*Błyskawicznie opanuj nowy język programowania!*

# JavaScript

## *dla programistów PHP*

HELION

O'REILLY®

*Stoyan Stefanov*

Tytuł oryginału: JavaScript for PHP Developers

Tłumaczenie: Rafał Jońca

ISBN: 978-83-246-8217-1

© 2013 Helion S.A.

Authorized Polish translation of the English edition of JavaScript for PHP Developers, ISBN 9781449320195 © 2013 Stoyan Stefanov.

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to sell the same.

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 jakiejkolwiek 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.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą 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)

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

Printed in Poland.

• Kup książkę
• Poleć książkę
• Oceń książkę

• Księgarnia internetowa
• Lubię to! » Nasza społeczność

# Spis treści

**Podziękowania** ..... **11**

**Wstęp** ..... **13**

**1. Wprowadzenie** ..... **15**
Zakres niniejszej książki 17
Język 18
Nauka środowiska 18
Przeglądarki 19
JavaScriptCore 20
Node.js i Rhino 21
Dłuższe przykłady 22
Zaczynamy 22

**2. Składnia języka JavaScript** ..... **23**
Zmienne 23
Jakie jest zadanie znaku $ w JavaScript? 24
Wartości 25
Introspekcja typeof 26
Wartości null i undefined 27
Tablice 28
Tablice asocjacyjne 29
Instrukcje warunkowe 30
Operator trójargumentowy 30
Ścisła kontrola typu 31
Konstrukcja switch 33
Konstrukcja try-catch 33
Pętle while i for 35
Pętle for-in 36



Kup książkę Poleć książkę

Inne operatory 37
Operator in 37
Łączenie fragmentów tekstów 38
Rzutowanie typów 39
Operator void 39
Operator przecinka 40

**3. Funkcje ..... 41**
Parametry domyślne 41
Dowolna liczba argumentów 42
Sztuczka z arguments.length 43
Zwracanie wyniku funkcji 44
Funkcje są obiektami 44
Nieco inna składnia 45
Zakres widoczności zmiennych 46
Przenoszenie na początek 48
Przenoszenie na początek funkcji 49
Domknięcia 50
Domknięcia w języku PHP 50
Łańcuch zakresu widoczności zmiennych 52
Łańcuch zakresów w konsoli WebKit 54
Zachowanie zakresu 57
Przechowywane są referencje, a nie wartości 58
Domknięcia w pętli 59
Ćwiczenie — pętla z onclick 61
Funkcje natychmiastowe 62
Inicjalizacja 64
Prywatność 64
Przekazywanie i zwracanie funkcji 65
Wywołania zwrotne nie są tekstami ..... 67

**4. Programowanie obiektowe ..... 69**
Konstruktory i klasy 69
Zwracanie obiektów 70
Dodatkowe informacje na temat this 72
Wymuszenie wywołania konstruktora 73



Kup książkę Poleć książkę

Prototypy 74
Literał obiektu 75
Dostęp do właściwości 75
Mylące kropki 76
Metody w literałach obiektów 77
Rozbudowane tablice 78
Właściwości własne 79
Właściwość __proto__ 80
Stosować this lub prototype? 81
Dziedziczenie 82
Dziedziczenie wykorzystujące prototypy 82
Dziedziczenie przez kopiowanie właściwości 83
Funkcja kreująca 84
„Klasyczne" extend() 85
Pożyczanie metod 86
Wnioski 87

**5. Wbudowane interfejsy programistyczne.....89**
Obiekt globalny 89
Właściwości globalne 90
Funkcje globalne 91
Liczby 91
Kodowanie adresów URL 92
Konstruktory wbudowane 93
Konstruktor Object 94
Konstruktor Array 98
Konstruktor RegExp 105
Konstruktor Function 107
Konstruktor String 109
Konstruktor Number 113
Konstruktor Boolean 114
Obiekt Math 114
Konstruktor Error 115
Konstruktor Date 116
Podsumowanie konstruktorów 118



Kup książkę Poleć książkę

**6. ECMAScript 5 ..... 119**
Tryb ścisły 119
Atrybuty właściwości 120
Nowe API dotyczące obiektów 121
Metoda Object.create() 121
Metoda Object.getOwnPropertyDescriptor() 123
Metody Object.defineProperty() i Object.defineProperties() 123
Ograniczenie zmian obiektów 123
Pętle alternatywne 125
Metoda Object.getPrototypeOf() 125
Nowe API dotyczące tablic 126
Metoda Array.isArray() 126
Metody indexOf() i lastIndexOf() 126
Przechodzenie przez elementy tablicy 127
Filtrowanie 128
Sprawdzanie zawartości tablicy 128
Odwzorowanie i redukcja 128
Przycinanie tekstu 129
Nowości w obiekcie Date 130
Metoda Function.prototype.bind() 130
Obiekt JSON 131
Shimy 131

**7. Wzorce w języku JavaScript ..... 133**
Właściwości prywatne 133
Metody prywatne 134
Uwidacznianie obiektów prywatnych 135
Zwracanie tablic prywatnych 136
Kopiowanie głębokie przy użyciu JSON 137
Wzorzec udostępniania 139
Stałe 140
Przestrzenie nazw 142
Moduły 144
Moduły CommonJS 144
Definicja modułu CommonJS 145
Użycie modułu CommonJS 145
Utworzenie modułu uniwersalnego 147



Kup książkę Poleć książkę

AMD 148
Wzorce projektowe 149
Singleton 149
Fabryka 151
Dekorator 152
Dokumentacja i testy 155
Podręcznik użytkownika 155
Dokumentacja kodu 155
Testy jednostkowe 156
Narzędzie JSLint 157
Uruchomienie lint dla kodu zawartego w książce 157

**Skorowidz ..... 159**



Kup książkę Poleć książkę

Kup książkę Poleć książkę

ROZDZIAŁ 6.

# ECMAScript 5

Do tej pory książka opisywała tak naprawdę standard ECMAScript 3, ponieważ to najbardziej rozpowszechniona wersja języka. Najnowszym standardem jest jednak ECMAScript 5.1 (wersja 4. okazała się ślepą uliczką i została zarzucona). Większość nowoczesnych przeglądarek obsługuje ES5. Warto wiedzieć, co przyniosła nowa wersja standardu, nawet jeśli cały czas trzeba pisać kod działający również w starszych przeglądarkach.

Trzema głównymi obszarami aktualizacji są:

- wprowadzenie trybu ścisłego,
- zmiany w tworzeniu obiektów i atrybuty właściwości,
- wprowadzenie nowych API.

## Tryb ścisły

Tryb ścisły nie jest zgodny wstecz, więc trzeba jawnie wskazać chęć jego użycia. Raz na funkcję lub też raz na cały program (plik) można wskazać chęć przejścia w tryb ścisły, używając poniższego wiersza:

```
"use strict";
```

Ponieważ to zwykła instrukcja zawierająca tekst, starsze przeglądarki zignorują taki wpis i nie zgłoszą błędu. Interpreter obsługujący tryb ścisły zobaczy wpis z prośbą o użycie trybu ścisłego, więc wyłączy niektóre funkcje i konstrukcje języka JavaScript, które przez ostatnie lata okazały się przynosić więcej złego niż pożytku. W niniejszej książce pominąłem wszystkie niezalecane praktyki, więc nie musisz unikać żadnej z prezentowanych tu konstrukcji.



