

PRZYDATNY KOD
ZAWSZE POD RĘKĄ!

PHP i MySQL

ROZMÓWKI

Christian Wenz



Helion



Tytuł oryginału: PHP and MySQL Phrasebook

Tłumaczenie: Daniel Kaczmarek

ISBN: 978-83-246-7023-9

Authorized translation from the English language edition, entitled: PHP AND MYSQL PHRASEBOOK; ISBN 0321834631; by Christian Wenz; published by Pearson Education, Inc, publishing as Addison Wesley.

Copyright © 2013 Pearson Education, Inc.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

Polish language edition published by HELION S.A. Copyright © 2013.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Wydawnictwo HELION dołożyło wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie bierze jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Wydawnictwo HELION nie ponosi również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
<ftp://ftp.helion.pl/przyklady/phmsro.zip>

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
<http://helion.pl/user/opinie/phmsro>
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubią to! » Nasza społeczność](#)

Spis treści

O autorze	11
Wprowadzenie	13
Wprowadzenie do drugiego wydania	17
1 Operacje na ciągach znaków	21
Porównywanie ciągów znaków	22
Sprawdzanie nazw użytkowników i haseł	23
Przekształcanie ciągów znaków w kod HTML	25
Stosowanie znaków nowego wiersza	29
Szyfrowanie ciągów znaków	29
Sprawdzanie sum kontrolnych ciągów znaków	31
Wyodrębnianie fragmentów ciągów znaków	35
Zabezpieczanie adresów poczty elektronicznej przy użyciu kodów ASCII	36
Skanowanie sformatowanych ciągów znaków	41
Uzyskiwanie szczegółowych informacji na temat zmiennych	43
Wyszukiwanie w ciągach znaków	44
Stosowanie wyrażeń regularnych zgodnych z Perlem	48
Znajdowanie znaczników przy użyciu wyrażeń regularnych	49

Weryfikacja obecności danych wymaganych	50
Weryfikacja poprawności adresów poczty elektronicznej	54
Wyszukiwanie i zastępowanie	56
2 Korzystanie z tablic	59
Odczytywanie wszystkich elementów z tablic numerycznych	61
Odczytywanie wszystkich elementów z tablic asocjacyjnych	63
Odczytywanie wszystkich elementów z tablic zagnieżdżonych	65
Przekształcanie zawartości tablicy w zmienne	68
Przekształcanie ciągów znaków w tablice	69
Przekształcanie tablic w ciągi znaków	70
Alfabetyczne sortowanie zawartości tablic	71
Alfabetyczne sortowanie zawartości tablic asocjacyjnych	73
Sortowanie zawartości tablic zagnieżdżonych	75
Sortowanie zagnieżdżonych tablic asocjacyjnych	77
Sortowanie adresów IP w sposób naturalny	79
Sortowanie dowolnych wartości	81
Sortowanie ciągów w różnych językach	82
Przetwarzanie wszystkich elementów tablicy	85
Filtrowanie tablic	89
Odczytywanie z tablicy losowo wybranych elementów	91
Nadawanie obiektom zachowania charakterystycznego dla tablic	93
3 Data i czas	97
Używanie danych tekstowych w funkcji <code>date()</code>	100
Formatowanie obiektów <code>DateTime</code>	102
Automatyczna lokalizacja dat	103
Ręczna lokalizacja dat	107
Odczytywanie bieżącej daty w formatach amerykańskim, brytyjskim i europejskim	108

Formatowanie z góry określonej daty	109
Weryfikacja poprawności daty	111
Obliczanie daty względnej	112
Tworzenie znacznika czasu, który można sortować	113
Przekształcanie ciągu znaków w datę	115
Ustalanie czasu wschodu i zachodu słońca	116
Używanie daty i czasu dla celów porównawczych	118
Zastosowanie pól formularzy do wyboru daty	120
Formularz do wyboru daty, który sam się uaktualnia	122
Obliczanie różnicy między dwiema datami	124
Zastosowanie daty i czasu GMT	128
4 Praca z obiektami (i zagadnienia pokrewne)	131
Definiowanie klas	132
Dziedziczenie	134
Korzystanie z abstrakcyjnych klas i interfejsów	136
Zapobieganie dziedziczeniu i pokrywaniu	141
Automatyczne ładowanie	142
Klonowanie obiektów	145
Serializacja i deserializacja obiektów	147
Implementowanie singletonów	149
Stosowanie cech	154
5 Przetwarzanie formularzy internetowych	159
Wysyłanie danych z formularza z powrotem do skryptu	161
Odczytywanie danych z formularza	162
Sprawdzanie, czy formularz został wysłany na serwer	164
Zapisywanie danych z formularza w pliku cookie	166
Wypełnianie pól tekstowych i pól haseł wartościami predefiniowanymi	169

Wypełnianie wartościami predefiniowanymi wielowierszowych pól tekstowych	173
Wstępne zaznaczanie pól opcji	175
Wstępne zaznaczanie pól wyboru	177
Wstępne zaznaczanie pozycji na listach wyboru	178
Wstępne zaznaczanie pozycji na listach wielokrotnego wyboru	180
Przetwarzanie graficznych przycisków przesyłania danych z formularzy	183
Sprawdzanie pól obowiązkowych	185
Sprawdzanie list wyboru	187
Neutralizacja danych wynikowych	190
Weryfikacja poprawności danych wejściowych	192
Zapisywanie wszystkich danych formularza do pliku	193
Wysyłanie danych formularza pocztą elektroniczną	195
Odczytywanie informacji na temat plików wysyłanych na serwer	196
Przenoszenie plików wysłanych na serwer do bezpiecznej lokalizacji	199
Śledzenie postępu wysyłania pliku na serwer	201
6 Zapamiętywanie danych użytkowników — pliki cookie i sesje	207
Pliki cookie	208
Tworzenie pliku cookie	212
Odczytywanie danych z plików cookie	215
Ustawianie (rozsądnej) daty wygasania ważności	215
Ustawianie daty wygasania ważności dla konkretnego klienta	217
Usuwanie pliku cookie	219
Udostępnianie plików cookie różnym domenom	220
Sprawdzanie, czy klient obsługuje pliki cookie	222

Zapisywanie wielu danych w jednym pliku cookie	224
Zapisywanie języka preferowanego przez użytkownika	227
Sesje	229
Gdzie są przechowywane sesje?	230
Jak utrzymuje się stan sesji?	231
Rozpoczęcie sesji	233
Odczytywanie i zapisywanie sesji	234
Zamykanie sesji	235
Zmiana identyfikatora sesji	236
Implementacja własnego mechanizmu zarządzania sesjami	238
Tworzenie zabezpieczonego obszaru za pomocą sesji	243
Tworzenie zabezpieczonego obszaru bez korzystania z sesji	245
7 Korzystanie z plików w systemie plików serwera	249
Otwieranie i zamykanie plików	250
Odczytywanie danych z plików	255
Zapisywanie danych do plików	256
Blokowanie plików	258
Uzyskiwanie dostępu do plików przy użyciu ścieżek względnych	259
Unikanie zagrożeń bezpieczeństwa związanych z dostępem do plików	261
Przetwarzanie plików z danymi w formacie CSV	262
Parsowanie plików INI	267
Odczytywanie informacji o plikach	269
Kopiowanie, przenoszenie i usuwanie plików	272
Przeglądanie zawartości systemu plików	273
Korzystanie ze strumieni PHP	274
Przetwarzanie archiwów Bzip2	277
Zwracanie plików w żądaniu HTTP	280

8 Korzystanie z baz danych MySQL	283
Nawiązywanie połączenia z MySQLi	285
Wysyłanie poleceń języka SQL do serwera MySQL	288
Instrukcje przygotowywane w MySQL	290
Odczytywanie wyników zapytania wykonanego przez MySQL	292
Sprawdzanie ostatnio wstawionego identyfikatora	295
Wykonywanie transakcji	296
9 Korzystanie z innych baz danych	299
Nawiązywanie połączenia ze SQLite	300
Wysyłanie poleceń SQL do bazy SQLite	303
Odczytywanie wyników wykonania zapytania przez SQLite	305
Wykonywanie instrukcji przygotowywanych w SQLite	308
Nawiązywanie połączenia z PostgreSQL	310
Wysyłanie poleceń SQL do bazy PostgreSQL	311
Zmienianie danych w bazie PostgreSQL	313
Odczytywanie wyników zapytania wykonanego w bazie PostgreSQL	314
Nawiązywanie połączenia z bazą danych Oracle	316
Wysyłanie poleceń SQL do bazy danych Oracle	317
Odczytywanie wyników zapytania wykonanego w bazie Oracle	320
Nawiązywanie połączenia z serwerem MSSQL	322
Wysyłanie poleceń SQL do bazy danych MSSQL	324
Odczytywanie wyników zapytania wykonanego w bazie MSSQL	326
Wykonywanie instrukcji przygotowywanych w MSSQL	328
Korzystanie z serwera MSSQL bez systemu Windows	329
Nawiązywanie połączenia z serwerem Firebird	332
Wysyłanie poleceń SQL do bazy Firebird	333

