

Najlepsze rozwiązania Twoich problemów!

PHP

Wzorce projektowe



O'REILLY®

William Sanders

Tytuł oryginału: Learning PHP Design Patterns

Tłumaczenie: Jakub Hubisz

ISBN: 978-83-246-7455-8

© 2013 Helion S.A.

Authorized Polish translation of the English edition of Learning PHP Design Patterns, ISBN 9781449344917 © 2013 William B. Sanders.

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

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/phpwzo>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:

<ftp://ftp.helion.pl/przyklady/phpwzo.zip>

Printed in Poland.

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

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

Spis treści

Przedmowa	13
I Wstęp do wzorców projektowych.....	19
1. PHP a programowanie obiektowe.....	21
Wstęp do średnio zaawansowanego i zaawansowanego programowania	21
Dlaczego programowanie zorientowane obiektowo?	22
W jaki sposób ułatwić rozwiązywanie problemów?	22
Modularyzacja	23
Klasy i obiekty	23
Zasada jednej odpowiedzialności	24
Konstruktory w PHP	24
Klient jako klasa żądająca	25
Co z wydajnością?	28
Szybkość tworzenia i modyfikacji rozwiązań	28
Szybkość w zespole	29
Problemy z programowaniem sekwencyjnym i proceduralnym	29
Programowanie sekwencyjne	29
Programowanie proceduralne	30
Zapłać teraz lub zapłać potem	30
2. Podstawowe zagadnienia programowania obiektowego.....	33
Abstrakcja	33
Klasy abstrakcyjne	34
Właściwości i metody abstrakcyjne	35
Interfejsy	37
Interfejsy i stałe	38
Podpowiadanie typów: prawie typowanie	39

Enkapsulacja	41
Enkapsulacja w życiu codziennym	41
Zapewnienie enkapsulacji poprzez widoczność	42
Funkcje ustawiające i pobierające	44
Dziedziczenie	46
Polimorfizm	47
Jedna nazwa, wiele implementacji	49
Polimorfizm wbudowany we wzorcach projektowych	50
Tylko spokojnie	50
3. Podstawowe zagadnienia wzorców projektowych	51
Wprowadzenie do MVC	51
Podstawowe zasady wzorców projektowych	53
Pierwsza zasada wzorców projektowych	54
Wykorzystanie typów interfejsowych przy podpowiadaniu typów	54
Klasy abstrakcyjne i ich interfejsy	55
Druga zasada wzorców projektowych	58
Przykład złożoności przy wykorzystaniu klienta	58
Delegacja: różnica między MA a JEST	61
Wzorce projektowe jako wielka ściaga	61
Organizacja wzorców projektowych	62
Wybór wzorca projektowego	63
Czym spowodowana jest konieczność powtórnego projektowania?	63
Co się zmienia?	63
Czym różnią się wzorce projektowe od frameworków?	64
4. Wykorzystanie UML-a we wzorcach projektowych	65
Po co język modelowania UML?	65
Diagramy klas	66
Symbole uczestnictwa	67
Notacja relacji	69
Relacje asocjacji	69
Relacje agregacji	71
Relacje implementacji i dziedziczenia	72
Relacje tworzenia	74
Relacje wielokrotne	74
Diagramy obiektów	75
Diagramy interakcji	76
Rola diagramów i notacji w programowaniu obiektowym	77
Narzędzia dla UML	77
Inne diagramy UML	78

II Wzorce kreacyjne	79
5. Wzorzec Metoda Fabrykująca	81
Czym jest wzorzec Metoda Fabrykująca?	81
Kiedy korzystać z Metody Fabrykującej	82
Minimalistyczny przykład	83
Praca z fabryką	83
Klient	85
Wprowadzanie zmian w klasach	86
Dodanie elementów graficznych	86
Koordynowanie produktów	87
Zmiany w produkcie tekstowym	88
Zmiany w produkcie graficznym	89
Dodanie nowych produktów i parametryzowanych żądań	90
Jedna fabryka i wiele produktów	91
Nowe fabryki	91
Nowe produkty	92
Klient z parametrem	93
Klasy pomocnicze	94
Diagram plików	95
Produkty się zmieniają, interfejs zostaw w spokoju	96
6. Wzorzec Prototyp.....	99
Czym jest wzorzec Prototyp?	99
Kiedy wykorzystać wzorzec Prototyp	99
Funkcja klonująca	100
Podczas klonowania konstruktor nie jest wywoływany	102
W funkcji konstruktora nie powinny być wykonywane żadne zadania	103
Minimalistyczny przykład	103
Badanie muszek owocówek	103
Dodanie do prototypu elementów obiektowych	106
Nowoczesna organizacja biznesowa	107
Enkapsulacja w interfejsie	107
Implementacje interfejsów	108
Klient	111
Wprowadzanie zmian, rozbudowa funkcjonalności	113
Dynamiczne tworzenie instancji obiektów	114
Wzorzec Prototyp w świecie PHP	115

III	Wzorce strukturalne	117
7.	Wzorzec Adapter.....	119
	Czym jest wzorzec Adapter?	119
	Kiedy wykorzystywać wzorzec Adapter	120
	Wzorzec Adapter zaimplementowany przy wykorzystaniu dziedziczenia	122
	Minimalistyczny przykład adaptera klas: wymiana walut	122
	Wzorzec Adapter zaimplementowany przy wykorzystaniu kompozycji	126
	Z komputerów osobistych na urządzenia mobilne	126
	Adaptory a zmiany	133
8.	Wzorzec Dekorator	135
	Czym jest wzorzec Dekorator?	135
	Kiedy wykorzystywać wzorzec Dekorator	136
	Minimalistyczny przykład dekoratora	137
	Interfejs komponentu	137
	Interfejs dekoratora	137
	Konkretny komponent	138
	Konkretne dekoratory	139
	Klient	141
	Co z opakowywaniem?	142
	Opakowywanie prymitywów	142
	Klasy i funkcje opakowujące wbudowane w PHP	142
	Wzorce projektowe polegające na opakowywaniu	143
	Dekoratory z wieloma komponentami	143
	Wiele konkretnych komponentów	144
	Konkretne komponenty z wieloma stanami i wartościami	144
	Usługi randkowe dla programistów	145
	Interfejs użytkownika (UI) w HTML	150
	Klasa Client przekazująca dane z HTML	154
	Od nazwy zmiennej do instancji obiektu	155
	Dodawanie dekoracji	155
IV	Wzorce czynnościowe.....	157
9.	Wzorzec Metoda Szablonowa.....	159
	Czym jest wzorzec Metoda Szablonowa?	159
	Kiedy wykorzystywać wzorzec Metoda Szablonowa	160
	Wykorzystanie wzorca Metoda Szablonowa z obrazami i podpisami:	
	minimalistyczny przykład	160
	Klasa abstrakcyjna	161
	Klasa konkretna	161

Klasa Client	162
Zasada Hollywood	162
Wykorzystanie Metody Szablonowej z innymi wzorcami projektowymi	164
Zmniejszenie obowiązków klienta	165
Uczestnicy wzorca Metoda Szablonowa	166
Uczestnicy wzorca Metoda Fabrykująca	167
Hak we wzorcu Metoda Szablonowa	169
Ustawienie haka	171
Implementacja haka	171
Klasa Client i uruchomienie haka	172
Mały, ale potężny wzorzec Metoda Szablonowa	174
10. Wzorzec Stan.....	175
Czym jest wzorzec Stan?	175
Kiedy korzystać ze wzorca Stan	176
Maszyna stanowa	177
Światło włączone i wyłączone: minimalistyczna implementacja wzorca	178
Kontekst jest najważniejszy	178
Stany	181
Klasa Client wykonuje żądanie przez Context	182
Dodawanie stanów	183
Zmiana interfejsu	183
Zmiana stanów	184
Aktualizacja klasy Context	186
Zaktualizowana klasa Client	187
Nawigator: więcej wyborów i komórek	188
Ustalenie szablonu stanów dla macierzy	189
Przygotowanie interfejsu	189
Kontekst	190
Stany	192
Klient wybiera ścieżkę	197
Wzorzec Stan a PHP	198
V MySQL i wzorce projektowe PHP	199
11. Uniwersalna klasa połączeniowa i wykorzystanie wzorca Proxy dla bezpieczeństwa.....	201
Prosty interfejs i klasa do połączeń MySQL	201
Interfejs w ciąży	202
Uniwersalna klasa połączeniowa MySQL i zmienne statyczne	202
Prosty klient	204