Kup książkę Poleć książkę

Oto przykłady funkcjonalności, które w trybie ścisłym spowodują zgłoszenie błędu:

- użycie instrukcji `with`,
- użycie niezadeklarowanych zmiennych,
- użycie `arguments.callee` lub `arguments.caller`,
- próba przypisania czegokolwiek do właściwości tylko do odczytu (np. `window.Infinity = 0;`),
- próba usunięcia niekonfigurowalnych właściwości (wkrótce wyjaśnię, co oznacza **konfigurowalność** właściwości),
- literały obiektów ze zduplikowanymi właściwościami,
- parametry funkcji ze zduplikowanymi nazwami (ponieważ, co zadziwiające, w ES3 zapis `function (a, a, a) {}` jest uznawany za prawidłowy).

Poniżej znajduje się krótki przykład ilustrujący różnicę w działaniu między trybem ścisłym i zwykłym:

```
// Tryb zwykły.
var obj = (function () {
   return {a: 1, a: 2};
})();
obj; // {a: 2}

// Tryb ścisły.
var obj = (function () {
   "use strict";
   return {a: 1, a: 2};
})(); // SyntaxError: zduplikowana właściwość.
```

# Atrybuty właściwości

Drugim obszarem zmian w ES5 jest wprowadzenie atrybutów właściwości. Tak naprawdę atrybuty te istniały w języku od dawna, ale nie były bezpośrednio dostępne dla programisty (nie można ich było odczytać lub zmienić).

Właściwość ma atrybut `value` (zawierający wartość) oraz trzy inne atrybuty typu logicznego, które określają, czy właściwość jest:

- wyliczalna,
- zapisywalna,
- konfigurowalna.



Kup książkę Poleć książkę

Atrybuty właściwości definiuje się przy użyciu specjalnych obiektów nazywanych **deskryptorami**. Oto przykład:

```
var stealth_descriptor = {
   value: "nie mogę jej dotknąć",
   enumerable: false,  // Nie pojawi się w pętlach for-in.
   writable: false,    // Nie można zmienić jej wartości.
   configurable: false // Nie można jej usunąć lub zmienić jej atrybutów.
};
```

Właściwości z wyłączoną konfiguracyjnością mogą mieć atrybut `writable` ustawiony tylko na wartość `false`.

Deskryptory właściwości zapewniają dodatkowy poziom kontroli nad zmianami dotyczącymi właściwości. Umożliwiają między innymi utworzenie stałych, których nie można usunąć. Choć wiele elementów nowego API ES5 można z powodzeniem zaimplementować w ES3, deskryptorów właściwości nie można emulować. W ES3 wszystkie tworzone właściwości można dowolnie modyfikować i usuwać.

Dalsze informacja, a także dodatkowe przykłady dotyczące deskryptorów pojawią się w dalszej części rozdziału.

# Nowe API dotyczące obiektów

Nowe API obiektów dotyczy przede wszystkim obsługi atrybutów właściwości i dodatkowo definiuje kilka nowych metod:

- `Object.create()`,
- `Object.defineProperty()`,
- `Object.defineProperties()`,
- `Object.getOwnPropertyDescriptor()`.

## Metoda Object.create()

Metoda służy do utworzenia nowego obiektu, ale jednocześnie potrafi zdefiniować dla niego:

- dziedziczenie,
- właściwości obiektu,
- atrybuty właściwości.



Kup książkę Poleć książkę

Rozważmy następujący fragment kodu:

```
var human = {name: "Jan"};
var programmer = Object.create(
   human,
   {
      secret: stealth_descriptor,
      skill: {value: "Prawdziwy ninja kodu"}
   }
);
```

Obiekt `programmer` dziedziczy po obiekcie `human` przy użyciu `__proto__`, więc nie zawiera właściwości `name` sam w sobie, ale otrzymuje ją dzięki łańcuchowi prototypów:

```
programmer.name; // "Jan"
programmer.hasOwnProperty('name'); // false
programmer.__proto__.hasOwnProperty('name'); // true
```

Dwie pozostałe właściwości są jego własnymi właściwościami:

```
programmer.hasOwnProperty('secret'); // true
programmer.hasOwnProperty('skill');  // true
```

W przypadku właściwości `secret` zastosowano pełny deskryptor umieszczony w zmiennej `stealth_descriptor`. Ustawieniu uległa wartość oraz trzy atrybuty.

Właściwość `skill` definiuje jedynie wartość. Oznacza to, że wszystkie atrybuty zostaną ustawione na swoje wartości domyślne, czyli `false`.

Ustawienie właściwości bez użycia deskryptora spowoduje, że wszystkie atrybuty przyjmą wartość `true`, podobnie jak w ES3:

```
programmer.likes = ['pizza', 'piwo', 'kawa'];
```

Metoda `Object.create()` przypomina użycie funkcji `begetObject()` (opisywanej w rozdziale 4. w podrozdziale „Dziedziczenie”) wraz z dodaną obsługą deskryptorów.

Wcześniej w książce wskazałem, że „puste” obiekty (na przykład `var o = {};`) nie są tak naprawdę puste, ponieważ dziedziczą po `Object.prototype` metody takie jak `toString()`. W ES5 można utworzyć naprawdę pusty obiekt, używając poniższego kodu:

```
var o = Object.create(null); // Nie dziedzicz po niczym.
typeof o.toString;           // "undefined"
```



Kup książkę Poleć książkę

## Metoda Object.getOwnPropertyDescriptor()

Metoda `Object.getOwnPropertyDescriptor()` umożliwia sprawdzenie zawartości deskryptorów:

```
Object.getOwnPropertyDescriptor(programmer, 'secret').configurable; // false
Object.getOwnPropertyDescriptor(programmer, 'likes').configurable;  // true
```

## Metody Object.defineProperty() i Object.defineProperties()

Metody `Object.defineProperty()` i `Object.defineProperties()` umożliwiają zdefiniowanie właściwości wraz z deskryptorem już po utworzeniu obiektu:

```
Object.defineProperty(programmer, 'hello', stealth_descriptor);

Object.defineProperties(programmer, {
   goodbye: stealth_descriptor,
   bye: stealth_descriptor
});
```

## Ograniczenie zmian obiektów

W ES3 można bez przeszkód zmienić wszystkie obiekty (poza kilkoma wyjątkami dotyczącymi obiektów wbudowanych). Nie zawsze jest to dobre rozwiązanie, ponieważ użytkownicy obiektów mogą je zmienić do tego stopnia, że przestaną one prawidłowo funkcjonować. W ES5 możliwe jest ograniczenie dostępu do obiektów. Niektóre właściwości można przełączyć w tryb tylko do odczytu, ustawiając atrybut `writable` na wartość `false`.