Odczytywanie wyników wykonania zapytania przez Firebird	335
Nawiązywanie połączenia za pośrednictwem PDO	336
Wykonywanie poleceń SQL za pośrednictwem PDO	339
Odczytywanie wyników wykonania zapytania za pośrednictwem PDO	340
10 Korzystanie z języka XML	343
Parsowanie danych XML przy użyciu SAX	345
Parsowanie danych XML przy użyciu XMLReadera	347
Odczytywanie danych XML przy użyciu DOM	349
Zapisywanie danych XML przy użyciu DOM	351
Zapisywanie danych XML przy użyciu XMLWritera	352
Zastosowanie SimpleXML	354
Zastosowanie XPath wraz z SimpleXML	356
Przekształcanie danych XML przy użyciu XSL	356
Weryfikacja poprawności danych XML	358
11 Komunikowanie się z innymi źródłami	361
Łączenie się z serwerami HTTP	361
Łączenie się z serwerami FTP	365
Sprawdzanie, czy serwer wciąż odpowiada	367
Tworzenie usługi sieciowej za pomocą NuSOAP	372
Automatyczne generowanie WSDL za pomocą NuSOAP	374
Korzystanie z usługi sieciowej za pomocą NuSOAP	376
Tworzenie usługi sieciowej przy użyciu rozszerzenia SOAP PHP 5	378
Korzystanie z usługi sieciowej za pomocą rozszerzenia SOAP PHP 5	381
Wykorzystanie technologii Ajax	382
Wymiana danych z serwerem	386
Skorowidz	391

Spis treści

Praca z obiektami (i zagadnienia pokrewne)

Programowanie zorientowane obiektowo (ang. *object-oriented programming* — OOP) to jeden z najczęściej współcześnie stosowanych paradygmatów programowania. W pierwszych latach swego istnienia PHP w ogóle nie był postrzegany jako język zorientowany obiektowo. Dopiero w PHP 4 dodano pewne ograniczone mechanizmy obiektowe i znacznie je wzbogacono w PHP 5.

Niniejsza książka jest leksykonem rozmówek, a programowanie zorientowane obiektowo dość wyraźnie różni się od innych poruszanych tutaj zagadnień. Rozmówki w orientacji obiektowej zwykle są obszerne, dotyczą pewnych bardzo konkretnych przypadków albo wręcz odwrotnie — mają charakter na tyle ogólny, że wręcz można je postrzegać jako wiedzę powszechnie znaną (której poszukuje się stosunkowo rzadko).

Definiowanie klas

Zastanawiałem się nad kilkoma podejściami do tego rozdziału i w końcu zdecydowałem się zaprezentować mieszankę zagadnień zorientowanych obiektowo i im pokrewnych — frazy, które ilustrują przydatność mniej trywialnych mechanizmów obiektowych dostępnych w PHP, frazy, które wyjaśniają pewne podstawowe konstrukcje obiektowe, i jeszcze kilka fraz o innym charakterze. W programowaniu obiektowym najważniejsze jest praktyczne doświadczenie, lecz sądzę, że pomimo tego informacje zawarte w tym rozdziale również od czasu do czasu okazały się pomocne.

Definiowanie klas

```
class MyClass {  
}
```

W tym punkcie przedstawię zwięzły (choć zdecydowanie niepełny) opis najważniejszych mechanizmów obiektowych obsługiwanych przez PHP. Centralnym elementem orientacji obiektowej jest klasa. Klasa może zawierać stałe, właściwości i metody. Z kolei słowa kluczowe określające widoczność, takie jak `public`, `protected` i `private`, wskazują zakres dostępu do danej właściwości lub metody.

Gdy utworzona zostanie instancja klasy (służy do tego słowo kluczowe `new`), słowo kluczowe `$this` stanowi odwołanie do wywołującego obiektu. Jeżeli klasa zawiera metodę o nazwie `__construct()`, zostanie ona wywołana w momencie tworzenia instancji klasy, co widać na listingu 4.1.

Listing 4.1. Implementacja klasy (class.php)

```
<?php
class MyClass {
    private $prop = null;

    public function setProperty($value) {
        $this->prop = $value;
    }

    public function getProperty() {
        return $this->prop;
    }

    public function __construct($value = null) {
        if ($value !== null) {
            $this->prop = $value;
        }
    }
}

$c = new MyClass('abc');
echo $c->getProperty();
?>
```

UWAGA

Dostępne są trzy następujące poziomy widoczności:

- **public** — dostępność z każdego miejsca kodu.
- **protected** — dostępność jedynie z wnętrza tej samej klasy oraz z jej klas potomnych, a także z instancji tej samej klasy i klas potomnych.
- **private** — dostępność jedynie z wnętrza tej samej klasy oraz z instancji tej samej klasy.

Dziedziczenie

Dostęp do właściwości i metod klasy można też uzyskać bez wcześniejszego tworzenia instancji tej klasy — służy do tego kontekst statyczny (i słowo kluczowe `static`). Co oczywiste, dostęp do metod statycznych nie jest możliwy za pośrednictwem `$this`. Odwołanie do bieżącej klasy zapewnia natomiast słowo `self`. Odpowiedni przykład znajduje się na listingu 4.2.

Listing 4.2. Implementacja klasy z metodą i właściwością statyczną (`static.php`)

```
<?php
class MyStaticClass {
    private static $prop = 'abc';

    public static function getProperty() {
        return self::$prop;
    }
}

echo MyStaticClass::getProperty();
?>
```

Dziedziczenie

```
class MyDerivedClass extends MyBaseClass {
}
```

Klasa może dziedziczyć po innej klasie za pomocą słowa kluczowego `extends`. W konsekwencji nowa klasa będzie zawierać (dziedziczyć) wszystkie publiczne i chronione metody klasy bazowej. Oczywiście w klasach potomnych można pokrywać odziedziczone metody.

Dostęp do klasy przodka zapewnia słowo kluczowe `parent`. Jego składnia przypomina odwołanie statyczne, jednak takim nie jest. W wywoływanej w ten sposób metodzie można zatem używać słowa `$this`.

Kod widoczny na listingu 4.3 pokrywa konstruktor w klasie potomnej, ale także wywołuje konstruktor klasy przodka. Warto zwrócić uwagę, że sygnatury obydwóch konstruktorów różnią się od siebie. W kodzie głównym również wywołwana jest metoda klasy bazowej.

Listing 4.3. Dziedziczenie w PHP (extends.php)

```
<?php
class MyBaseClass {
    protected $value1 = null;
    protected $value2 = null;

    protected function __construct($value = null) {
        if ($value !== null) {
            $this->value1 = $value;
        }
    }

    public function getValue1() {
        return $this->value1;
    }
}

class MyDerivedClass extends MyBaseClass {
    protected $value2 = null;

    public function __construct($value1 = null,
        ↪$value2 = null) {
        if ($value1 !== null) {
            parent::__construct($value1);
            if ($value2 !== null) {
                $this->value2 = $value2;
            }
        }
    }
}
```

```
    }  
  }  
  
  public function getValue2() {  
    return $this->value2;  
  }  
}  
  
$c = new MyDerivedClass('abc', 'def');  
echo '1: ', $c->getValue1(), ', 2:',  
     ↪$c->getValue2();  
?>
```

UWAGA

PHP nie obsługuje dziedziczenia wielokrotnego, co oznacza, że dana klasa może dziedziczyć tylko po (dokładnie) jednej innej klasie. Oczywiście klasa B może dziedziczyć po klasie A, a klasa C może dziedziczyć po klasie B, dzięki czemu klasa C będzie dziedziczyć również publiczne i chronione metody klasy A.

Korzystanie z abstrakcyjnych klas i interfejsów

```
abstract class MyAbstractBaseClass {  
}
```

PHP obsługuje dwa dodatkowe mechanizmy dziedziczenia: klasy abstrakcyjne i interfejsy. Obydwie konstrukcje są do siebie podobne, lecz różnice między nimi mają charakter zasadniczy. Celem obydwóch jest wyznaczenie szablonu dla

klasy potomnej; inaczej mówiąc, klasa bazowa (albo klasy bazowe, o czym więcej powiem za chwilę) definiuje, które metody trzeba zaimplementować w nowej klasie.

