nie można dziedziczyc po klasie final
}
?>
```

Obiektoowość w PHP 5 - podsumowanie

- Model obiektowy w PHP 5 pozwala stosować bardziej wyrafinowane mechanizmy obiektowe jak klasy abstrakcyjne czy interfejsy
- Stwarza o wiele szersze niż w PHP 4 możliwości wykorzystania techniki OOP w programowaniu aplikacji internetowych i bazodanowych
- W PHP 5 technikę tę można stosować korzystając ze wszystkich możliwości oferowanych przez typowo obiektowe języki programowania.

Przykład 1 – obiektowy licznik Licznik.php

```
<? class Licznik{  
    private $ile;  
    private $plik;  
    function __construct()  
    { if (!(file_exists("licznik.txt")))  
        { $this->plik=fopen("licznik.txt","w+");  
          fputs($this->plik,"0");  
          $this->ile=0;  
        }  
    else  
    { $this->plik=fopen("licznik.txt","r+");  
      if (!$this->plik) {echo "Nie da się otworzyc pliku."; exit;}  
    }
```

Przykład 1 – obiektowy licznik Licznik.php

```
flock($this->plik, 2);  
$this->ile=fgets($this->plik,255);  
$this->ile++;  
fseek($this->plik,0);  
fputs($this->plik,$this->ile);  
flock($this->plik,3);  
}  
  
}  
  
public function __destruct() {fclose($this->plik);}  
public function ile_odwiedzin() {return $this->ile;}  
}  
//koniec klasy Licznik  
?>
```

Przykład 1 – Testlicznika.php

<?

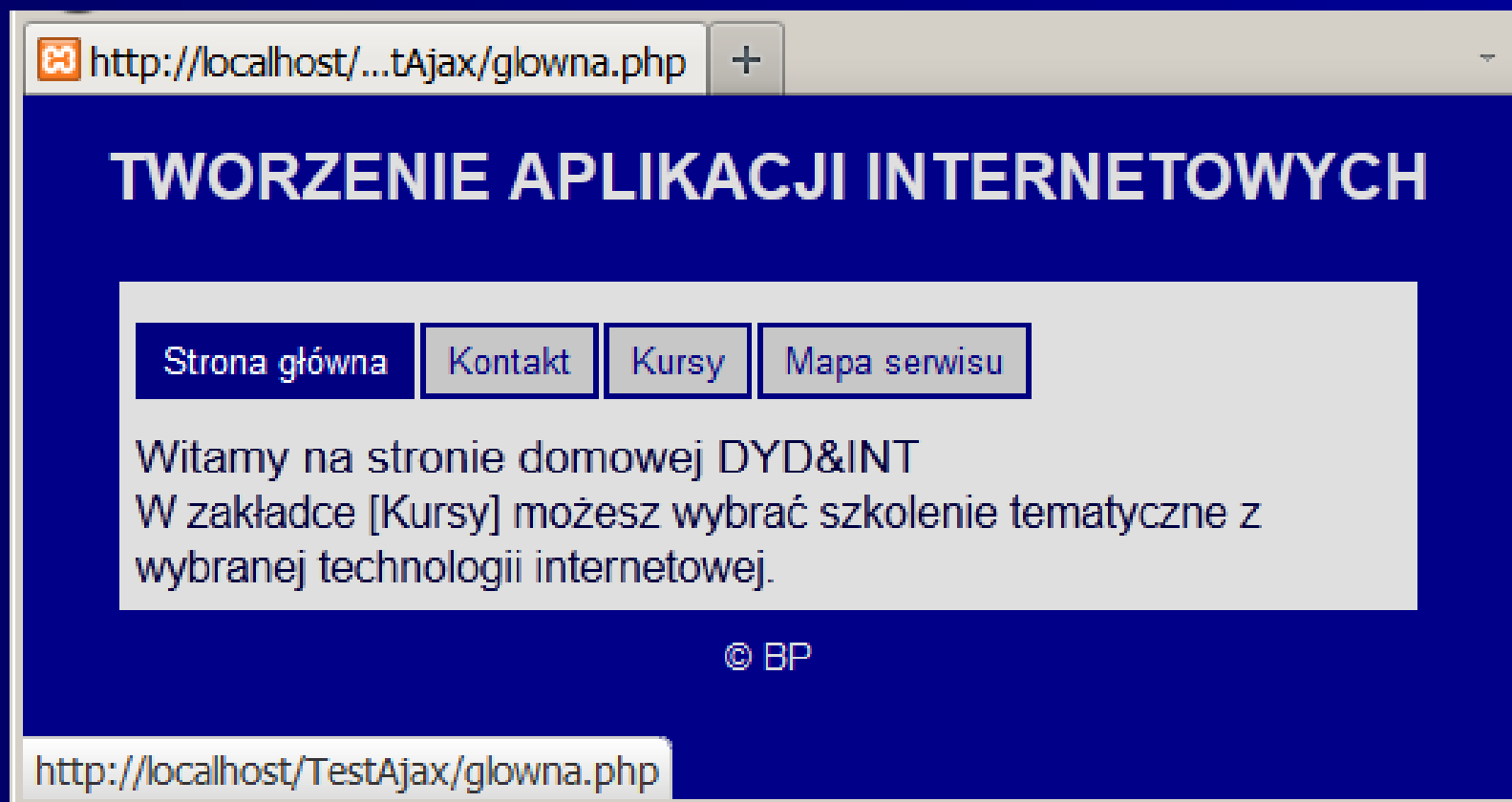
```
include("Klasalicznik.php");  
$licznik=new Licznik();  
echo "Test licznika obiektowego:";  
echo "<h3>Stronę odwiedzono:". $licznik->ile_odwiedzin()  
    ." razy.</h3>";
```

?>

Test licznika obiektowego:

Stronę odwiedzono:5 razy.

Przykład 2 - obiektowy szablon strony



Przykład 2 – klasa Strona.php