Możliwe jest również ograniczenie dostępu do obiektu jako całości:

- wyłączenie możliwości jego rozszerzania (czyli nie będzie możliwe dodanie nowych właściwości);
- „zapieczętowanie” obiektu, czyli poza wyłączeniem możliwości rozszerzania również zamiana wszystkich istniejących właściwości na niekonfigurowalne (więc nie będzie się ich dało usunąć);
- „zamrożenie” obiektu, czyli „zapieczętowanie” połączone z dodatkowym ustawieniem wszystkich właściwości jako tylko do odczytu.

Rozważmy następujący standardowy obiekt:

```
var pizza = {
   tomatoes: true,
   cheese: true
};
```



Kup książkę Poleć książkę

Zawsze można do niego dodać nowe właściwości:

```
pizza.pepperoni = 'dużo';
```

Można zmieniać lub usuwać właściwości, ponieważ wszystkie są konfigurowalne i zapisywalne:

```
pizza.cheese = 'mozzarella';
delete pizza.pepperoni; // true
```

Po wyłączeniu możliwości rozszerzania nie uda się dodać nowych właściwości. Metoda `Object.isExtensible()` zwraca wartość `true`, jeśli obiekt można rozszerzać. Metoda `Object.preventExtensions()` zabezpiecza obiekt przed dalszym rozszerzaniem:

```
Object.isExtensible(pizza);       // true
Object.preventExtensions(pizza); // Zwraca obiekt 'pizza'.
Object.isExtensible(pizza);       // false

pizza.broccoli = 'och, nie'; // Błąd, nie można dodać nowych właściwości.
typeof pizza.broccoli;        // "undefined"
```

Wyłączenie usuwania umożliwia metoda `seal()`:

```
Object.isSealed(pizza); // false
Object.seal(pizza);      // Zwraca obiekt 'pizza'.
Object.isSealed(pizza); // true

delete pizza.cheese; // Błąd, nie można usunąć.
pizza.cheese;         // "mozzarella"
```

Nadal jednak możliwa jest zmiana wartości istniejących właściwości:

```
pizza.cheese = 'ricotta';
```

Po wywołaniu metody `freeze()` wszystkie właściwości staną się niezapisywalne:

```
Object.isFrozen(pizza); // false
Object.freeze(pizza);    // Zwraca obiekt 'pizza'.
Object.isFrozen(pizza); // true

pizza.cheese = "gorgonzola"; // Błąd, 'cheese' jest właściwością tylko do odczytu.
pizza.cheese;  // "ricotta"
```

Podsumujmy:

- Metoda `freeze()` wykonuje te same zadania co metoda `seal()` oraz dodatkowo ustawia atrybut `writable` wszystkich właściwości na wartość `false`.
- Metoda `seal()` wykonuje te same zadania co `preventExtensions()` oraz dodatkowo ustawia atrybut `configurable` wszystkich właściwości na wartość `false`.



Kup książkę Poleć książkę

- Metoda `preventExtensions()` nie ustawia żadnych atrybutów, ale po jej wywołaniu do obiektu nie można dodać żadnych nowych właściwości.
- Żadnej z powyższych operacji nie można cofnąć (innymi słowy, nie ma metod `unfreeze()`, `defrost()` lub `allowExtensions()`).

## Pętle alternatywne

W ES3, by otrzymać listę wszystkich właściwości (czyli wynik podobny do funkcji `array_keys()` z PHP), trzeba było skorzystać z pętli `for-in`. W ES5 dostępne są dwie dodatkowe metody: `Object.keys()` i `Object.getOwnProperty↪Names()`. Metoda `keys()` zwraca wszystkie wyliczane właściwości (czyli takie, które pojawiłyby się w pętli `for-in`), natomiast `getOwnPropertyNames()` zwraca wszystkie własne właściwości niezależnie od statusu ich wyliczania.

Obie metody działają w zasadzie identycznie, jeśli nie używa się deskryptorów do zmiany atrybutów `enumerable`:

```
var pizza = {tomatoes: true, cheese: true};
Object.keys(pizza);                 // ["tomatoes", "cheese"]
Object.getOwnPropertyNames(pizza); // ["tomatoes", "cheese"]
```

Obie metody zwracają tylko właściwości własne, więc nie biorą pod uwagę właściwości dziedziczonych po prototypie. Utwórzmy obiekt `pizza_v20`, który dziedziczy po `pizza`, i dodajmy do niego nowe właściwości:

```
var pizza_v20 = Object.create(pizza, {
   salami: {value: "lots", enumerable: true},
   sauce: {value: "secret"}
});
```

Metoda `keys()` nie zwróci `sauce`, ponieważ nie jest to właściwość wyliczana, ale metoda `getOwnPropertyNames()` zwróci wszystkie właściwości:

```
Object.keys(pizza_v20);                 // ["salami"]
Object.getOwnPropertyNames(pizza_v20); // ["sauce", "salami"]
```

Zauważ, że wyniki nie zawierają właściwości obiektu `pizza`, bo stanowią element dziedziczenia:

```
pizza_v20.cheese;              // true
pizza_v20.__proto__ === pizza; // true
```

## Metoda Object.getPrototypeOf()

Metoda `Object.getPrototypeOf()` to już ostatnia z nowości konstruktora `Object()` w ES5. W ES3 inspekcja możliwa była tylko przy użyciu metody `isPrototypeOf()`,



Kup książkę Poleć książkę

więc trzeba było zgadnąć obiekt nadrzędny, by sprawdzić, czy to faktycznie on jest prototypem.

W ES5 można zapytać bezpośrednio: „Kto jest twoim prototypem?”:

```
Object.getPrototypeOf(pizza_v20) === pizza; // true
```

W zasadzie metoda ta zwraca taki sam wynik jak właściwość `__proto__`, ale warto pamiętać, że `__proto__` nie jest standardem (i nie jest dostępne w przeglądarce IE). ES5 przyznaje, że istnieje potrzeba właściwości `__proto__` w wersji przynajmniej do odczytu, więc pojawiła się metoda `getPrototypeOf()`. Wersja z możliwością zapisu nadal jest źródłem gorących debat.

# Nowe API dotyczące tablic

ES5 wprowadza również kilka przydatnych metod do `Array.prototype`, a także jedną metodę do konstruktora `Array`.

## Metoda Array.isArray()

Metoda `Array.isArray()` to wygodny (bez uciekania się do dodatkowych sztuczek) sposób na stwierdzenie, czy obiekt jest tablicą, czy zwykłym obiektem (pamiętaj, że tablice **są** obiektami).

Jeśli chcesz utworzyć tak zwany **shim** (uzupełnienie) metody i korzystać z niego również w ES3, skorzystaj ze sztuczki `Object.prototype.toString()` w sposób pokazany we wcześniejszej części książki:

```
if (!Array.isArray) {
   Array.isArray = function (candidate) {
      return Object.prototype.toString.call(candidate) === '[object Array]';
   };
}

Array.isArray([]); // true
```

Metoda jest równoważna funkcji `is_array()` z PHP.