Gdy używa się klasy abstrakcyjnej, dziedziczenia dokonuje się tak jak zwykle, czyli za pomocą słowa kluczowego `extends`. Nowością jest natomiast fakt, że klasa bazowa jest definiowana jako abstrakcyjna, o czym decyduje słowo `abstract`. Ponadto zbiór metod z klasy bazowej też można oznaczyć jako metody abstrakcyjne, lecz nie zawierają one wówczas żadnego kodu źródłowego:

```
protected abstract function myMethod();
```

W takim przypadku klasa potomna musi bezwzględnie implementować tak zdefiniowane metody, metody te muszą mieć identyczne sygnatury oraz takie same (lub mniej restrykcyjne) ustawienia widoczności. Na przykład jeżeli abstrakcyjna klasa bazowa definiuje abstrakcyjną metodę chronioną, wówczas klasa potomna musi implementować tę metodę jako chronioną lub publiczną. Jeżeli w klasie bazowej wskazywane są typy, w klasie potomnej muszą one być identyczne.

Jeżeli metoda abstrakcyjna zawiera dodatkowe metody, które nie są zadeklarowane jako abstrakcyjne, wówczas dziedziczy się je tak samo jak po zwykłych klasach, o ile będą one chronione lub publiczne. Ilustruje to kod źródłowy przedstawiony na listingu 4.4.

Listing 4.4. Zastosowanie klasy abstrakcyjnej (abstract.php)

```
<?php
abstract class MyAbstractBaseClass {
    protected $value1 = null;
    protected $value2 = null;

    protected function getValue1() {
        return $this->value1;
    }

    protected function getValue2() {
        return $this->value2;
    }

    protected abstract function dumpData();
}

class MyAbstractClass extends
↳MyAbstractBaseClass {
    public function __construct($value1 = null,
↳$value2 = null) {
        if ($value1 !== null) {
            $this->value1 = $value1;
        }
        if ($value2 !== null) {
            $this->value2 = $value2;
        }
    }
}

public function dumpData() {
    echo '1: ', $this->getValue1(), ' 2: ',
↳$this->getValue2();
}
}

$c = new MyAbstractClass('ghi', 'jkl');
$c->dumpData();
?>
```

Podobnie jak zwykle klasy, klasy abstrakcyjne obsługują tylko dziedziczenie po jednej klasie przodka.

„Takie samo, ale inne” jest działanie interfejsów. Interfejs wygląda jak zwykła klasa języka PHP, w której jednak zamiast słowa `class` używa się słowa `interface`. Interfejs w ogóle nie zawiera żadnej implementacji, a jedynie sygnatury funkcji. Wszystkie te funkcje muszą mieć charakter publiczny.

Definiowana klasa „dziedziczy” po interfejsie. W tym kontekście mówi się jednak zawsze, że klasa *implementuje* interfejs. Podkreśla to słowo kluczowe `implements`, które zastępuje słowo `extends`.

Pozostałe reguły są bardzo podobne jak w przypadku klas abstrakcyjnych. Wszystkie metody interfejsu muszą zostać zaimplementowane, a sygnatury funkcji muszą pozostać identyczne (włącznie z poziomem widoczności, który i tak zawsze jest publiczny). Dodatkową korzyścią jest to, że klasa może jednocześnie implementować więcej niż jeden interfejs, o ile tylko w interfejsach tych nie powtarza się nazwa żadnej z metod.

Kod widoczny na listingu 4.5 tworzy dwa interfejsy, a następnie implementuje je w jednej klasie.

Listing 4.5. Zastosowanie interfejsu (interface.php)

```
<?php
interface MyInterface1 {
    public function getValue1();
    public function getValue2();
}

interface MyInterface2 {
```

Korzystanie z abstrakcyjnych klas i interfejsów

```
    public function dumpData();
}

class MyInterfaceClass implements MyInterfacel,
↳MyInterface2 {
    protected $value1 = null;
    protected $value2 = null;

    public function __construct($value1 = null,
↳$value2 = null) {
        if ($value1 !== null) {
            $this->value1 = $value1;
            if ($value2 !== null) {
                $this->value2 = $value2;
            }
        }
    }

    public function getValue1() {
        return $this->value1;
    }

    public function getValue2() {
        return $this->value2;
    }

    public function dumpData() {
        echo '1: ', $this->getValue1(), ', 2:',
↳$this->getValue2();
    }
}

$c = new MyInterfaceClass('mno', 'pqr');
$c->dumpData();
?>
```

Zapobieganie dziedziczeniu i pokrywaniu

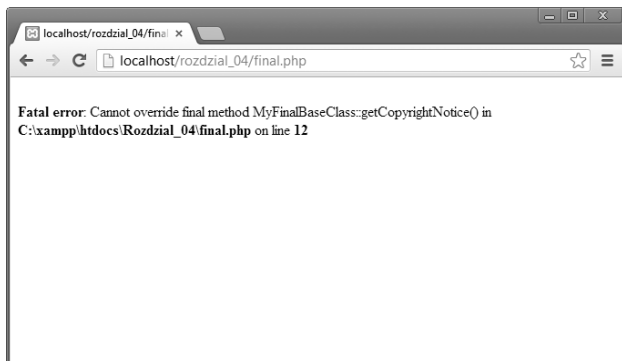
```
final class MyFinalBaseClass {}  
public final function myMethod() {}
```

Istnieją klasy, po których nie powinno się już dziedziczyć (zwłaszcza jeśli implementują one najbardziej istotne funkcje i nie chcemy, aby inni programiści te kluczowe funkcje w jakikolwiek sposób nadpisywali). Aby temu zapobiec, w PHP udostępniono słowo kluczowe `final`. Finalną można uczynić klasę (w takim przypadku nie będzie już można w ogóle po niej dziedziczyć); można też zadeklarować metodę jako finalną (wówczas nie będzie można jej już więcej pokrywać). Dlatego kod przedstawiony na listingu 4.6 rzuci wyjątek, widoczny na rysunku 4.1.

Listing 4.6. Zapobieganie nadpisywaniu dzięki słowu kluczowemu `final` (final.php)

```
<?php  
class MyFinalBaseClass {  
    public final function getCopyrightNotice() {  
        return '&copy; by Autor Oryginalny';  
    }  
}  
  
class MyFinalClass extends MyFinalBaseClass {  
    function getCopyrightNotice() {  
        return '&copy; ja!';  
    }  
}  
?>
```

Automatyczne ładowanie



Rysunek 4.1. Metod finalnych nie można pokrywać

Oczywiście jeśli samemu pracujesz nad implementacją klasy albo metody, deklarowanie ich jako finalnych nie wydaje się konieczne. Trzeba jednak pamiętać, że któregoś dnia zespół programistyczny może się powiększyć albo możesz zdecydować się na refaktoring własnego kodu i umieszczenie go w bibliotece udostępnianej innym programistom.

Automatyczne ładowanie

```
spl_autoload_register('myAutoloadFunction');
```

Aby możliwe było użycie klasy, musi ona zostać wcześniej zdefiniowana; w przeciwnym razie PHP rzuci błąd. Gdy jednak w momencie użycia klasy ta jeszcze nie jest dostępna, można skorzystać z drugiej szansy. Daje ją funkcja `spl_autoload_register()`, która rejestruje funkcję przeznaczoną do wywołania w sytuacji, gdy kod PHP podejmie

próbę użycia klasy dotąd niezarejestrowanej. Po wywołaniu funkcji rejestrującej kod źródłowy, który usiłował się odwołać do niezdefiniowanej klasy, zostanie wykonany ponownie. Jeżeli w tym momencie klasa będzie już dostępna, wykonanie kodu będzie przebiegać dalej w normalny sposób. W przeciwnym razie zwrócony zostanie standardowy błąd.

Standardową praktyką jest używanie z góry zdefiniowanego schematu nazewnictwa plików, które zawierają kod źródłowy klas, i umieszczanie tych plików w zdefiniowanym katalogu. W takiej konwencji klasa `MyClass` będzie się znajdować w pliku `/classes/MyClass.php`, a funkcja automatycznego ładowania widoczna na listingu 4.7 załaduje klasę na żądanie.

Listing 4.7. Automatyczne ładowanie klas (autoload.php)

```
<?php
function myAutoloadFunction($classname) {
    if (preg_match('/^[a-zA-Z_\x7f-\xff]
        ↪[a-zA-Z0-9_\x7f-\xff]*$/', $classname)) {
        // sprawdzenie poprawności nazwy klasy
        require_once "/ścieżka/do/classes/
            ↪$classname.class.php";
    }
}

spl_autoload_register('myAutoloadFunction');
?>
```

Tak więc wywołanie w postaci:

```
$c = new UnknownClass();
```

spowoduje, że wywołana zostanie funkcja `myAutoloadFunction()`, której parametrem `$classname` będzie nazwa klasy `UnknownClass`.