```
<?php
class strona    //początek definicji klasy
{
    //własności klasy:
    protected $zawartosc;
    protected $tytul="TWORZENIE APLIKACJI INTERNETOWYCH";
    protected $slowa_kluczowe="HTTP, HTML, CSS, XML,
        JavaScript, AJAX, PHP";
    protected $przyciski = array (        "Strona
        główna"=>"glowna.php",
        "Kontakt"=>"kontakt.php",
        "Kursy"=>"kursy.php",
        "Mapa serwisu"=>"mapa.php"
    );
}
```

Przykład 2 – klasa

Strona.php

//interfejs klasy:

```
function ustaw_zawartosc($nowa_zawartosc)
{ $this->zawartosc=$nowa_zawartosc; }
```

```
function ustaw_tytul($nowy_tytul)
{ $this->tytul=$nowy_tytul; }
```

```
function ustaw_slowa_kluczowe($nowe_slowa)
{ $this->slowa_kluczowe=$nowe_slowa; }
```

```
function ustaw_przyciski($nowe_przyciski)
{ $this->przyciski=$nowe_przyciski; }
```

```
function ustaw_style($url)
{ echo '<link rel="stylesheet" href=".' . $url . '
  type="text/css" />'; }
```

Przykład 2 – klasa Strona.php

```
function wyswietl()  
{  
    $this->wyswietl_naglowek();  
    $this->wyswietl_zawartosc();  
    $this->wyswietl_stopke();  
}
```

```
function wyswietl_tytul()  
{ echo "<title>$this->tytul</title>"; }
```

```
function wyswietl_slowa_kluczowe()  
{ echo "<meta name=\"keywords\" contents=\"  
    $this->slowa_kluczowe\">";  
}
```

Przykład 2 – klasa Strona.php

```
function wyswietl_naglowek()
{ ?>
<!DOCTYPE html>
<html>
<head>
  <title> Strona </title>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
    scale=1.0">
  <?php $this->ustaw_style('bp.css');
    echo "</head><body>";
    echo "<div id='nag'><h1>";
    echo $this->tytul."</h1></div>";
  }
function wyswietl_menu()
{ echo "<div id='nav'>";
  while (list($nazwa,$url)=each($this->przyciski))
    echo ' <a href="'. $url. "'>'. $nazwa. '</a>';
    echo "</div>";
}
```

Przykład 2 – klasa Strona.php

```
function wyswietl_zawartosc()
{
    echo "<div id='tresc'>";
    $this->wyswietl_menu();
    echo $this->zawartosc."</div>";
}
```

```
function wyswietl_stopke()
{
    echo '<div id="stopka"> &copy; BP</div>';
    echo '</body></html>';
}
```