## Metody indexOf() i lastIndexOf()

Dwie nowe metody `Array.prototype.indexOf()` i `Array.prototype.lastIndexOf()` oferują nowe możliwości związane z przeszukiwaniem zawartości tablic:

Metoda `Array.prototype.indexOf()` zwraca indeks pierwszego wystąpienia elementu:



Kup książkę Poleć książkę

```
var a = ['raz', 'i', 'dwa', 'i', 'trzy', 'i', 'cztery'];
a.indexOf('trzy'); // 4
```

Przypomina w działaniu funkcję `array_search()` z PHP.

Gdy metoda `Array.prototype.indexOf()` zwraca pierwsze wystąpienie, metoda `Array.prototype.lastIndexOf()` zwraca ostatnie:

```
var a = ['raz', 'i', 'dwa', 'i', 'trzy', 'i', 'cztery'];
a.indexOf("i");     // 1
a.lastIndexOf("i"); // 5
```

Wyszukiwanie korzysta ze ścisłego porównywania wartości (czyli uwzględnia również typ wartości):

```
[1, 2, 100, "100"].indexOf("100"); // 3
[1, 2, 100, "100"].indexOf(100);   // 2
```

Metoda `indexOf()` przyjmuje dodatkowo indeks pozycji startowej, a metoda `lastIndexOf()` indeks pozycji końcowej:

```
var arr = [100, 1, 2, 100];

arr.indexOf(100);     // 0
arr.indexOf(100, 2);  // 3

arr.lastIndexOf(100);    // 3
arr.lastIndexOf(100, 2); // 0
```

## Przechodzenie przez elementy tablicy

W ES5 pojawiła się metoda `Array.prototype.forEach()`, która umożliwia przejście w pętli przez wszystkie elementy tablicy bez potrzeby stosowania instrukcji pętli. Przyjmuje funkcję wywołania zwrotnego, która potrafi wykonać konkretną operację na elemencie tablicy lub też całej tablicy.

Oto bardzo prosty przykład, wyświetlający wszystkie argumenty przekazane do funkcji wywołania zwrotnego:

```
["a", "b", "c"].forEach(function () {
   console.log(arguments);
});
```

W konsoli pojawi się następujący wynik:

```
["a", 0, Array[3]]
["b", 1, Array[3]]
["c", 2, Array[3]]
```

Jak łatwo się domyślić, argumentami są:

```
[aktualny element, jego indeks, cała tablica]
```



Kup książkę Poleć książkę

## Filtrowanie

ES5 wprowadza metodę `Array.prototype.filter()`, która przypomina w działaniu funkcję `array_filter` z PHP i umożliwia wykonanie funkcji zwrotnej dla każdego elementu tablicy. Jeśli wywołanie zwrotne zwróci wartość `true`, element trafia do nowej tablicy, która zostanie zwrócona po sprawdzeniu wszystkich elementów:

```
function testVowels(char) {
   return (/[aeiou]/i).test(char);
}

var input = ["a", "b", "c", "d", "e"];
var output = input.filter(testVowels);

output.join(', '); // "a, e"
```

## Sprawdzanie zawartości tablicy

Dwie nowe metody, `every()` i `some()`, umożliwiają analizę zawartości tablicy i zwrócenie wartości logicznej, jeśli elementy tablicy spełniły test określony w wywołaniu zwrotnym lub go nie spełniły.

Przypuśćmy, że chcemy sprawdzić, czy tablica zawiera liczby parzyste. Funkcja testu ma postać:

```
function isEven(num) {
   return num % 2 === 0;
}
```

A sam test wygląda następująco:

```
// Czy *wszystkie* liczby są parzyste?
[1, 2, 4].every(isEven); // false

// Czy niektóre (przynajmniej jedna) liczby są parzyste?
[1, 2, 4].some(isEven); // true
```

## Odwzorowanie i redukcja

W ES5 pojawiły się metody przypominające w działaniu funkcje `array_map()` i `array_reduce()` z PHP: `Array.prototype.map()` i `Array.prototype.reduce()`, a także metoda dodatkowa `Array.prototype.reduceRight()`.

Metoda `map()` zwraca nową tablicę, której elementy zostały zmienione przez przekazaną funkcję wywołania zwrotnego.



Kup książkę Poleć książkę

Oto funkcja, która zmienia znaki na ich odpowiedniki w postaci encji HTML:

```
function entity(char) {
   return "&#" + char.charCodeAt(0) + ";";
}
```

Zastosujmy ją dla wszystkich elementów tablicy:

```
var input = ['a', 'b', 'c'];
var out = input.map(entity);
out; // ["&#97;", "&#98;", "&#99;"]
```

Metoda `Array.prototype.reduce()` przyjmuje tablicę, a następnie konwertuje ją do pojedynczej wartości, używając funkcji wywołania zwrotnego.

Przypuśćmy, że chcemy zsumować wszystkie wartości w tablicy i dodać 100 do wyniku. Zaczynamy więc od 100 i w każdym kolejnym kroku do sumy dodajemy nową wartość:

```
function sum(running_sum, value, index, array) {
   console.log(arguments);
   return running_sum + value;
}

[1, 2, 3].reduce(sum, 100); // 106
```

Po wykonaniu kodu w konsoli pojawią się następujące wpisy:

```
[100, 1, 0, Array[3]]
[101, 2, 1, Array[3]]
[103, 3, 2, Array[3]]
```

Metoda `Array.prototype.reduceRight()` działa w ten sam sposób, ale analizuje elementy tablicy od prawej do lewej, czyli zaczyna od ostatniego elementu i przesuwa się w stronę pierwszego.

W tym przykładzie wynik końcowy będzie identyczny, ale w konsoli pojawią się inne wartości:

```
[1, 2, 3].reduceRight(sum, 100); // 106
```

W konsoli pojawią się wpisy:

```
[100, 3, 2, Array[3]]
[103, 2, 1, Array[3]]
[105, 1, 0, Array[3]]
```

# Przycinanie tekstu

ES5 wprowadziło metodę `String.prototype.trim()`, przypominającą w działaniu metodę `trim()` z PHP:

```
" witaj ".trim(); // "witaj"
```



Kup książkę Poleć książkę

To jedyna metoda dodana do obsługi tekstów w ES5.

Niektóre środowiska oferują również metody `trimLeft()` i `trim↪Right()`, ale nie stanowią one części standardu.

# Nowości w obiekcie Date

Do obsługi dat w języku JavaScript dodano trzy nowe metody.