Proxy zabezpieczające proces logowania	204
Stworzenie formularza rejestracji	205
Implementacja proxy logowania	209
Proxy i bezpieczeństwo w prawdziwym świecie	214
12. Elastyczny wzorzec projektowy Strategia.....	217
Algorytmy enkapsulujące	217
Różnica między wzorcami Strategia i Stan	218
Żadnych poleceń warunkowych, proszę	219
Rodzina algorytmów	219
Minimalistyczny wzorzec Strategia	219
Klasa Client i skrypty wyzwalające	221
Klasa Context i interfejs strategii	223
Konkretne strategie	224
Rozszerzony wzorzec Strategia uwzględniający bezpieczeństwo danych i parametryzowane algorytmy	227
Klasa pomocnicza odpowiedzialna za bezpieczeństwo danych	227
Dodanie parametru do metody algorithm	230
Tabela survey	230
Moduły wprowadzania danych	231
Klasa Client wzywa pomocy	235
Drobna, ale ważna zmiana w klasie Context	237
Konkretne strategie	237
Elastyczny wzorzec Strategia	242
13. Wzorzec projektowy Łańcuch Odpowiedzialności.....	245
Podaj dalej	245
Łańcuch Odpowiedzialności w aplikacji pomocy technicznej	247
Utworzenie i uzupełnienie tabeli odpowiedzi	247
Łańcuch Odpowiedzialności pomocy technicznej	251
Zautomatyzowany Łańcuch Odpowiedzialności i Metoda Fabrykująca	256
Łańcuch Odpowiedzialności i żądania w oparciu o datę	256
Metoda Fabrykująca wieńczy dzieło	261
Łatwość aktualizacji	265
14. Budowa systemu CMS za pomocą wzorca Obserwator.....	267
Wbudowane interfejsy Obserwatora	267
Kiedy korzystać ze wzorca Obserwator	268
Wykorzystanie SPL we wzorcu Obserwator	269
SplSubject	270
SplObserver	270
SplObjectStorage	271

Konkretny podmiot SPL	271
Konkretny obserwator SPL	272
Klient SPL	273
Czyste PHP i wzorzec Obserwator	274
Implementacja klas abstrakcyjnych Subject i ConcreteSubject	274
Interfejs Observer i wiele konkretnych obserwatorów	276
Klient	277
Budowa prostego systemu CMS	278
Infrastruktura programu	279
Obserwator dla wielu urzędzeń	283
Myślenie obiektowe	294
Skorowidz	295

Wzorzec Metoda Fabrykująca

Ruchy kobiet wywołały wśród pracowników fabryk wielką mobilizację, która zniszczyła dawne wzorce.

— Emma Bonino

Projektowanie to metoda łączenia formy i zawartości. Projektowanie, jako dziedzina sztuki, ma wiele definicji; nie ma jednej definicji. Projektowanie może być sztuką. Projektowanie może być estetyczne. Projektowanie jest bardzo proste, dlatego właśnie jest takie skomplikowane.

— Paul Rand

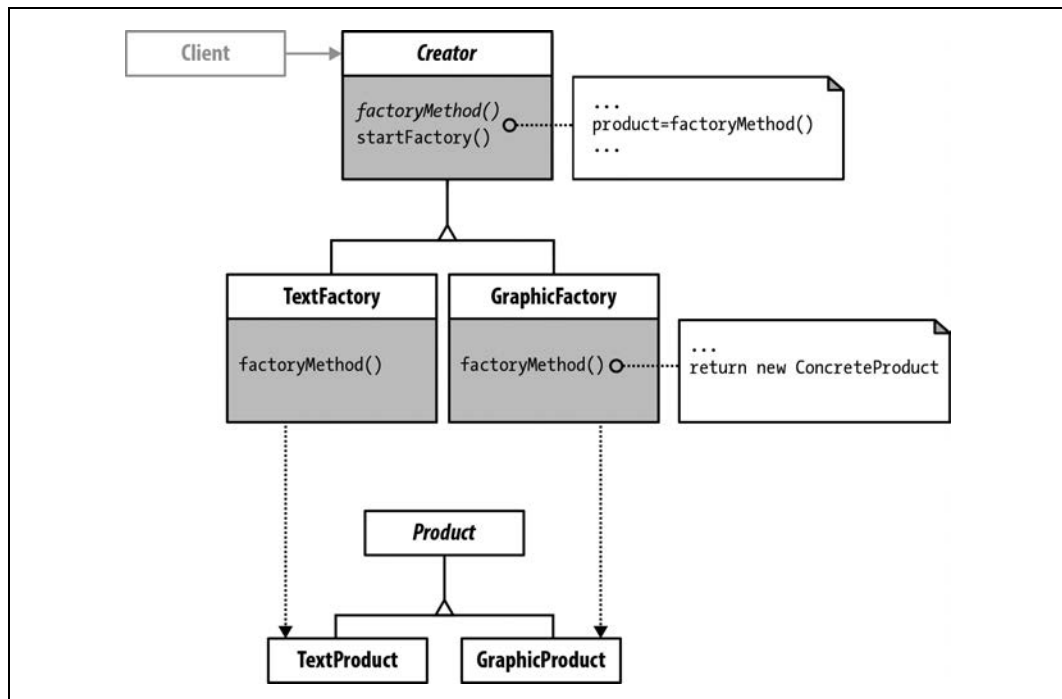
Stwórz swoją własną metodę. Nie polegaj ślepo na mojej. Wymyśl coś, co sprawdzi się dla Ciebie! Ale łam konwencje, błagam.

— Konstanty Stanisławski

Czym jest wzorzec Metoda Fabrykująca?

Jako część kategorii wzorców kreacyjnych wzorzec Metoda Fabrykująca ma za zadanie *coś* stworzyć. W przypadku tego wzorca to *coś* to produkt niezwiązany z klasą, która go stworzyła. Aby zachowane zostało luźne łączenie, klient przesyła żądanie przez fabrykę. Następnie fabryka tworzy żądany produkt. Można też powiedzieć, że Metoda Fabrykująca uwalnia produkt od klienta wykonującego żądanie. Rysunek 5.1 przedstawia diagram klas dla wzorca.

Klasa `Client` jest domyślna. Jak widać na rysunku 5.1, `Client` przechowuje referencję do klasy `Creator` (interfejs fabryki), dzięki której może zażądać produktu graficznego lub tekstowego. Nie tworzy instancji żadanego produktu. Instancje tworzone są w fabrykach konkretnych. Wyobraź sobie, że na przyjęcie z okazji Halloween chcesz zamówić czekoladowe babeczki z czarnym i pomarańczowym lukrem. Dzwonisz do piekarza (`creator`), który robi dla Ciebie babeczki (`product`). Nie angażujesz się w proces tworzenia obiektu, który zamówiłeś, a otrzymujesz gotowy produkt.



Rysunek 5.1. Diagram klas zaimplementowanego wzorca Metoda Fabrykująca

Kiedy korzystać z Metody Fabrykującej

Wzorce projektowe częściowo wybiera się na podstawie zmian, jakie chcesz wprowadzać w przyszłości w programie. Tabela 3.1 z rozdziału 3. pokazuje, że ten wzorzec powinien być wykorzystywany, kiedy podklasa tworzonego obiektu może się zmieniać. W przykładach z tego rozdziału wszystkie podklasy interfejsu `Project` są zmienne — oznaczają różne kraje. Jak się przekonasz, projekty to obiekty składające się z tekstu (zapiski) lub grafiki (mapy). Na początku zakładamy, że programista nie wie, ile krajów może pojawić się w programie. Innymi słowy, liczba i typy obiektów są nieznane. Klasa nie może przewidzieć liczby obiektów, które będzie musiała stworzyć, a więc nie chcemy, aby była z tymi obiektami silnie związana.

Jeżeli klasa ma skończoną i znaną liczbę obiektów, które muszą zostać utworzone, można napisać ją tak, aby obiekty były tworzone w przewidywalny sposób. Na przykład: jeżeli robisz aplikację mapy świata z odrębnymi obiektami dla każdego z siedmiu kontynentów, możesz być pewien, że ta liczba nie ulegnie zmianie. Z drugiej strony, jeżeli tworzysz stronę zawierającą różne gatunki insektów, możesz być pewien, że gatunki będą odkrywane, zmieniane, a część z nich wyginie w niedługich odstępach czasu. Produkt programistyczny, który poradzi sobie z taką zmiennością, musi być bardzo elastyczny. Właśnie w takim projekcie powinieneś rozważyć wykorzystanie wzorca projektowego Metoda Fabrykująca.