Automatyczne ładowanie

Jeżeli funkcja `spl_autoload_register()` zostanie wywołana więcej niż jeden raz, zarejestrowane zostaną po kolei wszystkie funkcje automatycznego ładowania. PHP będzie więc wywoływać każdą z nich aż do momentu, gdy zdefiniowana zostanie pożądana klasa — albo gdy wszystkie te funkcje zostaną wywołane.

UWAGA

Funkcja `spl_autoload_register()` została po raz pierwszy udostępniona w PHP 5.1.2 jako element biblioteki SPL (ang. *Standard PHP Library*). We wcześniejszych wersjach języka trzeba było implementować funkcję o nazwie `__autoload()` i w niej umieszczać kod odpowiedzialny za automatyczne ładowanie. W którymś momencie rozwoju PHP obsługa funkcji `__autoload()` może zostać całkowicie wyeliminowana, dlatego najlepiej jest już teraz używać tylko funkcji `spl_autoload_register()`.

UWAGA

PHP umożliwia wywoływanie metod i właściwości klas, które nie istnieją:

- Jeżeli wywoływana jest metoda klasy i metoda ta nie istnieje, wówczas wywołana zostaje metoda `__call()` tej klasy, o ile ta metoda istnieje.
- Jeżeli odczytywana jest właściwość klasy i właściwość tej klasy nie istnieje, wówczas wykonywane są metody `__get()` i `__set()` tej klasy (przeznaczone odpowiednio do odczytywania i ustawiania właściwości), o ile metody takie istnieją.

Klonowanie obiektów

```
$b = clone $a;
```

Przyjmijmy, że zmiennej `$a` przypisano instancję klasy. Instrukcja przypisania `$b = $a` nie stworzy kopii instancji `$a`, lecz zmiennej `$b` zostanie przypisane odwołanie do `$a`. Wciąż istnieć będzie zatem tylko jedna instancja klasy.

Alternatywą jest udostępniane przez PHP słowo kluczowe `clone`, które służy do tworzenia kopii klasy:

```
$b = clone $a;
```

Czasami jednak nie chcemy tworzyć *dokładnych* kopii klas. Wyobraźmy sobie na przykład, że każda instancja klasy zawiera globalnie unikatowy identyfikator GUID. Stworzony klon instancji klasy mimo wszystko powinien posiadać identyfikator GUID, który będzie niepowtarzalny.

PHP pozwala na wstrzykiwanie kodu do obiektu, który został sklonowany. Jeżeli klasa posiada metodę o nazwie `__clone()`, zostanie ona wywołana na sklonowanej instancji klasy od razu po zakończeniu procesu klonowania.

Listing 4.8. Klonowanie obiektów (i zmiana ich zawartości) (clone.php)

```
<?php
class MyCloneableClass {
    private $guid = null;

    public function __construct() {
        $this->guid = uniqid();
    }
}
```

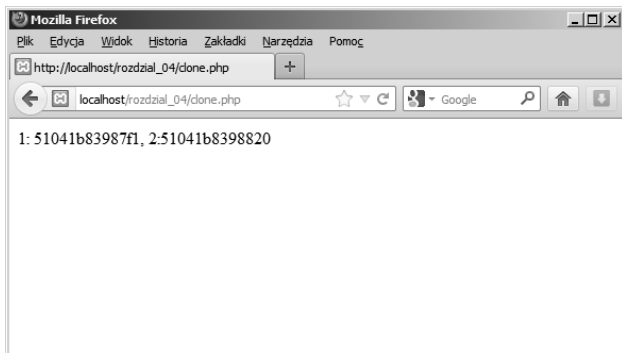
Klonowanie obiektów

```
public function __clone() {
    $this->guid = uniqid();
}

public function getGuid() {
    return $this->guid;
}
}

$c1 = new MyCloneableClass();
$c2 = clone $c1;
echo '1: ', $c1->getGuid(), ', 2:', $c2->getGuid();
?>
```

Wynik wykonania kodu z listingu 4.8 będzie podobny do widocznego na rysunku 4.2: dwie instancje tej samej klasy posiadają różne identyfikatory GUID.



Rysunek 4.2. Dwie instancje tej samej klasy posiadają różne identyfikatory GUID

Serializacja i deserializacja obiektów

```
public function __sleep() {
    $this->db->close();
    return array('guid');
}
public function __wakeup() {
    $this->connectDb();
}
```

Gdy trzeba przechowywać obiekty w celu ich wykorzystania w późniejszym czasie, trzeba te obiekty przekształcić do takiej postaci, która pozwoli na ich zapisanie na przykład w bazie danych. PHP zapewnia co najmniej kilka mechanizmów, dzięki którym instancję obiektu można przekształcić do postaci ciągu znaków (a potem stworzyć na powrót obiekt w pierwotnej postaci). Najczęściej używanym rozwiązaniem jest wykorzystanie funkcji `serialize()` i `deserialize()`, z których będę korzystał również w kolejnych rozdziałach tej książki.

Z perspektywy programowania zorientowanego obiektowo ciekawostką jest to, że proces serializacji i deserializacji można „przechwycić”. Przyjmijmy na przykład, że klasa zawiera właściwość, która nie powinna (albo nie może) być serializowana, bo jest, dajmy na to, uchwytem bazy danych. Najlepiej byłoby usunąć tę właściwość przed rozpoczęciem serializacji i odtworzyć w przypadku deserializacji.

Serializacja i deserializacja obiektów

Twórcy języka PHP zdecydowali się ułatwić tego typu zadanie i zaimplementowali dwie „magiczne” metody. Jeżeli dana klasa będzie te metody implementować, zostaną one wywołane w odpowiednich okolicznościach. Metodami tymi są:

- **__sleep()** — musi zwracać tablicę ze wszystkimi właściwościami podlegającymi serializacji; może też czyścić serializowany obiekt.
- **__wakeup()** — wywołuje się ją po wykonaniu serializacji; można w niej na nowo definiować właściwości.

Kod z listingu 4.9 stanowi typową implementację mechanizmów zamykania i resetowania połączenia z bazą danych. Po wykonaniu serializacji i deserializacji identyfikator GUID pozostaje taki sam, natomiast połączenie z bazą danych jest ustanawiane na nowo.

Listing 4.9. Serializacja i deserializacja obiektów (serialize.php)

```
<?php
class MySerializableClass {
    public $guid = null;
    public $db = null;

    public function __construct() {
        $this->guid = uniqid();
        $this->connectDb();
    }

    private function connectDb() {
        $this->db = new MySQLi('serwer', 'uzytkownik',
            ↪ 'haslo', 'bazadanych');
    }

    public function __sleep() {
        $this->db->close();
        return array('guid');
    }
}
```

```

    }

    public function __wakeup() {
        $this->connectDb();
    }
}

$c1 = new MySerializableClass();
echo 'Przed: ', $c1->guid;
$s = serialize($c1);
$c2 = unserialize($s);
echo '; po: ', $c2->guid;
?>

```

UWAGA

Przed deserializacją obiektu trzeba się upewnić, że dana klasa istnieje. W przeciwnym wypadku PHP użyje do stworzenia wyniku deserializacji „specjalnej” klasy o nazwie `__PHP_Incomplete_Class_Name`.

WSKAZÓWKA

PHP udostępnia całe mnóstwo „magicznych” metod i funkcji. Pełna lista takich metod i funkcji znajduje się w podręczniku online, dostępnym pod adresem <http://php.net/oop5.magic>.

Implementowanie singletonów

```

if (self::$instance == null) {
    self::$instance = new self;
}
return self::$instance;

```

Implementowanie singletonów

Równie często stosowanym, co krytykowanym wzorcem projektowym jest wzorzec singleton. Implementacja singletonu nie jest prosta, dlatego poświęciłem mu oddzielny punkt.

Głównym celem singletonu jest to, aby niezależnie od tego, ile razy tworzy się instancję danej klasy, zawsze powstawał jeden obiekt. Takie rozwiązanie jest użyteczne na przykład w sytuacji, gdy tworzy się połączenie z bazą danych: często wystarczy w całym kodzie źródłowym używać tylko jednego połączenia z bazą.