Metoda `Date.now()` to równoważnik `(new Date()).getTime()` lub `+new Date()`. Zwraca aktualny czas w postaci znacznika czasowego:

```
Date.now(); // 1327943271496
Date.now() === +new Date(); // true
```

Metoda `Date.prototype.toISOString()` to kolejna z metod odpowiedzialnych za konwersję daty do formatu tekstowego:

```
var date = new Date(2012, 11, 31);
date.toDateString(); // "Mon Dec 31 2012"
date.toISOString();  // "2012-12-31T08:00:00.000Z"
```

Metoda `toISOString()` przydaje się do reprezentacji dat w formacie JSON. Używa jej metoda `JSON.stringify()`, o której za chwilę. W zasadzie istnieje również metoda `Date.prototype.toJSON()`, która zwraca dokładnie taki sam wynik jak metoda `toISOString()`.

```
var today = new Date();
today.toJSON() === today.toISOString(); // true
```

# Metoda Function.prototype.bind()

Ponieważ funkcje są obiektami, które można dowolnie przekazywać, a `this` wewnątrz funkcji zależy od sposobu jej wywołania, czasem ryzykowne jest użycie `this`, ponieważ nie ma się pewności, co tak naprawdę zawiera. Rozważmy następujący przykład:

```
var breakfast = {
   drink: "kawa",
   eat: "bekon",
   my: function () {
      return this.drink + " i " + this.eat;
   }
};
breakfast.my(); // "kawa i bekon"
```



Kup książkę Poleć książkę

Jak na razie żadnych niespodzianek. Przypuśćmy jednak, że utworzymy referencję do metody i ją wywołamy:

```
var morning = breakfast.my;
morning(); // "undefined i undefined"
```

W tym przypadku `this` to obiekt globalny, który nie zawiera właściwości `drink` lub `eat`. Można jednak powiązać metodę `my()` z obiektem `breakfast`, by wszystko działało zgodnie z oczekiwaniami:

```
var morning = breakfast.my.bind(breakfast);
morning(); // "kawa i bekon"
```

# Obiekt JSON

Kończąc dyskusję na temat ECMAScript 5, warto wspomnieć o obiekcie `JSON`. To dodatek do ES5 standaryzujący coś, co zostało zaimplementowane we wszystkich przeglądarkach internetowych, włączając IE8 lub nowszą. JSON to skrót od *JavaScript Object Notation* (notacja obiektowa JavaScript) i stanowi standard wymiany danych używający literałów obiektów i tablic JavaScript do kodowania dowolnych danych.

Obiekt JSON posiada dwie (niezbyt szczęśliwie nazwane) metody — `stringify()` i `parse()` — które odpowiadają działaniu funkcji `json_encode()` i `json_decode()` z PHP:

```
JSON.stringify({hello: "world"});        // '{"hello":"world"}'
JSON.parse('{"hello":"world"}').hello; // "world"
```

W środowisku, które nie posiada wbudowanej obsługi JSON, można użyć biblioteki dostępnej pod adresem *https://github.com/douglascrockford/JSON-js*.

# Shimy

„Shim” lub „polyfill” to nazwa stosowana w przypadku udostępniania nowego API w starszym środowisku. Ponieważ JavaScript umożliwia zmianę wbudowanych obiektów i ich prototypów, emulacja nowego API jest bardzo prosta w realizacji. Oto przykład shimu:

```
if (!Date.now) {
   Date.now = function () {
      return new Date().getTime();
   };
}
```



Kup książkę Poleć książkę

I jeszcze jeden przykład:

```
// źródło:
// https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Global_Objects/String/Trim
if (!String.prototype.trim) {
   String.prototype.trim = function () {
      return this.replace(/^\s+|\s+$/g, '');
   };
}
```

Shimy są najczęściej bardzo krótkie, więc można je wczytać przed wykonaniem pierwszego wiersza właściwego kodu. Dzięki temu w starych przeglądarkach łatwo uzyskać nowe funkcjonalności.

Warto jednak zdawać sobie sprawę z tego, że nie wszystkie funkcjonalności można zrealizować w ten sposób. Przykładem są tryb ścisły i wszystkie metody dotyczące deskryptorów właściwości.

Aby sprawdzić, które funkcje ES5 są obsługiwane w poszczególnych przeglądarkach, skorzystaj ze strony *http://kangax.github.io/es5-compat-table/*.



Kup książkę Poleć książkę

# Skrowidz


<object>, 250

## A

abstract, 44
abstrakcja, 42, 47, 236
abstrakcyjny interfejs, 60
adres IP, 268
adres URL, 252
agregacja, 136, 137, 179, 181, 196
agregat, 136
akcesory, 30
analizator składni, 208
animacje Flash, 252
antywzorce, 291
API, application programming interface, 55, 152
aplety, 28
aplikacje hybrydowe, 23, 241
aplikacje klient-serwer
  przepływ danych, 266
  rozwiązanie własnościowe, 266
  uruchamianie, 270, 276
  XML, 271
aplikacje mobilne, 241
aplikacje typu klient-serwer, 265
arkusze stylów, 216
asocjacja, 136, 137, 180, 181, 197
asocjacje opcjonalne, 186
atrybuty, 29, 35, 192
  inicjowanie, 92
  klasowe, 81
  lokalne, 78
  obiektowe, 79
  prywatne, 38, 90
  publiczne, 38
  statyczne, 90, 95, 105

## B

baza danych
  obiektowa, 112
  relacyjna, 112
  Oracle, 204
  SQL Server, 204
bezpieczeństwo, 118
bezpieczeństwo klientów, 244
biznes elektroniczny, 163
blok try-catch, 76
błędy, 74, 166

## C

catch, 76
COM, 130
CORBA, Common Object Request Broker Architecture, 254, 255
  elementy systemu, 257
CSS, cascading style sheets, 216
cykl życia obiektu, 226
czarna skrzynka, 24

## D

dane, 25
  globalne, 25, 107
  historyczne, 200
  obiektu, 29
  prywatne, 38
  przenośne, 203
DCOM, 258
decyzje projektowe, 136
definicja
  atrybutu XML, 232
  klasy, 33




Kup książkę Poleć książkę

definicja
obiektu, 28
typu dokumentu, 206
XML obiektu, 272
definiowanie wymagań, 121
destruktory
projektowanie, 102
diagram
hierarchii klas, 43
interfejsu, 157
klas, 31, 32, 190
klasy Cabbie, 139, 190
klasy DataBaseReader, 56, 71
klasy Dog, 131
klasy Person, 37
prezentujący kompozycję, 197
reprezentujący asocjację, 198
reprezentujący kompozycję, 196
systemu Shop, 167
wzorca Singleton, 284
diagramy UML, 32, 36, 189
dokument
DTD, 209,210
HTML z arkuszem stylów, 222
HTML z obiektem JSON, 220
opisujący system, 119
SOW, 119
wymagań, 120
XML, 206, 210
dokumentacja, 103
dostęp do relacyjnej bazy danych, 235
drukowanie, 108
drzewo dziedziczenia, 42
drzewo obiektów JavaScript, 249
DTD, 206, 209
dziedziczenie, 47, 48, 129, 131, 133, 194
abstrakcja, 42
decyzje projektowe, 134
definicji, 158
generalizacja, 133
implementacji, 84, 158
pojedyncze, 42
specjalizacja, 133
wielokrotne, 194
wielokrotne, 42, 83, 156
zachowań, 84