Minimalistyczny przykład

Pierwszy przykład, rozpoczynający naszą pracę z wzorcem Metoda Fabrykująca, zwraca tylko tekst. Wzorec tworzony jest na potrzeby programu, w którym programista wie, że będzie musiał tworzyć obiekty tekstowe i graficzne dla projektu zawierającego mapy i notatki. Nie wie natomiast, ile dokładnie par map i notatek będzie potrzebnych, nie wie też, co klient będzie chciał dodać do programu. Powiedziano mu, że konieczne będą mapy w formie graficznej i opisujący je tekst. Na początku tworzy skromny wzorec Metoda Fabrykująca, wyświetlający na ekranie informacje tekstowe — część w formie „grafiki”, część w formie „tekstu”. Jeżeli rozwiązanie zostanie zaprojektowane prawidłowo, nie będzie problemu z przechowywaniem dowolnej liczby obiektów.

Praca z fabryką

Pierwszy krok to stworzenie fabryki: interfejs `Creator`. W tej implementacji jako interfejs wykorzystamy klasę abstrakcyjną. Jeżeli przyjrzymy się bliżej diagramowi klas, zauważymy, że jedna z adnotacji zawiera metodę konkretną — `startFactory()`. Dzięki temu wiemy, że interfejs musi być klasą abstrakcyjną, a nie interfejsem — interfejsy mogą zawierać tylko metody abstrakcyjne. Projekt wymaga też metody abstrakcyjnej `factoryMethod()`. W klasie abstrakcyjnej wszystkie takie metody muszą być abstrakcyjne; w innym przypadku będą traktowane jako konkretne. Plik `Creator.php` zawiera kod pierwszego uczestnika wzorca.

```
<?php
//Creator.php
abstract class Creator
{
    protected abstract function factoryMethod();
    public function startFactory()
    {
        $mfg= $this->factoryMethod();
        return $mfg;
    }
}
?>
```

Zwróć uwagę, że adnotacje na diagramie wskazują, że metoda `startFactory()` musi zwracać **produkt**. W implementacji metoda `startFactory()` oczekuje, że `factoryMethod()` zwróci obiekt produktu. W związku z tym konkretna implementacja metody `factoryMethod()` musi stworzyć instancję obiektu implementowanego z interfejsu `Product`.

Dwie konkretne klasy fabryk dziedziczą po klasie `Creator` i implementują metodę `factoryMethod()`. Implementacja metody `factoryMethod()` zwraca produkt graficzny lub tekstowy za pośrednictwem metody klasy `Product` — `getProperties()`. Implementacje `TextFactory` i `GraphicFactory` zawierają następujące elementy:

```
<?php
//TextFactory.php
include_once('Creator.php');
include_once('TextProduct.php');
class TextFactory extends Creator
{
    protected function factoryMethod()
    {
        $product=new TextProduct();
        return($product->getProperties());
    }
}
```

```

    }
}
?>

<?php
//GraphicFactory.php
include_once('Creator.php');
include_once('GraphicProduct.php');
class GraphicFactory extends Creator
{
    protected function factoryMethod()
    {
        $product=new GraphicProduct();
        return($product->getProperties());
    }
}
?>

```

Obie implementacje są bardzo podobne — różnią się tylko tym, że jedna tworzy obiekt typu `TextProduct`, a druga — typu `GraphicProduct`.

Produkt

Drugi interfejs we wzorcu Metoda Fabrykująca to `Product`. W naszej minimalistycznej implementacji wszystkie klasy implementujące interfejs muszą zaimplementować tylko jedną metodę, `getProperties()`.

```

<?php
//Product.php
interface Product
{
    public function getProperties();
}
?>

```

Dzięki temu, że klasa nie ma właściwości, możemy w klasach potomnych zdecydować, co dokładnie chcemy zrobić z metodą `getProperties()`. W PHP, w którym sygnatura zawiera tylko nazwę i widoczność, możemy dowolnie modyfikować metodę abstrakcyjną, włączając w to zwracanie wartości — a jeżeli tylko sygnatura się zgadza, implementacja będzie poprawna.

W implementacji Metody Fabrykującej można zaobserwować **polimorfizm** na przykładzie metody `getProperties()`. Zostanie wykorzystana do zwrócenia tekstu lub grafiki, ale wiemy, że dopóki będzie posiadała poprawną sygnaturę, będzie dostarczała to, czego potrzebujemy. Dokładnie ta sama metoda, `getProperties()`, posiada wiele (*poli*) różnych form (*morf*). W tym przypadku jedna z form zwraca tekst, a druga grafikę:

```

<?php
//TextProduct.php
include_once('Product.php');
class TextProduct implements Product
{
    private $mfgProduct;
    public function getProperties()
    {
        $this->mfgProduct="To jest tekst.";
        return $this->mfgProduct;
    }
}
?>

```

Możesz pomyśleć: „Wielka mi rzecz, to zwraca tylko zmienną tekstową”. Na razie jest to prawda. Możesz jednak w implementacji zawrzeć, co tylko chcesz, a Metoda Fabrykująca stworzy to i zwróci do metody Client. Kiedy więc zobaczysz wynik „To jest tekst” lub „To jest grafika”, wyobraź sobie dowolny obiekt, który mógłbyś chcieć stworzyć. Kolejna implementacja zwraca abstrakcyjną grafikę w formie tekstu:

```
<?php
//GraphicProduct.php
include_once('Product.php');
class GraphicProduct implements Product
{
    private $mfgProduct;
    public function getProperties()
    {
        $this->mfgProduct="To jest grafika.<3";
        return $this->mfgProduct;
    }
}
?>
```

Każda z dwóch implementacji fabryki i produktu implementuje metody abstrakcyjne w celu stworzenia dwóch różnych fabryk i produktów, stosując się do interfejsów.

Klient

Ostatnim uczestnikiem wzorca jest domyślny klient. Nie chcemy, aby klasa Client wysyłała żądania bezpośrednio do produktów. Chcemy natomiast, aby żądania były przesyłane przez interfejs Creator. Później, jeżeli będziemy dodawać produkty lub fabryki, klient będzie mógł wykonać takie samo żądanie i uzyskać o wiele szerszą gamę produktów bez konieczności wprowadzania zmian w programie:

```
<?php
//Client.php
include_once('GraphicFactory.php');
include_once('TextFactory.php');
class Client
{
    private $someGraphicObject;
    private $someTextObject;
    public function __construct()
    {
        $this->someGraphicObject=new GraphicFactory();
        echo $this->someGraphicObject->startFactory() . "<br />";
        $this->someTextObject=new TextFactory();
        echo $this->someTextObject->startFactory() . "<br />";
    }
}

$worker=new Client();
?>
```

Jeżeli wszystko zadziała tak jak powinno, otrzymasz wynik:

```
To jest grafika.<3
To jest tekst.
```

Zauważ, że klient nie wykonał żadnego bezpośredniego żądania do produktu, a instancje uzyskuje poprzez fabrykę. Ważne jest to, że klient pozostawia kwestie związane z charakterystyką produktu poszczególnym implementacjom.

Wprowadzanie zmian w klasach

Największą wartością wzorców projektowych nie jest szybkość wykonywania operacji, ale szybkość tworzenia rozwiązań. W prostych aplikacjach, takich jak przykład wzorca Metoda Fabrykująca, może być ciężko to zaobserwować. Kiedy jednak zaczniemy wprowadzać zmiany, będzie to lepiej widoczne.