Standardowa implementacja singletonu w PHP opiera się na kilku mechanizmach programowania zorientowanego obiektowo, których celem jest zapobieganie tworzeniu różnych instancji jednej klasy. Najczęściej konstruktor klasy staje się niedostępny przez zadeklarowanie go jako konstruktora prywatnego:

```
private function __construct() {  
}
```

Jednym ze sposobów tworzenia dwóch instancji klasy jest stworzenie jednej instancji i następnie jej sklonowanie. Można temu zapobiec przez zadeklarowanie magicznej metody `__clone()` również jako metody prywatnej:

```
private function __clone() {  
}
```

W takiej sytuacji nie można już stworzyć więcej niż jednej instancji tej samej klasy. Nie można nawet stworzyć *jednej* jej instancji. Dlatego trzeba zaimplementować metodę, która

stworzy instancję klasy. W kodzie klasy można wywołać jej konstruktor, ponieważ jest on zadeklarowany jako `private`. Instancja klasy będzie wówczas przechowywana we właściwości prywatnej i statycznej. Metoda odpowiedzialna za stworzenie instancji będzie zatem najpierw sprawdzać, czy instancja klasy jest już przypisana właściwości statycznej. Jeżeli tak, metoda zwróci tę instancję. W przeciwnym razie kod stworzy nową, tylko jedną instancję klasy:

```
static private $instance = null;

static public function getInstance() {
    if (self::$instance == null) {
        self::$instance = new self;
    }
    return self::$instance;
}
```

Kod z listingu 4.10 zawiera pełną implementację klasy singletonu.

Listing 4.10. Implementacja singletonu (singleton.php)

```
<?php
class PHPSingleton{
    static private $instance = null;

    static public function getInstance() {
        if (self::$instance == null) {
            self::$instance = new self;
        }
        return self::$instance;
    }

    private function __construct() {
    }
    private function __clone() {
```

Implementowanie singletonów

```
}  
}  
?>
```

UWAGA

Jak wspomniałem już wcześniej, wzorce singletonów mają pewne wady. Przede wszystkim w przypadku korzystania z singletonów kod staje się trudniejszy do testowania, ponieważ klasa singletonu musi być klasą globalną i nadaje stan aplikacji. Jak zwykle w takich przypadkach, ostateczne rozwiązanie zależy od charakteru rozwiązywanego problemu.

Korzystanie z przestrzeni nazw

Jednym z ważniejszych nowych mechanizmów dodanych do PHP 5.3 jest obsługa przestrzeni nazw. Przestrzeń nazw to narzędzie, które służy do grupowania powiązanych ze sobą funkcji z jednoczesnym uniknięciem konfliktów nazw. Wyobraźmy sobie, że implementowana jest funkcja o ogólnej nazwie `showInfo()`. Kolejny programista, który pracuje na tym samym projekcie, implementuje inną funkcję, ale nadaje jej tę samą nazwę. Często w takiej sytuacji nazwę funkcji poprzedza się jakimś przedrostkiem, aby uczynić ją unikatową. W efekcie powstają nieczytelne, zbyt długie identyfikatory, na przykład `Projekt_Moduł_Podmoduł_↪showInfo()`. Z pomocą w takich sytuacjach przychodzą przestrzenie nazw. Przy użyciu przestrzeni nazw wskazuje się kontekst, w danej przestrzeni nazw funkcja `showInfo()` jest unikatowa, a inni programiści wykorzystują inne przestrzenie nazw. W konsekwencji dzięki przestrzeniom nazw łatwiej jest enkapsulować konkretne rozwiązania.

Słowem kluczowym dla przestrzeni nazw jest słowo namespace. Słowo to musi być pierwszym poleceniem obecnym na stronie PHP (mogą je poprzedzać co najwyżej komentarze, które są opcjonalne); nie może przed nim występować nawet żaden znacznik języka HTML. Aby stworzyć strukturę przestrzeni nazw, można zdefiniować hierarchię, w której w roli separatora będzie się używać znaku odwrotnego ukośnika:

```
namespace Projekt\Moduł\Podmoduł;
```

Tak zdefiniowana przestrzeń nazw będzie prawidłowo funkcjonować w całym pliku. Jeżeli w pliku tym umieści się implementację metody `showInfo()` i plik ten dołączy się do innego skryptu PHP, wówczas metodę `showInfo()` będzie można wykonać następującą instrukcją:

```
Projekt\Moduł\Podmoduł\showInfo();
```

Aby w jednym pliku zdefiniować więcej niż jedną przestrzeń nazw, można wykonać kilka instrukcji `namespace`. Zalecanym podejściem jest jednak w takiej sytuacji zastosowanie nawiasów klamrowych:

```
namespace Przestrzeńnazw1 {  
    //...  
}  
  
namespace Przestrzeńnazw2 {  
    //...  
}
```

W PHP istnieje kilka sposobów używania przestrzeni nazw. Można oczywiście podawać w pełni kwalifikowaną nazwę przestrzeni nazw, jak w przykładzie. Dozwolone jest także stosowanie nazw względnych. W przypadku przestrzeni nazw o charakterze lokalnym bardziej dogodnym podejściem jest stosowanie aliasów. Składnia `use <przestrzeńnazw>` albo `use <przestrzeńnazw> as <alias>` może znacząco skrócić czas implementacji kodu i ułatwić

Stosowanie cech

korzystanie z przestrzeni nazw. Poniższy fragment kodu ilustruje sposób, w jaki importuje się przestrzeń nazw:

```
import Projekt\Moduł\Podmoduł;  
Podmoduł\showInfo();
```

Jeżeli użyje się aliasu, kod może przybrać następującą postać:

```
import Projekt\Moduł\Podmoduł as Mod;  
Mod\showInfo();
```

Na pierwszy rzut oka skorzystanie z wbudowanych klas PHP wewnątrz przestrzeni nazw może budzić wątpliwości, ponieważ interpreter będzie szukał wskazanych klas w bieżącej przestrzeni nazw. Wystarczy jednak poprzedzić nazwę klasy wbudowanej znakiem odwrotnego ukośnika, który pozwala PHP poszukać klasy na liście klas globalnych:

```
namespace Projekt\Moduł\Podmoduł;  
$c = new \SoapClient('plik.wsdl');
```

Na temat przestrzeni nazw można by mówić jeszcze długo, jednak ze względu na formułę tej książki skupiłem się jedynie na sposobie ich działania (i to też w dużym skrócie). Zdecydowanie więcej informacji na temat przestrzeni nazw można znaleźć w podręczniku online, dostępnym pod adresem <http://php.net/namespace>.

Stosowanie cech

```
class MyTraitClass {  
    use MyTrait1, MyTrait2 {  
        MyTrait1::showTime as now;  
        MyTrait2::showCopyright insteadof MyTrait1;  
    }  
}
```

Jednym z nowych rozwiązań dodanych do PHP 5.4 jest obsługa tak zwanych cech (ang. *trait*), które stanowią narzędzie ponownego wykorzystania istniejącego kodu źródłowego. Cecha jest kolekcją funkcji — swego rodzaju klasą, która nie posiada instancji. Aby skorzystać z nowego mechanizmu, trzeba najpierw stworzyć cechę, wykorzystując do tego celu słowo kluczowe `trait`.

```
trait MyTrait1 {
    public function showTime() {
        $date = new DateTime('now');
        echo $date->format('H:i:s');
    }
}
```

Następnie w kodzie klasy można takie zdefiniowane cechy łączyć przy użyciu słowa kluczowego `use`. W ten sposób metody stają się dostępne wewnątrz klasy:

```
class MyTraitClass {
    use MyTrait1;
}

$c = new MyTraitClass();
$c->showTime();
```

W jednej klasie można korzystać z więcej niż jednej cechy. Trzeba jednak wówczas zwrócić szczególną uwagę na to, czy cechy te nie implementują tych samych metod. W takiej sytuacji zwrócony zostanie komunikat błędu, podobny do komunikatu rzucanego wtedy, gdy implementuje się interfejsy z metodami o takich samych nazwach. W przeciwieństwie do interfejsów cechy pozwalają jednak zapobiegać tego typu błędom. Gdy ładuje się cechę, potencjalne konflikty nazw można rozwiązywać przy użyciu operatora

Stosowanie cech

instanceof. Załóżmy na przykład, że istnieją dwie cechy: MyTrait1 i MyTrait2 i obydwie te cechy zawierają definicję metody showTime(). Składnia przedstawiona poniżej zapewni, że żaden błąd się nie pojawi oraz że w kodzie użyta zostanie implementacja metody showTime() pochodząca z cechy MyTrait1 zamiast MyTrait2:

```
class MyTraitClass {
    use MyTrait1, MyTrait2{
        MyTrait1::showTime insteadof MyTrait2;
    }
}
```

Innym rozwiązaniem jest zastosowanie aliasu, użycie słowa kluczowego as, a nawet skorzystanie w cesze z metod abstrakcyjnych. W kodzie widocznym na listingu 4.11 implementowane są dwie cechy, wykorzystywany jest operator insteadof oraz alias, a wynik działania tego kodu znajduje się na rysunku 4.3.