## E

edytor formatu JSON, 219
edytor w3schools, 221
egzemplarz, instantiation, 32
egzemplarze klasy, 32
elementy składowe wzorca, 281
Enterprise JavaBeans, 258

## F

faza analizy, 119
Financial products Markup Language, 204
Flash, 252
format JSON, 219
format XML, 271
formatowanie danych, 214
formatowanie elementów, 216
FpML, 204
framework, 150
funkcje, 29

## G

generowanie dokumentacji, 89
graficzny interfejs użytkownika, GUI, 52, 54
gromadzenie wymagań, 120
GUI, graphical user interface, 52

## H

hermetyzacja, 37, 48, 138
dziedziczenie, 139
odpowiedzialność obiektów, 141
polimorfizm, 141
hermetyzacja, encapsulation, 26, 37
hierarchia agregacji, 180
hierarchia klasy, 41, 43
abstrakcyjnej, 154
Car, 138
Dog, 133
Shape, 142
HTML, 206, 214, 245
HTML, Hypertext Markup Language, 205



Kup książkę Poleć książkę

## I

IDE, 120
identyfikacja implementacji, 64
identyfikacja interfejsów publicznych, 100
identyfikacja klas, 120
identyfikacja publicznych interfejsów, 63
identyfikacja użytkownika, 54
IDL, Interface Definition Language, 255
IIOP, Internet Inter-ORB Protocol, 257
implementacja, 32, 38, 52, 54, 55
    interfejsu, 39
    kontraktu, 153
    prywatna, 40
    sklepu, 168
implementacje, 229
infrastruktura programistyczna, 150
inicjalizacja atrybutów, 69
integracja DTD i XML, 210
interfejs
    Externalizable, 227
    IDL, 255
    ISerializable, 113
    klasy, 38
    MailInterface, 289
    Nameable, 161, 167
    programistyczny, 152
    publiczny, 40
    Serializable, 113, 227, 228
interfejsy, 37, 52, 54, 55, 59, 156, 158, 195, 229
    abstrakcyjne, 60
    minimalizacja, 57, 61
    określanie grupy docelowej, 62
    projektowanie, 59
    publiczne, 56, 100
interfejsy klasy, 54
interfejsy publiczne, 63
iteracja, 110

## J

JDBC, 235
język
    C# .NET, 16, 30, 203
    C, 16, 52
    C++, 16
    COBOL, 52
    FORTRAN, 52
    HTML, 205
    Java, 16, 246
    JavaScript, 217, 245
    Objective-C, 17
    SGML, 205
    Smalltalk, 16, 241, 280
    SQL, 235
    UML, 15, 36, 116
    VB .NET, 16
    Visual Basic, 16
    XML, 29, 203, 204, 205
języki
    modelowania, 16
    obiektowe, 207
    programowania, 16
    skryptowe, 242
JSON, JavaScript Object Notation, 218

## K

kaskadowe arkusze stylów, 216
kilka konstruktorów, 70
klasa, 35
    Circle, 155
    implementacja, 54
    interfejs, 54
    Invoice, 262
    JMenuBar, 152
    Person, 238
    Shape, 142
    Shop, 169
    TestShape, 147
    TestShop, 171
klasy, 84
    abstrakcyjne, 44, 153, 158
    atrybuty, 35, 81
    definicja, 33
    definiowanie wymagań, 121
    diagramy UML, 32
    dziedziczenie, 40, 47, 131
    hermetyzacja, 37
    implementacja interfejsu, 39
    iterfejs, 38



Kup książkę Poleć książkę

klasy
kompozycja, 47
konstruktor domyślny, 69
konstruktory, 91
ładowanie dynamiczne, 59
metody dostępowe, 93
metody statyczne, 105
modelowanie, 36, 71
nazwa, 87
ograniczanie zakresu, 107
określanie dostępu, 35
polimorfizm, 44
projektowanie, 26
referencje, 84
rozszerzalność, 105
rozszerzanie interfejsu, 101
ukrywanie danych, 37
ukrywanie implementacji, 101
wytyczne dotyczące projektowania, 99, 115
klasy abstrakcyjne, 145
klasy nadrzędne, 42
klasy o wysokim stopniu sprzężenia, 109
klient, 267, 273
kod JSON, 219
klienta, 267, 273
nieprzenośny, 106, 125
obiektu do serializacji, 266
pośredni, 56
serwera, 269, 274
strukturalny, 122, 124, 154
usług sieciowych, 261
komentarze, 89, 103
kompozycja, 47, 48, 129, 136, 176, 196
diagramy UML, 137
kompozycja, composition, 47
kompozycje
agregacja, 179
asocjacja, 180
unikanie zależności, 182
komunikacja klient-serwer, 266, 272
komunikacja między obiektami, 27, 36
komunikaty, 36
konkatenacja łańcuchów, 82
konkretyzacja, 32
konserwacja kodu, 109
konstruktor, 45
konstruktor domyślny, 69
konstruktory, 67, 91
projektowanie, 73, 102
wywoływanie, 68
zawartość, 68
kontrakt, 152, 260
implementacja, 153
interfejsy, 156
klasy abstrakcyjne, 153
zawieranie, 161
kontrola dostępu dodanych, 26
kontroler, Controller, 281
kontrolki, 250
konwencja nazewnicza, 106
kopiowanie głębokie, 84
kopiowanie obiektów, 84, 107
kopiowanie płytkie, 84

## L, Ł

liczba obiektów, 183
liczność, 199
liczność asocjacji klas, 184
liczność na diagramie, 185
liczność, cardinality, 183
LIFO, last-in, first-out, 46
lokalny adres IP, 268
łączenie łańcuchów, 82

## M

mapery relacyjno-obiektowe, 127
mechanizm usuwania nieużytków, 103
metadane, 33
metoda
get (), 93
set (), 93
metoda getArea(), 44
metoda open(), 58
metody, 29, 36, 192
abstrakcyjne, 44, 155
dostępowe, 30, 93, 232
interfejs, 30
interfejsu publicznego, 95
pobierające, getter, 29, 30, 93



Kup książkę Poleć książkę

prywatne, 95
przeciążanie, 70
przesłanianie, 44
przydzielanie pamięci, 94
statyczne, 105
sygnatura, 70
ustawiające, setter, 29, 30, 93
wirtualne, 145
współdzielone, 78
Microsoft COM, 130
minimalizacja interfejsu, 57, 61
minimalizacja interfejsu publicznego, 100
model, 135
kaskadowy, 116, 117
klas opisujący system, 121
klient-serwer, 236, 237, 243, 265
obiektowy, 116
obiektowy UML, 167
relacyjny, 234
wzorca singleton, 284
Model, 281
modelowanie klas, 36, 71
modelowanie obiektowe, 189
modelowanie systemów świata rzeczywistego, 99
modyfikator dostępu
internal, 194
private, 35, 193
protected, 35, 193
public, 35, 193
mutator, 30
mutatory, 30
MVC, Model-View-Controller, 280
myślenie abstrakcyjne, 59