Dodanie elementów graficznych

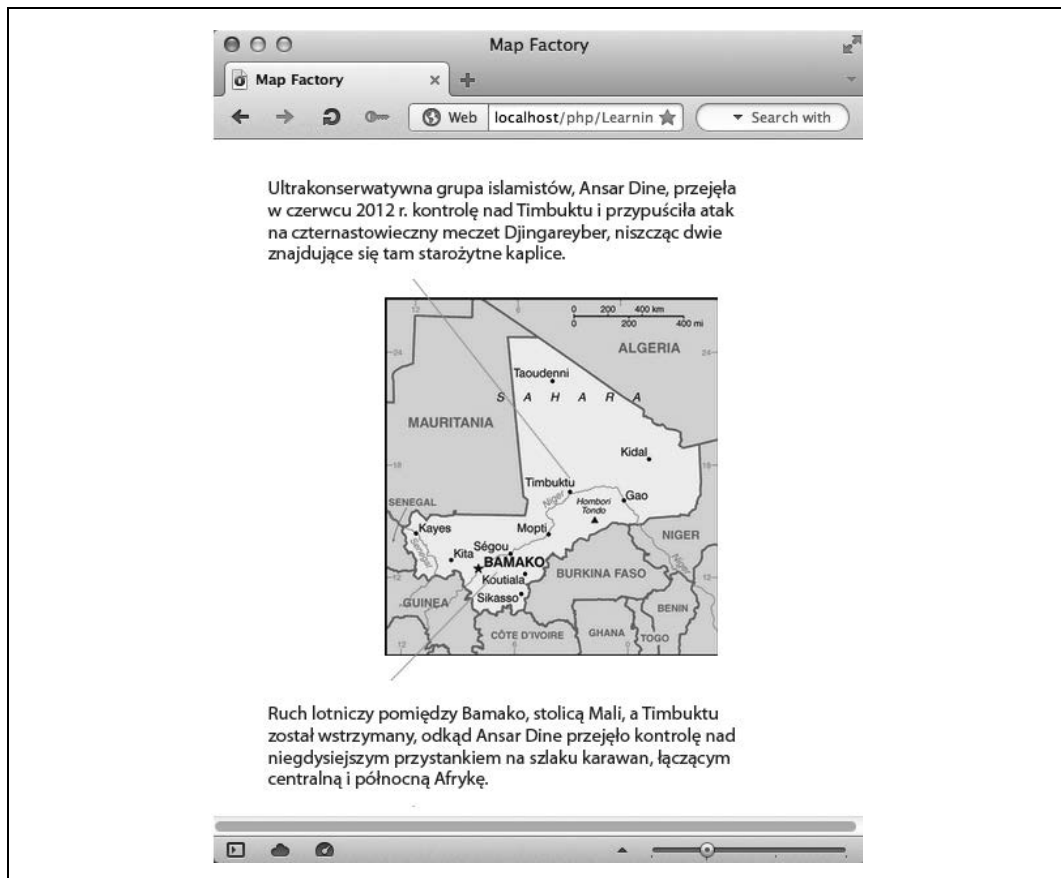
Pierwszym krokiem będzie zmienienie produktu, tak aby łądował grafikę w dokumencie HTML. Samo w sobie dodawanie grafiki na stronie jest bardzo proste, ale kiedy Twój program PHP będzie stał się bardziej złożony, może się to stać trudniejsze. Poniższy listing przedstawia zmodyfikowaną klasę `GraphicProduct`:

```
<?php
//GraphicProduct.php
include_once('Product.php');
class GraphicProduct implements Product
{
    private $mfgProduct;
    public function getProperties()
    {
        $this->mfgProduct="<!doctype html><html><head><meta charset='UTF-8' />";
        $this->mfgProduct.="<title>Fabryka map</title>";
        $this->mfgProduct.="</head><body>";
        $this->mfgProduct.="<img src='Mali.png' width='500' height='500' />";
        $this->mfgProduct.="</body></html>";
        return $this->mfgProduct;
    }
}
?>
```

Po raz kolejny możesz zaobserwować polimorfizm. Ta sama metoda `getProperties()` otrzymała zupełnie inną implementację. Czy to oznacza, że klient będzie musiał zmienić sposób wysyłania żądania? Nie, przy żądaniu grafiki opuszczane jest żądanie tekstu, zgodnie z poniższym listingiem:

```
<?php
//Client.php
include_once('GraphicFactory.php');
class Client
{
    private $someGraphicObject;
    private $someTextObject;
    public function __construct()
    {
        $this->someGraphicObject=new GraphicFactory();
        echo $this->someGraphicObject->startFactory() . "<br />";
    }
}
$worker=new Client();
?>
```

To proste żądanie jest identyczne jak w pierwszym przykładzie. Ponieważ jednak zmienił się obiekt `GraphicProduct`, wynik działania programu będzie inny:



Rysunek 5.2. Obiekt graficzny z osadzonym tekstem

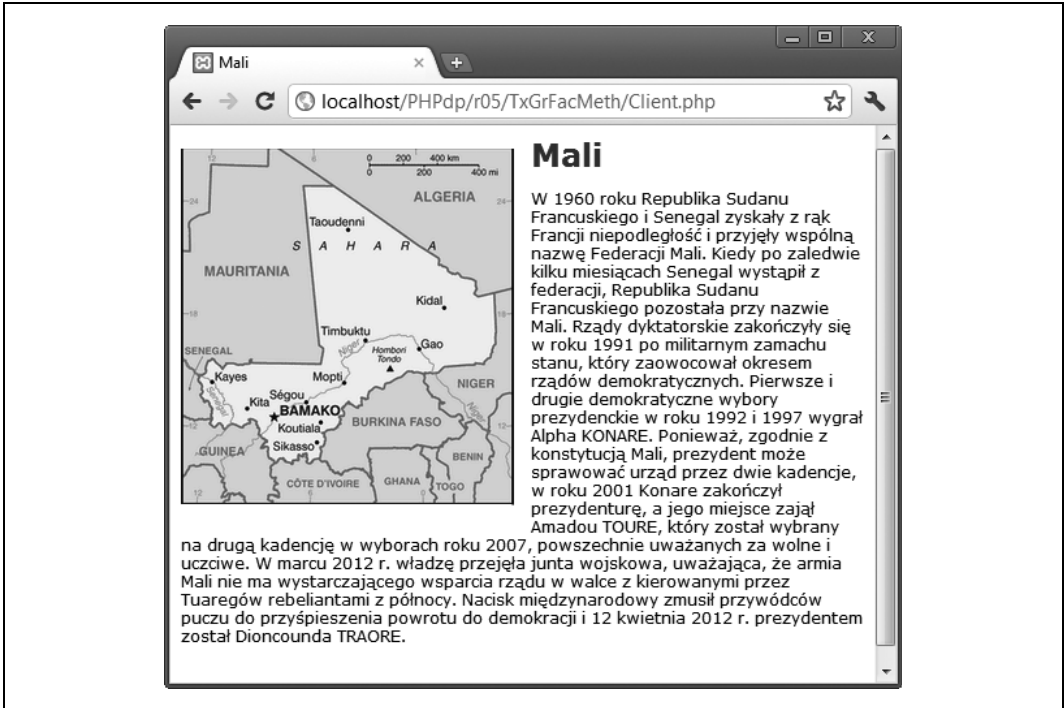
Patrząc na rysunek 5.2, można zaobserwować, że rysunek zawiera tekst, jednak ten tekst jest częścią obrazka, a nie tekstem zawartym w dokumencie HTML.

Koordynowanie produktów

Pobieranie produktów w PHP nie jest trudne, jednak kiedy strona będzie się rozrastać i stanie się bardziej złożona, maksymalne ułatwienie procesu wprowadzania zmian będzie bardzo istotne. Kolejnym krokiem będzie zmodyfikowanie tekstu i grafiki tak, aby mogły zostać umieszczone razem w dokumencie.

Jak widać na rysunku 5.2, tekst został wklejony w obrazek, więc zmiany pozwoliły na pokazanie, że obrazek może zostać załadowany bez konieczności wprowadzania zmian w żądaniach klasy Client. Czy to samo można osiągnąć, jeżeli trzeba skoordynować więcej niż jeden produkt?

Dzięki faktom dotyczącym Mali pobranym ze stron CIA połączyliśmy mapę i notatkę, zgodnie z rysunkiem 5.3.



Rysunek 5.3. Koordynacja obiektów tekstowego i graficznego

Metoda Fabrykująca pomaga uprościć wymagania rosnących i zwiększających poziom złożoności stron podczas wprowadzania i modyfikowania nowych produktów. Aby stworzyć stronę przedstawioną na rysunku 5.3, musimy wprowadzić zmiany jedynie w produktach tekstowym i graficznym. Pozostali uczestnicy wzorca pozostają bez zmian — ponieważ żądanie uzależnione jest od interfejsów, a nie od klas konkretnych.

Zmiany w produkcie tekstowym

Zmiany wprowadzane w produkcie tekstowym są relatywnie proste. Zwracany produkt będzie zawierał formatowanie i nagłówki, ale do klienta wysyłającego żądanie zwracana jest ta sama zmienna, która jest wyświetlana na ekranie. Poniższy listing przedstawia zmiany wprowadzone w klasie `TextProduct`:

```
<?php
//TextProduct.php
include_once('Product.php');
class TextProduct implements Product
{
    private $mfgProduct;

    public function getProperties()
    {
        $this->mfgProduct =<<<MALI
        <!doctype html>
        <html><head>
        <style type="text/css">
        header {
            color: #900;
```

```

        font-weight: bold;
        font-size: 24px;
        font-family: Verdana, Geneva, sans-serif;
    }
    p {
        font-family: Verdana, Geneva, sans-serif;
        font-size: 12px;
    }
</style>
<meta charset="UTF-8"><title>Mali</title></head>
<body>
<header>Mali</header>
<p>W 1960 roku Republika Sudanu Francuskiego i Senegal zyskały z rąk Francji
↳niepodległość i przyjęły wspólną nazwę Federacji Mali. Kiedy po zaledwie kilku
↳miesiącach Senegal wystąpił z federacji, Republika Sudanu Francuskiego pozostała
↳przy nazwie Mali. Rządy dyktatorskie zakończyły się w roku 1991 po militarnym
↳zamachu stanu, który zaowocował okresem rządów demokratycznych. Pierwsze i drugie
↳demokratyczne wybory prezydenckie w roku 1992 i 1997 wygrał Alpha KONARE. Ponieważ,
↳zgodnie z konstytucją Mali, prezydent może sprawować urząd przez dwie kadencje,
↳w roku 2001 Konare zakończył prezydenturę, a jego miejsce zajął Amadou TOURE, który
↳został wybrany na drugą kadencję w wyborach roku 2007, powszechnie uważanych
↳za wolne i uczciwe.
W marcu 2012 r. władzę przejęła junta wojskowa, uważająca, że armia Mali nie ma
↳wystarczającego wsparcia rządu w walce z kierowanymi przez Tuaregów rebeliantami
↳z północy. Nacisk międzynarodowy zmusił przywódców puczu do przyśpieszenia powrotu
↳do demokracji i 12 kwietnia 2012 r. prezydentem został Dioncounda TRAORE. </p>
</body></html>
MALI;
    return $this->mfgProduct;
}
}
?>