Listing 4.11. Użycie dwóch cech (trait.php)

```
<?php
trait MyTrait1 {
    public function showTime() {
        $date = new DateTime('now');
        echo $date->format('H:i:s');
    }

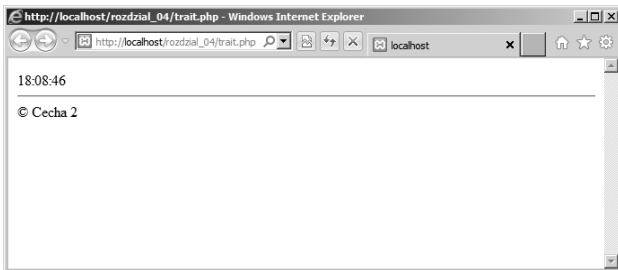
    public function showCopyright() {
        echo '&copy; Cecha 1';
    }
}

trait MyTrait2 {
    public function showCopyright() {
```

```
        echo '&copy; Cecha 2';
    }
}

class MyTraitClass {
    use MyTrait1, MyTrait2 {
        MyTrait1::showTime as now;
        MyTrait2::showCopyright insteadof MyTrait1;
    }
}

$c = new MyTraitClass();
$c->now();
echo '<hr />';
$c->showCopyright();
?>
```



Rysunek 4.3. W przykładowym kodzie wykonana została metoda drugiej cechy

Stosowanie cech

Skorowidz

A

- adres IP, 79
- AES, Advanced Encryption Standard, 31
- Ajax, Asynchronous JavaScript and XML, 382
- algorytm
 - AES, 31
 - Bzip2, 277
 - DES, 31
 - MD5, 32
 - Quicksort, 81
 - SHA1, 32
- algorytmy szyfrowania, 31
- alias, 153
- ASCII, 36
- atak
 - przemierzania katalogów, 261
 - XSS, 26, 160
- attribut
 - action, 161
 - checked, 176, 179
 - enctype, 197
 - method, 197

- name, 166
- value, 175

- Auth, 248
- automatyczna lokalizacja dat, 103
- automatyczne
 - generowanie WSDL, 379
 - ładowanie klas, 143

B

- baza danych
 - Firebird, 332
 - Microsoft SQL, 323
 - MySQL, 283
 - Oracle, 317
 - PostgreSQL, 311
 - SQLite, 300
- biblioteka
 - FreeTDS, 330
 - jQuery, 388, 389
 - libmysql, 286
 - lib-mysqli.dll, 286
 - libxslt, 357

biblioteka

- PEAR, 205, 282, 342, 360, 389
- PHP, 248
- php_mysqli.dll, 286
- php_sqlite.dll, 301
- php_xsl.dll, 357
- SPL, 94, 144
- ZZIPLib, 276

bibliotki ICU, 84

blok CDATA, 346

blokowanie

- plików, 258, 300
- plików cookie, 221

C

cecha, trait, 155

CGI, Common Gateway

Interface, 247

ciąg znaków

- neutralizacja, 26
- podciąg, 35
- porównywanie, 22
- przekształcanie
 - w datę, 115
 - w tablicę, 69
- skanowanie, 41
- suma kontrolna, 32
- szyfrowanie, 30
- weryfikacja loginów, 23
- wielokrotne łączenie, 39
- wyszukiwanie, 44
- zastępowanie, 56

D

dane

- BLOB, 309
- dynamiczne, 190
- JSON, 389
- z formularza, 161, 169

data wygaśnięcia cookie, 216

Date, 129

Date_Holidays, 129

DB_DataObject, 342

debugowanie, 43

DES, Data Encryption

Standard, 30

dokument RFC, 366

DOM, Document Object

Model, 349, 351

dopisywanie danych do pliku, 257

dostęp do

- biblioteki ZZIPLib, 276
- MySQL, 284
- obiektów danych PHP, 337
- pliku, 200, 253, 259, 261
- PostgreSQL, 311
- tablic, 162

DSN, Data Source Name, 339

dyrektywa

- include_path, 262
- open_basedir, 262
- session.auto_start, 234
- session.save_path, 230
- session.user_trans_sid, 232
- upload_tmp_dir, 200

dziedziczenie, 134

E

- element
 - <form>, 161
 - <option>, 179
 - blockquote, 66
- elementy formatujące, 28

F

- File, 282
- File_Find, 282
- File_SearchReplace, 282
- filtrowanie tablic, 89
- filtry weryfikacji, 192
- format
 - Bzip2, 279
 - CSV, 70, 262
 - GMT, 128
 - INI, 267
 - XML, 346
- formatowanie daty, 108
- formularz HTML, 159
 - odczytywanie danych, 162
 - weryfikacja danych, 185
 - wybieranie daty, 121
 - wysyłanie danych, 161
 - wysyłanie pliku, 203
 - zapisywanie danych, 166
 - zapisywanie do pliku, 193
- FTP, File Transfer Protocol, 250, 362
- funkcja
 - bzclose(), 279
 - addslashes(), 326
 - array_filter(), 90
 - array_map(), 86
 - array_rand(), 91
 - array_walk(), 88
 - arsort(), 74
 - asort(), 74
 - basename(), 260
 - bzopen(), 279
 - bzwrite(), 279
 - checkdate(), 111
 - chr(), 38
 - compareChar(), 84
 - copy(), 273
 - count(), 62
 - crypt(), 30, 34
 - date(), 98, 101
 - date_sunrise(), 117
 - date_sunset(), 117
 - DateTime
 - createFromFormat(), 111
 - deserialize(), 147
 - dirname(), 260
 - each(), 62
 - echo(), 43
 - explode(), 70
 - fgetcsv(), 263
 - fgets(), 256
 - file(), 255
 - file_exists(), 254
 - file_get_bzip2_contents(), 279
 - file_get_contents(), 255, 276, 365
 - file_put_contents(), 258, 276
 - filter_input(), 191
 - filter_var(), 192
 - flock(), 258
 - fopen(), 250, 252
 - fputcsv(), 265

funkcja

- fssockopen(), 363
- ftp_chdir(), 366
- ftp_close(), 366
- ftp_connect(), 366
- ftp_get(), 366
- ftp_login(), 366
- getCookieData(), 168
- getFormDataGET(), 172
- getFormDataPOST(), 172
- gettimeofday(), 118
- gmdate(), 128
- gmmktime(), 128
- gmstrftime(), 128
- header(), 223
- htmlentities(), 27
- htmlspecialchars(), 26, 85, 169, 190
- ibase_fetch_assoc(), 336
- ibase_prepare(), 334
- ibase_query(), 334
- implode(), 71, 255
- is_array(), 67
- is_numeric(), 53
- isset(), 186
- json_encode(), 387
- ksort(), 73
- list(), 68
- md5_file(), 35
- mktime(), 109
- move_uploaded_file(), 200
- mssql_fetch_assoc(), 331
- mssql_fetch_object(), 331
- mssql_fetch_row(), 331
- mssql_select_db(), 331
- mysqli_close(), 287
- mysqli_commit(), 297
- mysqli_connect(), 286
- mysqli_fetch_assoc(), 293
- mysqli_fetch_object(), 293
- mysqli_fetch_row(), 293
- mysqli_insert_id(), 296
- mysqli_prepare(), 291
- mysqli_query(), 288
- mysqli_rollback(), 297
- mysqli_select_db(), 287
- mysqli_stmt_execute(), 291
- natcasesort(), 80
- natsort(), 80
- nl2br(), 29
- oci_bind_by_name(), 320
- oci_connect(), 317
- oci_execute(), 319
- oci_fetch_all(), 321
- oci_fetch_assoc(), 321
- oci_fetch_object(), 321
- oci_fetch_row(), 321
- oci_parse(), 319
- ord(), 38
- parse_ini_file(), 268
- pg_connect(), 311
- pg_escape_string(), 312
- pg_fetch_all(), 316
- pg_fetch_assoc(), 316
- pg_fetch_object(), 316
- pg_fetch_row(), 316
- pg_insert(), 314
- pg_query(), 312
- pg_update(), 314
- preg_match(), 48
- preg_match_all(), 48
- preg_replace(), 56
- print(), 43
- print_r(), 65
- printf(), 39
- rename(), 273