## N

nadklasy, 42, 73
najlepsze praktyki, 279
namiastki, stub, 110
narzędzia do modelowania, 32
narzędzie do odbierania poczty, 289
nazwa klasy, 87
nazwy, 105
notacja obiektowa, 217
notacja UML, 189

## O

obiekt, 35
obiekt singletonowy, 286
obiektowa baza danych, 24
obiektowe skryptowe języki programowania, 242
obiektowy paradygmat myślenia, 15
obiektowy paradygmat programowania, 52
obiekty, 24, 25, 28, 94, 203, 241, 255
atrybuty, 29, 79
cykl życia, 226
dane, 29
definicja, 28
JavaScript, 248
komunikacja, 36
kopiowanie, 84
liczność, 183
operacje, 84
porównywanie, 84
serializacja, 113
szeregowanie, 113
trwałość, 57, 112
wielokrotne wykorzystanie, 129
zachowania, 29, 63
złożone, 136
obiekty na stronach internetowych, 248
obiekty nadające się do kooperacji, 104
obiekty opakowujące, 122
obiekty osłonowe, 23, 255
obiekty rozproszone, 252
obiekty sieciowe, 28
obiekty trwałe, 225
obiekty w aplikacjach hybrydowych, 241
obiekty w aplikacjach mobilnych, 241
obiekty w aplikacjach typu klient-serwer, 265
obiekty w usługach sieciowych, 241
obsługa błędów, 74, 75
ignorowanie problemu, 74
mieszanie technik, 75
precyzja przechwytywania wyjątków, 77
projektowanie mechanizmu, 103
wyszukiwanie, 75
zgłaszanie wyjątków, 76



Kup książkę Poleć książkę

ODBC, Open Database Connectivity, 236
odpowiedzialność obiektów, 141
odtwarzacze dźwięku, 250
odtwarzacze filmów, 251
ograniczanie liczności, 199
ograniczanie zakresu, 107
ograniczenia środowiska, 63
określanie dostępności, 193
określanie grupy docelowej, 62
określanie zakresu planowanych prac, 119
opakowywanie istniejących klas, 126
opakowywanie kodu strukturalnego, 124
opakowywanie nieprzenośnego kodu, 125
operacje obiektów, 84
oprogramowanie pośredniczące, 254
ORB, Object Request Broker, 256
organizacja W3C, 205
ozaznaczanie zakresu, 78

## P

paradygmat obiektowy, 16
parser, 208
pasek menu, 152
PCDATA, Parsed Character Data, 210
pionowe przenoszenie dane, 204
platforma .NET, 30
plik Invoice.xsd, 260
plik mwsoap.xml, 260
pliki płaskie, 226
pliki wsadowe, 271, 287
plug-and-play, 150
podklasy, 42
podobiekty, 84
podprocedury, 29
polimorfizm, 48, 141, 142
polimorfizm, polymorphism, 44
ponowne kompilowanie kodu, 59
poprawność dokumentów, 206
poprawność dokumentów XML, 212, 214
poprawność dokumentu względem DTD, 208
porównywanie obiektów, 84, 107
powtórzenia, 123
prezentacja danych, 214, 216
procedura inicjująca, 69
procedury, 24, 29
program ORB, 256
program porządkowy, 69
program RestorePerson, 230
program XML Notepad, 212
programowanie obiektowe, 24, 28, 116
programowanie proceduralne, 27, 99
programowanie strukturalne, 25
    powtórzenia, 123
    sekwencyjność, 123
    warunki, 123
projektowanie, 25, 28, 115, 199
    analizy, 119
    definiowanie wymagań, 121
    dokument SOW, 119
    gromadzenie wymagań, 120
    identyfikacja klas, 120
    model kaskadowy, 116
    model klas opisujący system, 121
    prototyp interfejsu użytkownika, 120, 121
    szybkie prototypowanie, 117
    współpraca między klasami, 121
    wytyczne, 115
    zakres planowanych prac, 119
projektowanie destruktorów, 102
projektowanie interfejsów, 59
projektowanie klas, 26
projektowanie konstruktorów, 73, 102
projektowanie mechanizmu obsługi błędów, 103
protokoły, 145
protokół IIOP, 257
protokół SOAP, 257
prototyp interfejsu użytkownika, 120, 121
prywatne metody, 95
przechowywanie danych historycznych, 200
przechwytywanie wyjątku, 76, 77
przeciążanie metody, method overloading, 70



Kup książkę Poleć książkę

przeciążanie operatorów, 82
przekazywanie referencji, 91
przenośność danych, 204
przepływ danych między klientem a serwerem, 266
przesłanianie, overriding, 44
przesyłanie danych, 223
przetwarzanie rozproszone, 241, 252, 253
  CORBA, 254
  DCOM, 258
  ReST, 263
  RPC, 258
  SOAP, 258, 259
  usługi sieciowe, 257
przewaga nad konkurencją, 119
przydzielanie pamięci metodom, 94
przydzielanie pamięci obiektom, 90
punkty dostępowe do systemu, 163

## R

RecipeML, 205
referencje, 84
referencje do singletonowego obiektu, 286
relacje kompozycji, 175
relacyjne bazy danych, 225, 233
  JDBC, 235
  model klient-serwer, 236
  ODBC, 236
  SQL, 235
  zapisywanie danych, 233
ReST, Representational State Transfer, 263
RMI, 258
RMI, remote method invocation, 258
rodzaje interfejsów, 38
rodzaje kompozycji, 179
rodzaje wzorców projektowych, 283
rozproszone działanie, 254
rozszerzalność, 105
rozszerzalny język znaczników, 205
rozszerzanie interfejsu, 101
RPC, Remote Procedure Call, 258

## S, Ś

samodzielne oprogramowanie, 58
sekwencyjność, 123
serializacja, 113, 225
  język XML, 231
serializacja metod, 231
serializacja obiektu, 230
serializacja pliku, 227
serializacja, serialization, 113
serwer, 269, 274
serwer WWW, 243
serwis w3schools, 214
SGML, Standard Generalized Markup Language, 205
siec komórkowa, 23
sieć mobilna, 23
składnia języka Java, 89
słownictwo, vocabulary, 204
słowo kluczowe
  private, 90
  static, 90
słowo kluczowe class, 88
słowo kluczowe init, 67
słowo kluczowe interface, 56, 157
słowo kluczowe new, 73
słowo kluczowe New, 67
słowo kluczowe private, 95
słowo kluczowe static, 284
słowo kluczowe this, 81
SOAP, 258
SOAP, Simple Object Access Protocol, 257
solidny artefakt, 292
SOW, statement of work, 119
specyfikacja formatowania elementów, 216
sprawdzanie poprawności dokumentów XML, 214
SQL, Structured Query Language, 235
standaryzacja, 150
static, 105
stos, 46
  wstawienie, push, 46
  zdejmowanie, pop, 46
struktura diagramu klasy, 190