```

Zmiany w obiekcie tekstowym wydają się być duże, ale jedyna metoda `getProperties()` zachowała ten sam interfejs i zwraca obiekt do fabryki. Format heredoc pozwala programistom na zawarcie kodu HTML bez konieczności otaczania każdej linii cudzysłowami, a wewnątrz zmiennej heredoc `zmienna` i stałe PHP są akceptowane (w podrozdziale „Klasy pomocnicze” zobaczysz, w jaki sposób klasa pomocnicza może zająć się formatowaniem).

Zmiany w produkcie graficznym

Patrząc na klasę `GraphicProduct`, zobaczymy tę samą metodę i interfejs, które były wykonywane, gdy klasa zwracała tylko tekst symbolizujący grafikę:

```

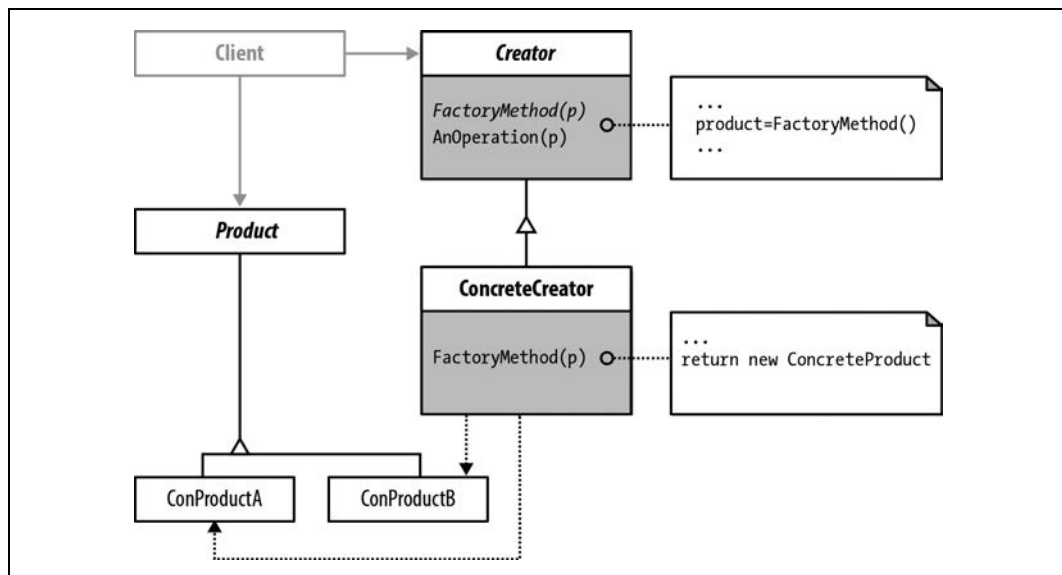
<?php
//GraphicProduct.php
include_once('Product.php');
class GraphicProduct implements Product
{
    private $mfgProduct;
    public function getProperties()
    {
        $this->mfgProduct="<img style='padding: 10px 10px 10px 0px';
        src='Mali.png' align='left' width='256' height='274'>";
        return $this->mfgProduct;
    }
}
?>

```

Podobnie jak w przypadku produktu tekstowego i magii polimorfizmu metoda `getProperties()` jest żywotna jak karaluch. Żadne zmiany nie zostały wprowadzone w obiektach fabryki, a klasa `Client` wykonuje takie samo żądanie, zmieniając tylko rodzaj żądania z tekstu na grafikę.

Dodanie nowych produktów i parametryzowanych żądań

Do tej pory pokazałem, że zmiana grafiki i tekstu nie zaburzy działania wzorca Metoda Fabrykująca, co jednak stanie się, kiedy zacniemy dodawać więcej map i notatek? Czy konieczne będzie dodanie nowej klasy z konkretnej fabryki za każdym razem, kiedy będziemy chcieli dodać nowy kraj? Oznaczałoby to, że dla każdego kraju konieczne będzie stworzenie dwóch klas — fabryki i produktu. Czas przyjrzeć się wzorcowi Metoda Fabrykująca z parametrami. Rysunek 5.4 przedstawia implementację takiego wzorca.



Rysunek 5.4. Pojedyncza klasa `Creator` produkująca wiele produktów

Diagram z rysunku 5.4 różni się w kilku szczegółach od pierwotnego. Oba przedstawiają wzorec Metoda Fabrykująca i oba pozwalają osiągnąć ten sam cel — różnią się jednak implementacją.

Jedną z głównych różnic pomiędzy wzorcem z parametrami a wzorcem, który przedstawiliśmy pierwotnie, jest to, że klient przechowuje referencje do fabryki i produktu. W parametryzowanym żądaniu musi nazwać produkt, a nie tylko fabrykę produktów. Parametr metody `factoryMethod()` reprezentuje produkt, który ma zostać wyprodukowany — klient musi więc wskazać, jaki dokładnie produkt chce uzyskać. Żądanie nadal odbywa się przez interfejs `Creator`. Mimo że klient przechowuje referencję do produktu, nadal jest oddzielony od niego interfejsem `Creator`.

Jedna fabryka i wiele produktów

W większości przypadków Metoda Fabrykująca z parametrami jest mniej złożona, ponieważ klient ma do czynienia tylko z jedną konkretną fabryką. Metoda Fabrykująca posiada jedną metodę, która kieruje tworzeniem odpowiedniego produktu. W poprzednim przykładzie każdy produkt posiadał swoją fabrykę i nie było konieczności przekazywania parametrów.

Aby zaimplementować więcej produktów za pośrednictwem Metody Fabrykującej z parametrami, wystarczy stworzyć dla nich konkretne klasy. Co więcej, skoro produkt powinien zawierać grafikę i tekst, to zamiast tworzyć dwa oddzielne produkty, w tym przykładzie jedna klasa będzie zawierała wszystkie informacje bez złamania zasady mówiącej, że klasa powinna mieć tylko jedną odpowiedzialność. Ta odpowiedzialność to wyświetlenie tekstu i grafiki odpowiadającej danemu krajowi. Ponieważ aplikacja jest bardzo prosta, odpowiedzialność każdego produktu także nie jest złożona.

Nowe fabryki

Nowe fabryki, `Creator` i `CountryCreator`, są podobne do poprzednich, ale zawierają parametr i podpowiadanie typu. Podpowiadanie typu pozwala na programowanie przy wykorzystaniu interfejsu (`Product`), a nie jego konkretnych implementacji.

```
<?php
//Creator.php
abstract class Creator
{
    protected abstract function factoryMethod(Product $product);
    public function doFactory($productNow)
    {
        $countryProduct=$productNow;
        $mfg= $this->factoryMethod($countryProduct);
        return $mfg;
    }
}
?>
```

Jak widać, w klasie abstrakcyjnej `Creator` obie metody, `factoryMethod()` i `startFactory()`, wymagają podania parametru. Co więcej, ponieważ podpowiedź typu wymaga obiektu typu `Product`, a nie implementacji produktu, może być wykorzystana z dowolną konkretną instancją typu `Product`.

Konkretna klasa `CountryCreator` implementuje metodę `factoryMethod()` z odpowiednim parametrem przy wykorzystaniu podpowiadania typów. Oczywiście klasa dziedziczy metodę `startFactory()`, która będzie wykorzystywana w klasie `Client`:

```
<?php
//CountryFactory.php
include_once('Creator.php');
include_once('Product.php');
class CountryFactory extends Creator
{
    private $country;
    protected function factoryMethod(Product $product)
    {
        $this->country=$product;
        return($this->country->getProperties());
    }
}
?>
```

Klasa `CountryCreator` zawiera prywatną zmienną, `$country`, która przechowuje produkt żądany przez klienta. Następnie za pośrednictwem metody `getProperties()` zwracany jest stworzony produkt.