```
} //koniec definicji klasy strona
```

```
?>
```

Przykład 2 – strona glowna.php

```
<?php
    require_once("strona.php");
    $strona_glowna = new strona();
    $tresc="<h2> Witamy na stronie
domowej DYD&INT</h2>
<h3> W zakładce [Kursy] możesz wybrać szkolenie
tematyczne z wybranej technologii internetowej.
</h3>";    //tresc może być pobrana np. ze
wskazanego pliku
    $strona_glowna->ustaw_zawartosc($tresc);
    $strona_glowna->wyswietl();
?>
```

Przykład 2 – fragment pliku bp.css

```
body {  
    text-align:center;  
    margin: 0;  
    padding: 0;  
    background:#000088 url(images/img01.jpg) repeat-x left  
    top;  
    font-size: small;  
    font-family:"Lucida Grande", sans-serif;  
    color: #000040;  
}  
h1, h2, h3, h4 {      margin: 0;font-weight: normal;}  
h1{font-size:180%;font-  
    weight:bold;color:#dfdfdf;padding:15px;}
```


Referencje do obiektów

- Często uważa się, że jednym z kluczowych punktów PHP 5 OOP jest "domyślne przekazywanie obiektów przez referencje,, co niezupełnie jest prawdą
- Referencja w PHP jest aliasem, który pozwala aby dwie różne zmienne nadpisywały tę samą wartość
- Od PHP 5 zmienne obiektowe nie definiują obiektu poprzez wartości, tworzą jedynie identyfikator obiektu, dzięki któremu można odnaleźć rzeczywisty obiekt
- Kiedy obiekt jest przekazywany w argumencie, zwracany w wyniku lub przypisywany do innej zmiennej – różne zmienne nie są aliasami: przechowują one kopię identyfikatora, który wskazuje na ten sam obiekt.

Przykład – obiekty i referencje

```
<?php
class A {    public $pole = 1; }

$a = new A;
$b = $a; // $a i $b są kopiami tego
          // samego identyfikatora
          // ($a) = ($b) = <id>
$b->pole = 2;
echo $a->pole."<br />";

$c = new A;
$d = &$c; // $c i $d są referencjami
          // ($c,$d) = <id>
$d->pole = 2;
echo $c->pole."<br />";
```

```
$e = new A;

function fun($obj) {
    // ($obj) = ($e) = <id>
    $obj->pole = 3;
}

fun($e);
echo $e->pole."<br />";

?>
```



2
2
3

Obsługa wyjątków (1)

```
<?php
function inverse($x) {
    if (!$x) { throw new Exception('Dzielenie przez zero.');
```



```
    else return 1/$x;
}

try {    echo inverse(5) . "<br/>"; }
catch (Exception $e) {
    echo 'Wyjatek: '. $e->getMessage(). "<br/>";}
finally {
    echo "Pierwsza klauzula finally<br/>";
}
```

Obsługa wyjątków (2)

```
try {  
    echo inverse(0) . "<br/>";  
} catch (Exception $e) {  
    echo 'Wyjątek: ' . $e->getMessage(). "<br/>"; }  
finally {  
    echo "Druga klauzula finally<br/>";}
```

// Ciąg dalszy skryptu

?>

Rozszerzenie PEAR

- PEAR "PHP Extension and Application Repository" - biblioteka klas posiadająca wsparcie twórców PHP
- Dokumentacja do pakietów PEAR i same pakiety dostępne pod adresem <http://pear.php.net>
- Celem PEAR jest dostarczenie:
 - otwartej biblioteki funkcji dla użytkowników PHP
 - standardów programowania w PHP
 - systemu dystrybucji kodu i zarządzania pakietami
 - rozszerzeń biblioteki (PECL)
 - wsparcia dla społeczności twórców PHP/PEAR
- **Pyrus** - następna generacja aplikacji PEAR (the best installation tool for PHP)

Przykładowe pakiety

PEAR

- Authentication
- Database
- Date and Time
- Encryption
- Event
- File Formats
- File System
- HTML
- HTTP
- Images
- Internationalization
- Logging
- Mail
- Math
- Networking
- Numbers
- PEAR
- PHP
- Streams
- Structures
- System
- Text
- Tools and Utilities
- Validate
- Web Services
- XML