Kup książkę Poleć książkę

strumienie, 227
sygnatura, 70, 71
system informatyczny przedsiębiorstwa, 252
system plików płaskich, 112
system trzywarstwowy, 255
system wielowarstwowy, 254
systemy oprogramowania, 177
szeregowanie obiektów, 113
szeregowanie, marshaling, 113
szukanie błędów, 75
szybkie prototypowanie, 117
śledzenie referencji, 85

## T

tablica asocjacyjna, 219
technika plug-and-play, 150
techniki przetwarzania rozproszonego, 241
testowanie interfejsu, 110
testowanie klasy, 140
testowanie kodu, 132
testy oprogramowania, 118
throw, 76
transfer danych między aplikacjami, 207
transmisja danych przez sieć, 28
trwałość obiektów, 57, 112, 225
tworzenie diagramów klas, 32
tworzenie egzemplarza, 32
tworzenie egzemplarzy klasy, 46
tworzenie klas, 47
tworzenie konstruktora, 69
tworzenie modeli obiektowych, 189
tworzenie modelu klas opisującego system, 121
tworzenie obiektów, 34, 45, 67, 159, 175–88
tworzenie obiektu, 32, 73
tworzenie prototypu interfejsu użytkownika, 121
tworzenie systemu, 177
typ zwrotny, 72
typy danych, 25, 33

## U

ukrywanie danych, 37, 38, 90
ukrywanie danych, data hiding, 26
ukrywanie implementacji, 101, 107
UML, 116
    agregacja, 137, 196
    asocjacja, 137, 197
    atrybuty, 192
    dziedziczenie, 194
    dziedziczenie interfejsów, 194
    interfejsy, 195
    kompozycja, 137, 196
    liczność, 199
    metody, 192
    określanie dostępności, 193
    wielokrotne dziedziczenie, 194
UML, Unified Modeling Language, 32, 189
uruchamianie aplikacji, 270
usługi sieciowe, 257
    ReSTful, 263
usuwanie nieużytków, 103
użytkownik, 54

## W

walidator, 214
wartość null, 91, 186
warunki, 123
weryfikacja danych, 244, 245
weryfikacja niepoprawnego dokumentu, 215
weryfikowanie danych, 246
widok, View, 281
wielodziedziczenie, 194
wielokrotne dziedziczenie, 83, 156
wielokrotne wykorzystanie kodu, 40, 104, 149, 151
wielokrotne wykorzystanie obiektów, 129
własności, 30
wskaźnik this, 248
współpraca między klasami, 121
wyciek pamięci, 103
wyjątek, exception, 76



Kup książkę Poleć książkę

wyjątki, 76
wykrywanie błędów, 166
wymagania, 120
wymiana danych, 208
wytyczne dotyczące projektowania, 115
wywołanie konstruktora, 68
wywołanie metody, 29
wyznaczanie trasy, 257
wzorce czynnościowe
  Interpretator, 290
  Iterator, 290
  Łańcuch Odpowiedzialności, 290
  Mediator, 290
  Metoda Szablonowa, 290
  Obserwator, 290
  Pamiątka, 290
  Polecenie, 290
wzorce czynnościowe
  Stan, 290
  Strategia, 290
wzorce konstrukcyjne
  Budowniczy, 283
  Fabryka Abstrakcyjna, 283
  Metoda Fabrykująca, 283
  Prototyp, 283
  Singleton, 283
wzorce projektowe, 279
  konsekwencje, 281
  nazwa, 281
  problem, 281
  rozwiązanie, 281
wzorce strukturalne
  Adapter, 288
  Dekorator, 288
  Fasada, 288
  Kompozyt, 288
  Most, 288
  Pośrednik, 288
  Waga Piórkowa, 288
wzorzec
  Adapter, 288
  Iterator, 290
  MVC, 280
    wady, 282
  Singleton, 283

## X

XML, 204, 206, 207, 214, 271
  dokumenty, 206
  integracja z DTD, 210
  PCDATA, 210
  poprawność dokumentu, 210
  serializacja, 231
  specyfikacja danych do wymiany, 208
  specyfikacja formatowania
    elementów, 216
  sprawdzanie poprawności
    dokumentu, 208
XML, Extensible Markup Language, 205

## Z

zachowania obiektu, 29, 63
zadania klienta, 273
zadania serwera, 274
zakres, 78, 107
zakres planowanych prac, 119
zakres, scope, 78
zapętlenie lokalne, 268
zapis liczności, 184
zapisywanie obiektu, 229
zapisywanie obiektu w pliku płaskim,
  226
zastosowanie obiektów, 241
zaślepki, 110, 112
zawieranie kontraktu, 149, 161
zdalne wywoływanie procedur, 258
zestawy, assembly, 203
zintegrowane środowisko
  programistyczne, 120
złożoność modelu, 135, 138
znak –, 39
znak +, 39
związek dziedziczenia, 176
związek typu „jest”, 42, 129, 131, 159, 160
związek typu „ma”, 47, 130



Kup książkę Poleć książkę

Kup książkę Poleć książkę

# 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

GRUPA WYDAWNICZA
Helion SA

Programowanie

O'REILLY®

# JavaScript dla programistów PHP

Czasy wąskiej specjalizacji programistów odchodzą w przeszłość. Współczesne projekty wymagają od nich szerokiej wiedzy, dotyczącej zarówno języków działających po stronie serwera (Java, PHP), jak i przeglądarki (JavaScript). Jest jednak światełko w tunelu – jeśli znasz tylko jeden z języków programowania, jesteś w stanie sprawnie opanować kolejny. Dzięki tej książce to zadanie stanie się prostsze!

Książka jest przeznaczona dla programistów PHP chcących szybko nauczyć się programowania w języku JavaScript. W trakcie lektury poznasz różnice i podobieństwa między tymi dwoma językami. W kolejnych rozdziałach rozgryziesz składnię JavaScriptu i błyskawicznie zaczniesz używać funkcji wbudowanych w ten język. Ponadto opanujesz programowanie obiektowe oraz sposoby jego wykorzystania. Na koniec nauczysz się testować kod napisany w JavaScripcie. To najlepszy podręcznik do nauki tego języka dla osób znających podstawy PHP.

***Dzięki tej książce:***

- poznasz składnię języka JavaScript
- będziesz programować w nim obiektowo
- zaznajomisz się z funkcjami wbudowanymi
- błyskawicznie poznasz kolejny język programowania

***Poznaj możliwości języka JavaScript i wykorzystaj je w Twoim projekcie!***


sięgnij po WIĘCEJ

KOD KORZYŚCI

helion.pl
księgarnia internetowa

Nr katalogowy: 16327

Księgarnia internetowa:
http://helion.pl

Zamówienia telefoniczne:
0 801 339900
0 601 339900

Helion

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



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

ISBN 978-83-246-8217-1

9 788324 682171

Cena 32,90 zł


Informatyka w najlepszym wydaniu