- funkcja
- sajax_show_javascript(), 384
 - serialize(), 147
 - session_destroy(), 236
 - session_id(), 237
 - session_regenerate_id(), 237
 - session_set_save_handler(), 239
 - session_start(), 234
 - setcookie(), 212, 222
 - setlocale(), 105
 - setrawcookie(), 214
 - sha1_file(), 35
 - showInfo(), 152
 - shuffle(), 92
 - simplexml_load_file(), 355
 - sort(), 72
 - spl_autoload_register(), 142
 - sprintf(), 41
 - sqlite_escape_string(), 304
 - sqlite_exec(), 303, 306
 - sqlite_fetch_all(), 306
 - sqlite_fetch_array(), 306
 - sqlite_fetch_object(), 306
 - sqlite_open(), 301
 - sqlite_query(), 306
 - sqlsrv_connect(), 324
 - sqlsrv_fetch_array(), 327
 - sqlsrv_fetch_object(), 328
 - sqlsrv_prepare(), 329
 - sqlsrv_query(), 325
 - scanf(), 42
 - str_replace(), 57
 - strcasecmp(), 23
 - strcmp(), 23
 - strftime(), 103–105, 109
 - strip_tags(), 28
 - strlen(), 38
 - strnatcmp(), 83
 - strpos(), 44
 - strrpos(), 44
 - strtolower(), 24
 - strtotime(), 115
 - strtoupper(), 24
 - substr(), 35
 - time(), 125, 216
 - trim(), 51
 - unlink(), 273
 - urlencode(), 27, 214
 - usort(), 81
 - var_dump(), 43
 - var_export(), 43
 - vprintf(), 41, 42
 - vsprintf(), 41
 - xml_parser_create(), 345
- funkcje
- anonimowe, 87
 - obsługi daty i czasu, 97
 - sprawdzające typ danych, 52
 - uchwyty, 346
 - wbudowane ftp, 367
 - zwracające informacje o plikach, 270
- ## G
- generowanie
- liczb, 54
 - WSDL, 379
- gniazdko, 363
- graficzne przyciski wysyłania danych, 184

H

hash, 32
 hasło, 24, 33
 HTML_QuickForm2, 205
 HTTP, Hypertext Transfer Protocol, 249, 362
 HTTP_Header, 389
 HTTP_Request2, 389
 HTTP_Server, 389
 HTTP_Session, 248
 HTTP_Upload, 205
 HTTPS, 364

I

identyfikator
 GUID, 145
 sesji, 231
 IIS, Internet Information Services, 253
 implementacja
 klasy, 133
 singletonu, 149
 informacje o plikach, 269
 instrukcje MySQL, 290
 interfejs, 136
 ArrayAccess, 94
 ArrayObject, 95
 Countable, 94
 Iterable, 95
 interwał czasu, 126

J

język
 SQL, 283
 XML, 343, 356

JSON, JavaScript Object Notation, 386

K

klasa, 132
 __PHP_Incomplete_Class
 __Name, 149
 Ctrx_SOAP_AutoDiscover, 380
 DateInterval, 126, 127
 DateTime, 102, 126
 dir, 273
 klasy
 abstrakcyjne, 136, 138
 bazowe, 135
 pośredniczące, proxy class, 377
 potomne, 134
 klient JSON, 388
 klonowanie obiektów, 145
 kod
 HTML, 27
 JavaScript, 26, 218
 kodowanie znaków, 160
 kompresja pliku, 279
 komunikat o błędzie, 373
 konstruktor prywatny, 150
 kopiowanie plików, 272

L

lista HTML, 347
 listy
 wielokrotnego wyboru, 180, 182
 wyboru, 178

logowanie, 25
losowanie elementów, 91

Ł

łączenie się z serwerami
FTP, 365
HTTP, 361

M

MDB_QueryTool, 342
MDB2, 342
mechanizm zarządzania
sesjami, 240
metoda
 __call(), 144
 __clone(), 145, 150
 __construct(), 132
 __get(), 144
 __set(), 144
 __sleep(), 148
 __wakeup(), 148
add(), 126
addFunction(), 379
bindValue(), 309
createElement(), 352
diff(), 126
exec(), 304
fetch(), 341
fetchAll(), 341
file_put_contents(), 365
GET, 25, 159
getElementById(), 350
getElementsByTagName(),
 350
getProxy(), 377

getTime(), 218
handle(), 379
importStylesheet(), 357
loadXML(), 350
open(), 302
POST, 25
query(), 340
read(), 348
save(), 352
setClass(), 380
showtime(), 156
sub(), 126
transformToDoc(), 357
validate(), 359
xpath(), 356

metody
 abstrakcyjne, 137
 finalne, 141
model DOM, 349
modyfikator
 +, 251
 b, 251
 t, 251

N

narzędzia Ajax, 383
narzędzie Webservice Helper,
 380
nawiasy kwadratowe, 181
nazwa użytkownika, 24
Net_FTP, 389
Net_Socket, 389
neutralizacja
 ciągów znaków, 26
 danych wynikowych, 190
 znaków specjalnych, 325

niejawna konwersja typów, 22
NuSOAP, 372–376, 380

O

obliczanie
 daty względnej, 112
 różnicy między datami, 124

obsługa
 domen TLD, 222
 plików, 249
 plików cookie, 223
 protokołu FTP, 366
 sesji, 230, 234
 XPath, 356

ochrona adresów poczty, 37

odczytywanie
 i zapisywanie sesji, 234
 informacji o pliku, 196
 zawartości katalogu, 273
 zawartości tablic, 61,
 63, 65

odczytywanie danych z
 InterBase/Firebird, 335
 Oracle, 320
 PostgreSQL, 315
 plików cookie, 171, 215
 pliku, 255
 MSSQL, 327–329
 MySQL, 292
 SQLite3, 305

operator
 ==, 22
 ===, 22
 instanceof, 350

opis WSDL, 381

otwieranie
 bazy danych SQLite3, 302
 pliku, 250

P

pakiet
 HTML_QuickForm2, 205
 HTTP_Header, 389
 HTTP_Request2, 389
 HTTP_Server, 389
 HTTP_Upload, 205
 Net_FTP, 389
 Net_Socket, 389

parser
 SAX, 345
 XMLReader, 348

parsowanie
 danych XML, 345, 349,
 354
 plików INI, 267

PDO, PHP Data Object,
 337–341

PECL, PHP Extension
 Community Library, 301

pętla
 for, 62
 foreach, 62, 64
 while, 69, 293

plik
 AddService.wsdl, 380
 check.inc.php, 51
 check.php, 51
 getFormData.inc.php, 181
 getFormData.inc.php, 179
 mutlilingual.php, 229
 nusoap.php, 372

- php.ini, 268
 - quotes.xml, 357
 - README, 366
 - select-multiple.php, 182
 - pliki
 - .htaccess, 246
 - Bzip2, 278
 - cookie, 166, 208–229
 - data wygaśnięcia, 211, 215
 - ustawianie domeny, 220
 - usuwanie, 219
 - zapisywanie danych, 224
 - CSV, 265
 - DTD, 359
 - INI, 267
 - rng, 358
 - WSDL, 377
 - XML, 359
 - xsd, 358
 - XSLT, 357
 - ZIP, 274
 - polo formularza, 165
 - obowiązkowe, 185
 - opcji, 175
 - tekstowe, 173
 - wyboru, 177
 - połączenie
 - INSERT, 296
 - SELECT, 293
 - połączenie z
 - Oracle, 316
 - MySQLi, 285
 - PostgreSQL, 310
 - InterBase/Firebird, 332
 - MSSQL, 322, 324
 - SQLite, 300
 - połączenie za pośrednictwem PDO, 337
 - pomiar wydajności, 119
 - porównywanie elementów tablicy, 72
 - postęp wysyłania pliku, 201
 - poziomy widoczności, 133
 - program
 - pgAdmin, 311
 - phpPgAdmin, 311
 - TAR, 276
 - programowanie zorientowane obiektowo, 131
 - protokół
 - FTP, 362
 - HTTP, 207, 362
 - HTTPS, 364
 - SMTP, 370
 - SOAP, 370
 - UDP, 370
 - XML-RPC, 369
 - przechowywanie danych sesji, 238
 - przechwytywanie błędów, 381
 - przekształcanie typów danych, 53
 - przenoszenie plików, 199
 - przestrzenie nazw, 152
- ## R
- relacyjna baza danych, 284
 - repozytorium
 - PEAR, 129
 - Subversion, 372
 - REST, Representational State, 370

Skorowidz

ręczna lokalizacja dat, 107
 RFC, Request For Comment, 366
 rozpakowywanie pliku, 277, 279
 rozpoczęcie sesji, 233
 rozpoznawanie języka użytkownika, 227
 rozszerzenie
 DOM, 350
 ext/mssql, 323
 ext/sqlite, 301
 ext/sqlite3, 302
 ext/sqldr, 329
 ext_sqlite3, 309
 filter, 191
 ibase, 332
 ieHTTPHeaders, 210
 LiveHTTPHeaders, 210
 NuSOAP, 379
 php_pgsql.dll, 311
 SimpleXML, 354
 SOAP, 378, 381

S

SAX, Simple API for XML, 345
 serializacja obiektów, 147
 serwer
 Apache, 253
 Firebird, 332
 FTP, 365
 HTTP, 362
 Microsoft IIS, 253
 MSSQL, 323, 331
 MySQL, 285