Nowe produkty

Zmiany w konkretnych produktach nie powodują konieczności wprowadzania zmian w interfejsie `Product` — pozostaje on niezmienny:

```
<?php
//Product.php
interface Product
{
    public function getProperties();
}
?>
```

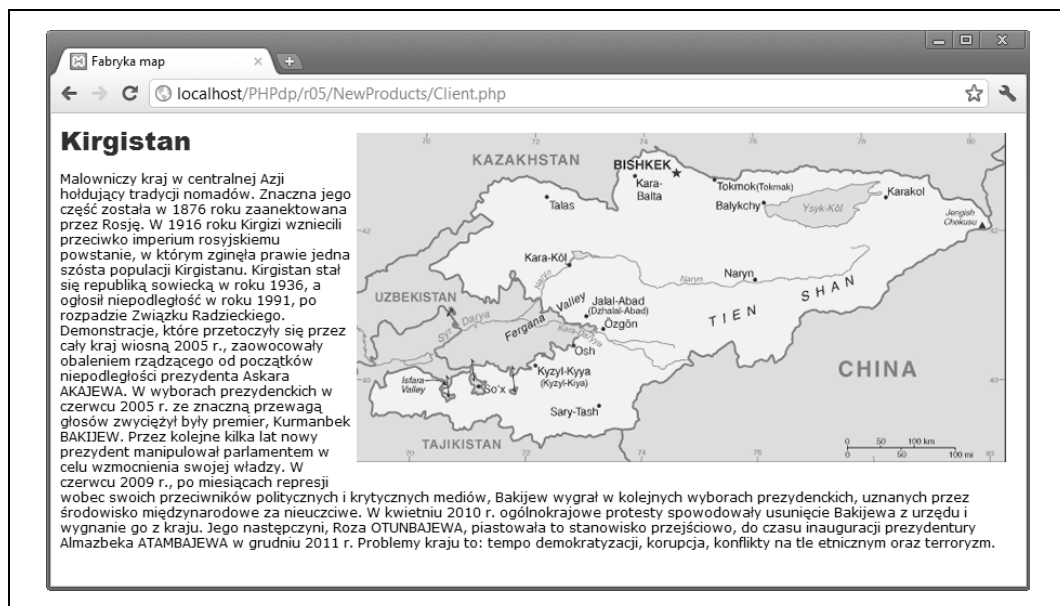
To oznacza, że produkty konkretne muszą posiadać ten sam interfejs i — jak niedługo zobaczysz — mają. Nowe implementacje zawierają jednak zarówno tekst, jak i grafikę. Tekst jest wpisany w ciele klasy (w rzeczywistości powinien być pobierany z pliku tekstowego lub bazy danych), a grafika wywoływana jest przy wykorzystaniu znacznika ``. Poniższa klasa zawiera tekst i grafikę pobraną ze stron CIA (<http://1.usa.gov/akOFIK>):

```
<?php
//KyrgyzstanProduct.php
include_once('FormatHelper.php');
include_once('Product.php');

class KyrgyzstanProduct implements Product
{
    private $mfgProduct;
    private $formatHelper;

    public function getProperties()
    {
        $this->formatHelper=new FormatHelper();
        $this->mfgProduct=$this->formatHelper->addTop();
        $this->mfgProduct.=<<<KYRGYZSTAN
        <img src='Countries/Kyrgyzstan.png' class='pixRight' width='600' height='304'>
        <header>Kirgistan</header>
        <p>Malowniczy kraj w centralnej Azji hołdujący tradycji nomadów. Znaczna jego
        ↳ część została w 1876 roku zaanektowana przez Rosję. W 1916 roku Kirgizi
        ↳ wznieśli przeciwko imperium rosyjskiemu powstanie, w którym zginęła prawie
        ↳ jedna szóstą populacji Kirgistanu. Kirgistan stał się republiką sowiecką w roku
        ↳ 1936, a ogłosił niepodległość w roku 1991, po rozpadzie Związku Radzieckiego.
        ↳ Demonstracje, które przetoczyły się przez cały kraj wiosną 2005 r., zaowocowały
        ↳ obaleniem rządzącego od początków niepodległości prezydenta Askara AKAJEWA.
        ↳ W wyborach prezydenckich w czerwcu 2005 r. ze znaczną przewagą głosów zwyciężył
        ↳ były premier, Kurmanbek BAKIJEW. Przez kolejne kilka lat nowy prezydent
        ↳ manipulował parlamentem w celu wzmocnienia swojej władzy. W czerwcu 2009 r.,
        ↳ po miesiącach represji wobec swoich przeciwników politycznych i krytycznych
        ↳ mediów, Bakijew wygrał w kolejnych wyborach prezydenckich, uznanych przez
        ↳ środowisko międzynarodowe za nieuczciwe. W kwietniu 2010 r. ogólnokrajowe
        ↳ protesty spowodowały usunięcie Bakijewa z urzędu i wygnanie go z kraju. Jego
        ↳ następczyni, Roza OTUNBAJEWA, piastowała to stanowisko przejściowo, do czasu
        ↳ inauguracji prezydentury Almazbeka ATAMBAJEWA w grudniu 2011 r. Problemy kraju
        ↳ to: tempo demokratyzacji, korupcja, konflikty na tle etnicznym oraz terroryzm.</p>
        KYRGYZSTAN;
        $this->mfgProduct .= $this->formatHelper->closeUp();
        return $this->mfgProduct;
    }
}
?>
```

Wynik działania nie uległ zmianie względem pierwotnego rozwiązania, w którym tekst i grafika były umieszczone w osobnych klasach. Rysunek 5.5 przedstawia wynik działania programu.



Rysunek 5.5. Parametryzowane fabryki pozwalają na obsługę wielu produktów

Być może zauważyłeś, że obrazek znajduje się po prawej stronie, zamiast po lewej, i jest większy w porównaniu z mapą Mali — poza tym wynik jest prawie identyczny. Nowością jest dodanie instancji klasy `FormatHelper`. Jest to klasa pomocnicza, której działanie musi zostać wyjaśnione w kontekście wzorca projektowego i tej konkretnej implementacji. Najpierw jednak musimy jeszcze raz przyjrzeć się klasie `Client`, ponieważ także ona uległa zmianie — wymaga podania parametru.

Klient z parametrem

Przykłady z początku rozdziału przedstawiają klasę `Client`, która po prostu wysyła żądanie poprzez interfejs fabryki w celu uzyskania produktu. Po wprowadzeniu zmian żądanie wysyłane jest przy wykorzystaniu parametru.

```
<?php
//Client.php
include_once('CountryFactory.php');
include_once('KyrgyzstanProduct.php');
class Client
{
    private $countryFactory;
    public function __construct()
    {
        $this->countryFactory=new CountryFactory();
        echo $this->countryFactory->doFactory(new KyrgyzstanProduct());
    }
}

$worker=new Client();
?>
```

Klasy pomocnicze

Klasa pomocnicza we wzorcu projektowym jest klasą mającą za zadanie wykonać jakąś operację, którą lepiej wyodrębnić do osobnego obiektu, niż włączyć do jednego z uczestników. O klasie pomocniczej możesz myśleć podobnie jak o zewnętrznym arkuszu stylów CSS. Mógłbyś dodawać te same style do każdej klasy, jednak o wiele wydajniejszym rozwiązaniem jest zawarcie ich w jednym pliku i dołączenie za pomocą znacznika `<link>`. Podobnie jeżeli posiadasz pewien zestaw znaczników formatujących, można je umieścić w odrębnym obiekcie, aby można było wykorzystać go wielokrotnie. Poniższy listing przedstawia wykorzystaną klasę pomocniczą:

```
<?php
class FormatHelper
{
    private $topper;
    private $bottom;

    public function addTop()
    {
        $this->topper="<!doctype html><html><head>
        <link rel='stylesheet' type='text/css' href='products.css' />
        <meta charset='UTF-8'>
        <title>Fabryka map</title>
        </head>
        <body>";
        return$this->topper;
    }

    public function closeUp()
    {
        $this->bottom="</body></html>";
        return$this->bottom;
    }
}
?>
```

Klasa zawiera kod HTML otaczający ciało strony, ale także załącza arkusz stylów CSS — *products.css*. Dla wygody klasa pomocnicza zawiera kod HTML dodawany zwykle na początku i końcu strony w dwóch metodach publicznych: `addTop()` i `closeUp()`. Dzięki temu instancja konkretnego produktu może zostać umieszczona pomiędzy odpowiednimi znacznikami formatującymi HTML.