PostgreSQL, 311
 sesje, 229–247
 czyszczenie, 239
 identyfikator, 231
 mechanizm zarządzania, 238
 niszczenie, 239
 odczytywanie, 234
 otwieranie, 233
 zamykanie, 235
 zapisywanie, 234
 zmiana identyfikatora, 236
 SimpleXML, 354, 356
 singleton, 150
 skanowanie ciągów znaków, 42
 skrypt
 login.php, 244
 mod_files.sh, 231
 setcookie-specific.php, 218
 słowo kluczowe
 \$this, 132
 as, 156
 clone, 145
 extends, 134
 final, 141
 global, 163
 implements, 139
 namespace, 153
 new, 132
 parent, 135
 static, 134
 trait, 155
 use, 155
 SOAP, 370, 378

sortowanie
 adresów IP, 79
 dowolnych wartości, 81
 naturalne, 80
 numeryczne, 80
 tablicy ze znakami
 narodowymi, 82
 zagnieżdżonych tablic
 asocjacyjnych, 77
 zawartości tablic, 71
 asocjacyjnych, 73
 zagnieżdżonych, 75
 sól, 31
 spam, 38
 specyfikacja
 HTML, 161
 plików cookie, 209
 SPL, Standard PHP Library,
 144
 sprawdzanie
 danych
 uwierzytelniających, 244
 identyfikatora, 295
 konfiguracji obsługi
 plików cookie, 223, 226
 list wyboru, 187
 list wielokrotnego
 wyboru, 189
 pól obowiązkowych, 185
 stanu serwera, 367
 uwierzytelnienia, 243
 zawartości tablicy
 \$_COOKIE, 223
 stała __FILE__, 260
 stałe, 191
 sterownik mysqlnd, 286
 sterowniki dla serwerów, 337
 Stream_Var, 282

strumienie PHP, 275
 suma kontrolna, 32
 symbole
 formatujące dla
 funkcji date(), 98
 funkcji strftime(), 104
 klasy DateInterval, 127
 zastępcze, 40, 319
 system phpMyAdmin, 284
 szyfrowane hasła, 29

Ś

ścieżka
 pliku cookie, 228
 względna, 259
 śledzenie stanu wysyłania, 202

T

tabela quotes, 284
 tablica
 \$_COOKIE, 171, 215
 \$_FILES, 163, 197
 \$_GET, 163
 \$_POST, 163
 \$_REQUEST, 163
 \$_SESSION, 202, 234
 \$HTTP_GET_VARS, 162
 \$HTTP_POST_VARS,
 162
 \$HTTP_REQUEST_VARS,
 162
 filtrowanie, 89
 losowanie, 92
 odczytywanie zawartości,
 61, 63, 65

tablica

- przekształcanie zawartości, 68
- przekształcenie w ciąg znaków, 70
- przetwarzanie elementów, 85
- sortowanie, 71, 73

tablice

- asocjacyjne, 59, 293
- numeryczne, 59
- superglobalne, 163
- zagnieżdżone, 65

technologia Ajax, 382

transakcje, 296

tryb

- autozatwierdzania, 297
- plikowy, 333
- serwerowy, 333

tryby otwarcia pliku, 252

tworzenie

- danych XML, 352
- klasy, 94
- kodu XML, 351
- pliku cookie, 212
- pliku relaxNG, 359
- sesji, 230
- tablicy, 59
- usługi sieciowej, 372, 378
- znacznika czasu, 113

typy

- blokad, 258
- danych, 309

U

- usługi sieciowe, Web Services, 343, 369, 377, 381
 - SOAP, 372
 - z WSDL, 374, 380
- usuwanie
 - danych sesji, 235
 - plików, 272
 - plików cookie, 219
 - znaczników HTML, 26
- uwierzytelnianie, 243, 246, 247

W

- wartości predefiniowane, 169, 173
- wartość mieszająca, 32
- weryfikowanie
 - adresów poczty, 54
 - danych, 185
 - danych XML, 358
 - daty, 111
 - hasła, 29, 31, 33
 - pól formularza, 51
 - typów danych, 52
- węzeł, 350
- WSDL, Web Services
 - Description Language, 371, 374, 380
- wskaźnik do zbioru wyników, 292
- wstrzyknięcie kodu, 145
 - JavaScript, 26
 - SQL, 289

wykonywanie
 instrukcji SQLite, 308
 polecenia SQL za
 pośrednictwem PDO,
 339
 transakcji, 296
 wyłączanie tablicy, 163
 wymiana danych, 386
 wyniki zapytania, 340
 wyodrębnianie podciagu, 35
 wyrażenia regularne PCRE,
 45, 48, 56
 wysyłanie
 danych, 184, 195
 metoda GET, 208
 metoda POST, 208
 nagłówek HTTP, 208
 formularza, 164
 plików, 161, 202
 pliku do klienta, 280
 poleceń SQL, 288
 MSSQL, 325
 InterBase/Firebird, 333
 Oracle, 318
 PostgreSQL, 312
 SQLite, 303
 SQLite3, 304
 wyszukiwanie znaczników, 49
 wyświetlanie
 elementów tablicy, 89
 zawartości zagnieżdżonych
 tablic, 67

X

XML, extensible Markup
 Language, 343
 XML_Beautifier, 360

XML_DTD, 360
 XML_Parser2, 360
 XML_Serializer, 360
 XML_Util, 360
 XMLReader, 347, 353
 XML-RPC, Remote
 Procedure Call, 369
 XMLWriter, 353
 XPath, 356
 XSL, 356
 XSLT, XSL Transformation,
 357
 XSS, Cross-Site Scripting,
 26, 160

Z

zabezpieczanie stron PHP, 246
 zagnieżdżanie tablic, 60
 zamykanie pliku, 253
 zapisywanie do pliku, 256
 zarządzanie
 serwerem MySQL, 285
 sesjami, 233, 238
 zastosowanie
 interfejsu, 139
 klasy abstrakcyjnej, 138
 zaznaczanie
 pozycji na listach, 178
 pół opcji, 176
 pół wyboru, 177
 zmienianie
 identyfikatora sesji, 236
 danych w bazie, 314
 zmienna
 \$_GET[], 160
 \$_POST[], 160
 znacznik czasu, 109, 113

Skorowidz

znak

@, 252
apostrofu, 170
cudzysłowu, 260, 265
dwukropka, 309
odwrotnego ukośnika,
153, 260
przecinka, 265
średnika, 268
zapytania, 291

znaki

nowego wiersza, 29
specjalne, 46, 190

Ż

żądanie HTTP, 280

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>



GOTOWE DO UŻYCIA
FRAGMENTY KODU

PHP i MySQL

ROZMÓWKI

PHP i MySQL to niewątpliwie najpopularniejszy duet do tworzenia dynamicznych witryn WWW i aplikacji internetowych. Jego popularność jest wynikiem połączenia ogromnych możliwości, przystępności oraz bezpłatnego dostępu do obu narzędzi. PHP i MySQL sprawdzą się w wielu zastosowaniach — od prostych skryptów, przez systemy do zarządzania treścią na stronach WWW, po sklepy internetowe oraz zaawansowane serwisy.

W tej książce znajdziesz dziesiątki przykładowych fragmentów kodu, które możesz od ręki wykorzystać do swoich potrzeb. Operacje na ciągach znaków i tablicach, formatowanie dat czy przetwarzanie formularzy to tylko niektóre z poruszanych tu zagadnień. Ponadto Twoją ciekawość powinien wzbudzić rozdział poświęcony programowaniu obiektowemu, korzystaniu z sesji oraz nawigowaniu po systemie plików na serwerze. W każdym z poruszanych tematów znajdziesz odwołania do repozytorium PEAR, którego zawartość pomoże Ci wiele problemów rozwiązać znacznie lepiej i szybciej. Tę książkę każdy programista PHP powinien mieć zawsze pod ręką!

Dzięki niej:

- błyskawicznie rozwiążesz typowe problemy z PHP i MySQL
- będziesz mieć pod ręką przydatne i gotowe do użycia fragmenty kodu
- sprawdzisz, jak zawartość repozytorium PEAR może Ci pomóc

DEVELOPER'S
LIBRARY

helion.pl
księgarnia
internetowa



Helion

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
<http://helion.pl>

Sprawdź najnowsze promocje:
● <http://helion.pl/promocje>
Książki najchętniej czytane:
● <http://helion.pl/bestsellery>
Zamów informacje o nowościach:
● <http://helion.pl/nowosci>

Cena: 39,90 zł

ISBN 978-83-246-7023-9



9 788324 670239

Nr katalogowy: 14212

Księgarnia internetowa:

<http://helion.pl>

Zamówienia telefoniczne:

0 801 339900

0 601 339900

Informatyka w najlepszym wydaniu