Arkusze stylów CSS dają programistom dodatkowe możliwości. Dwie klasy z poniższego listingu pozwalają zdecydować, czy rysunek powinien być wyrównany do lewej, czy prawej strony:

```
@charset "UTF-8";
/* CSS Document */
img
{
    padding: 10px 10px 10px 0px;
}

.pixRight
{
    float:right; margin: 0px 0px 5px 5px;
}

.pixLeft
```



```

{
  float:left; margin: 0px 5px 5px 0px;
}

header
{
  color:#900;
  font-size:24px;
  font-family:"Arial Black", Gadget, sans-serif;
}

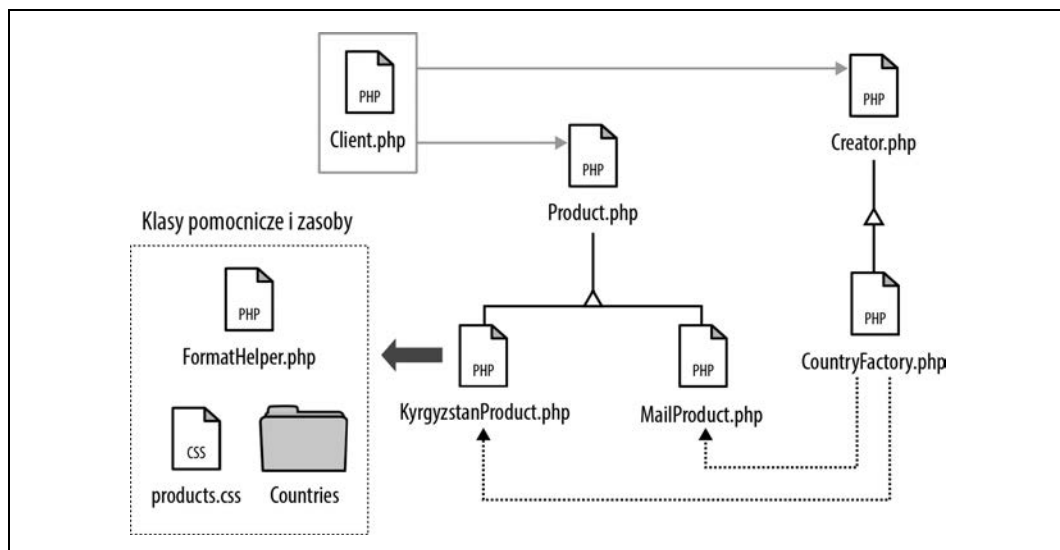
body
{
  font-family:Verdana, Geneva, sans-serif;
  font-size:12px;
}

```

Możesz nawet uznać plik ze stylami za klasę pomocniczą. W większych projektach możesz też zechcieć dodać jakieś skrypty JavaScript lub jQuery jako klasy pomocnicze.

Diagram plików

Nieformalny diagram, który uznałem za przydatny, to **diagram plików**. Składa się z rysunków plików i folderów wykorzystywanych we wzorcu projektowym wraz z relacjami, jakimi są połączone. Podobny jest do diagramu klas, ale przedstawia poszczególne pliki. Rysunek 5.6 przedstawia diagram plików dla wzorca Metoda Fabrykująca zaprezentowanego w tym rozdziale.



Rysunek 5.6. Pliki z klasami pomocniczymi i zasobami

Jak widać na diagramie, klasy pomocnicze i zasoby, mimo że nie są częścią wzorca projektowego, są wykorzystywane przez produkty. Jak wskazuje przerywana ramka, są oddzielone od samego wzorca, a pogrubiona strzałka świadczy o ich wykorzystaniu przez konkretne produkty.

Produkty się zmieniają, interfejs zestaw w spokoju

Jedną z największych zalet wzorców projektowych jest łatwość, z jaką można wprowadzać zmiany w klasach bez konieczności przerabiania pozostałych elementów programu. Sekretem tej łatwości jest zmiana zawartości przy zachowaniu tego samego interfejsu.

Działaniem, które uprościłoby klasy, jest przeniesienie tekstu na zewnątrz konkretnego produktu. Dzięki umieszczeniu notatek w pliku tekstowym, a następnie ładowaniu ich do zmiennych nie tylko zyskujemy na łatwości wprowadzania zmian w tekście, ale także klasa produktu stanie się bardziej czytelna.

W nowej klasie konkretnego produktu ładujemy tekst do zmiennej `$countryNow`. Zamiast więc dużej ilości tekstu konkretny produkt zawiera pięć linii kodu zapisującego notatkę w zmiennej. Poniższy listing przedstawia nowy produkt (*Mołdawia*) zawierający nowy sposób ładowania tekstu:

```
<?php
//TextProduct.php
include_once('FormatHelper.php');
include_once('Product.php');

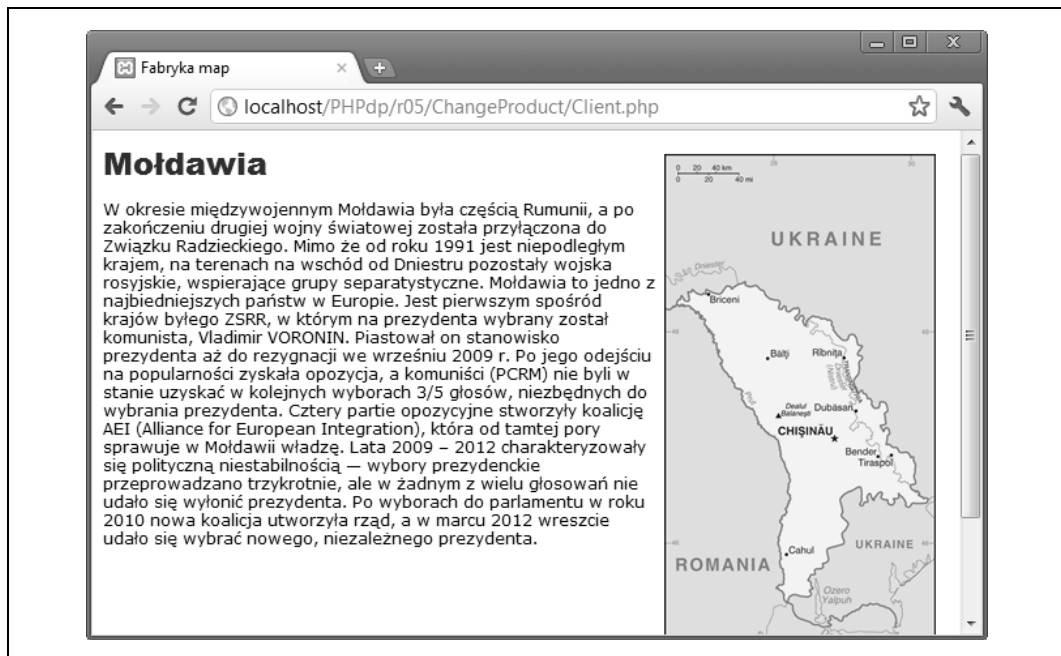
class MoldovaProduct implements Product
{
    private $mfgProduct;
    private $formatHelper;
    private $countryNow;

    public function getProperties()
    {
        //Ładuje tekst z pliku zewnętrznego
        $this->countryNow=file_get_contents("CountryWriteups/Moldova.txt");

        $this->formatHelper=new FormatHelper();
        $this->mfgProduct=$this->formatHelper->addTop();
        $this->mfgProduct.="<img src='Countries/Moldova.png' class='pixRight' width='208'
        ↪height='450'>";
        $this->mfgProduct .="<header>Mołdawia</header>";
        $this->mfgProduct .="<p>$this->countryNow</p>";
        $this->mfgProduct .=$this->formatHelper->closeUp();
        return $this->mfgProduct;
    }
}
?>
```

Jak widzisz, cały blok tekstu zniknął. To, co nie uległo zmianie — i jest bardzo ważne, aby nie uległo — to interfejs `getProperties()`. Dopóki interfejs pozostanie niezmieniony, wprowadzanie zmian we wzorcu nie spowoduje błędów w programie. Nawet dodanie nowych, zewnętrznych zasobów nie jest problemem. Rysunek 5.7 przedstawia wynik działania programu z nowym produktem.

Podczas gdy poziom złożoności Twoich produktów będzie wzrastał — będą wykonywały więcej zadań niż proste umieszczenie na stronie tekstu i grafiki — znaczenie interfejsu także będzie rosło. Na szczęście dbanie o pojedynczy interfejs jest o wiele łatwiejsze niż dbanie o większą liczbę klas i obiektów. Właśnie dlatego wzorec Metoda Fabrykująca pozwala uprościć sposób tworzenia obiektów — zachowuje pojedynczy interfejs.



Rysunek 5.7. Opis pochodzący z zasobów zewnętrznych

A

abstrakcja, 33, 269
adapter, 142
 kategorii klas, 117
 kategorii obiektów, 117
adnotacja z pseudokodem, 68, 69
adres URL, 278
agregacja, *Patrz:* relacja agregacji
akcesor, 107, 112, 180, 272, *Patrz też:* metoda pobierająca
Alexander Christopher, 13
algorytm, 73
 enkapsulujący, 217, 219, 222
 rodzina, 219, 224
 wyszukiwania, 239
arkusz stylów CSS, *Patrz:* CSS
asocjacja, *Patrz:* relacja asocjacji
autoreferencja, 179

B

Banda Czterech, *Patrz:* GoF, Gamma Erich, Helm Richard, Johnson Ralph, Vlissides John
baza danych MySQL, 115, 199, 227, 247, 278
bezpieczeństwo, 201, 204, 214, 227
biblioteka, 13
 jQuery Mobile, 285, 289, 290
 standardowa PHP, *Patrz:* SPL
błąd, 26
Booch Grady, 34, 68, 77, 78, 198

C

cecha, 36
Chelimsky David, 45
Chunqiao Zhang, 53
CMS, 278

CSS, 94, 128, 278, 282, 290
 wywołanie, 27

D

dane
 dostępowe, 200, 201
 połączeniowe, 247
 MySQL, 201, 202
 przetwarzanie, 217
 typowanie, 39
 wprowadzanie, 280, 281
 wrażliwe, 214
dekorator, 135, 137, 139, 144, 147, 155
delegacja, 61
diagram, 77, 78
 czynnościowy, 78
 interakcji, 76, 78
 klas, *Patrz:* klasa diagram obiektów, 75
 plików, 95
 przepływu, 77
 stanów, 78, 177, 189
 strukturalny, 78
dziedziczenie, 46, 47, 58, 61, 62, 72, 117, 126, 139
 płytkie, 47
 po klasie abstrakcyjnej, 55
 wielokrotne, 36, 117, 120, 122

E

element, *Patrz:* obiekt
enkapsulacja, 41, 42, 45, 62, 106, 202, 203, 217
 hasła, 210
 loginu, 210
 złamanie, 42

F

fabryka, 67, 83
firewall, 204
formularz HTML, 231, 249, 252
framework, 13, 64, 164, 170
Freeman Elizabeth, 219
Freeman Eric, 219
funkcja, 30
 clone, 99, 100, 102
 construct, 24, 25
 file_get_contents, 142
 include_once, 37
 makeConProto, 112
 md5, 207, 212, 214

G

Gamma Erich, 19, 39, 53, 62, 65, 117, 203
Gardner Lyza, 133
GoF, 53, 62, 65, 219, *Patrz też:* Gamma Erich,
 Helm Richard, Johnson Ralph, Vlissides John
Grigsby Jason, 133

H

hasło, 205, 209
Helm Richard, 19, 53, 65, 117
Hevery Miško, 103, 203
Hongwen Wang, 53

I

implementacja, *Patrz:* relacja implementacji
informacji ukrywanie, 41
interfejs, 34, 37, 38, 39, 41, 47, 54, 66, 72, 83, 96,
 204
 Creator, 83, 85
 dekoratora, 137
 dziedziczenie, 36
 implementacja, 37, 49, 55
 Observer, 267
 opisujący obiekt, 34
 prototypu, 107
 SplObjectStorage, 267, 269, 271
 SplObserver, 267, 269, 270, 294
 SplSubject, 267, 269, 270, 294
 Strategia, 50
 Subject, 267
 typ, 54

użytkownika, 41, 51, 150
 graficzny, 51
wzorca Obserwator, 267
znaczników HTML, 150

J

Johnson Ralph, 19, 53, 65, 68, 117, 160

K

kategoria
 klas, 62
 obiektów, 63
 zasięgu, 62
Kaye Jonathan, 19
klasa, 23, 62, 66, 204
 abstrakcyjna, 34, 35, 36, 39, 41, 66, 72, 161
 instancja, 54
 adapter, 119
 adaptowana, 143
 Client, 25, 27, 66, 81, 99, 105, 136, 149, 154,
 162, 173, 221, 246
 ConcreteSubject, 274
 Context, 175, 179, 180, 181, 219, 223
 delegacja, *Patrz:* delegacja
 diagram, 61, 66
 dziedzicząca, 46
 FormatHelper, 93
 instancja, 25, 30, 34, 67, 155
 interfejs, 35
 kategoria, *Patrz:* kategoria klas
 konstruktor, 24, 25
 metoda, *Patrz:* metoda
 nadrzędna, 36, 39, 46, 58
 obiekt, *Patrz:* klasa instancja
 obsługująca, 76
 odbierająca, 245
 odpowiedzialność, 24
 opakowująca, 142
 połączeniowa, 226
 pomocnicza, 94
 potomna, 36, 58, 160
 Proxy, 209, 212
 SplObjectStorage, 269, 271
 SplObserver, 269, 270
 SplSubject, 269, 270
 stanu, 181
 Subject, 274
 sygnatura, 35

właściwość, *Patrz:* właściwość
wprowadzanie zmian, 86, 87, 88, 89, 90
wysyłająca, 245
klient, 85, 111, 204, 277
SPL, 273
klonowanie, 99, 102, 105, 107, 112, 115
kod spaghetti, 30
kompozycja, 117, 120, 126
konstruktor, 102, 103
kontrola
odwrócenie, 164
przepływ, 163
kontroler, 51

L

logowanie, 204, 205, 209

M

Mao Zedong, 53
maszyna stanowa, 19, 78, 177, 178
metoda, 23, 25, 43, 55
abstrakcyjna, 35, 37, 73, 161, 270
algorithm, 221, 230
attach, 274
detach, 274
factoryMethod, 261
feedFactory, 261
getProperties, 84
pobierająca, 44, 45
setObservers, 272
szablonowa, 161
templateMethod, 159, 162
update, 270, 271
ustawiająca, 44, 45
model, 51
Model-View-Control, *Patrz:* MVC
modularyzacja, 23
mutator, 107, 112, 272, *Patrz:* metoda
ustawiająca
MVC, 51, 53, 267
Myers Tom, 22
myślenie abstrakcyjne, 33

N

Nakhimovsky Alexander, 22
Nixon Robin, 14, 21
notacja, 77

O

obiekt, 42, 62, 74
adapter, 119
diagram, *Patrz:* diagram obiektów
domenowy, 51
instancja, 99, 114
kategoria, *Patrz:* kategoria obiektów
odpowiedzialność, 164
powtórne wykorzystanie, 58
prezentacyjny, 51
produktu, *Patrz:* produkt
stan, 180
tworzenie, 62
złożoność, 58, 61
zmiana stanu, 175, 176, 267, 268
Object Management Group, *Patrz:* OMG
obserwator, 267, 269, 271, 272, 274, 276, 287
odpowiedź binarna, 22
OMG, 68
opakowywanie, 142, 143
Opdyke William, 160
operacja, 73
operator rozróżnienia zasięgu, 38, 202

P

podmiot, 267, 271
polecenie warunkowe, 176, 188, 219
polimorfizm, 47, 49, 50, 73, 84, 221
procedura, *Patrz:* funkcja
produkt, 83, 84
dodawanie, 90
graficzny, 89
tekstowy, 88
programowanie
dynamiczne, 22
obiektowe, 13, 20, 22, 29, 30, 31, 33, 45, 50,
77, 163, 294
proceduralne, 13, 19, 29, 30, 45, 163
sekwencyjne, 13, 19, 29, 77
przepływ kontroli, *Patrz:* kontrola przepływ

Q

Qing Jiang, 53

R

rekurencja, 179
relacja, 73
 agregacja, 71, 224
 asocjacji, 69, 71, 72, 74
 dziedziczenia, 72
 implementacji, 72, 73
 tworzenia, 74
 wielokrotna, 74
Rumbaugh James, 68

S

Singleton, 79
Sklar David, 21
skrypt wyzwalający, 223
słowo kluczowe
 clone, 102
 interface, 55
 static, 203
SPL, 267, 269, 294
stała, 38, 201
Standard PHP Library, *Patrz: SPL*
strefa czasowa, 257
subskrybent, 271
sygnatura, 36, 49, 55
system CMS, 278
szablon, 13, 64

T

tabela, 217
 MySQL, 279
 odpowiedzi, 247, 249
tablet, 233, 235, 269, 283, 290
tablica, 41, 230, 242
 pusta, 242
telefon komórkowy, 269, 289
test jednostkowy, 103
The Gang of Four, *Patrz: GoF, Gamma Erich, Helm Richard, Johnson Ralph, Vlissides John*
traits, *Patrz: cecha*
typ
 deklarowanie, 54
 implementacji, 39
 int, 41
 interfejsu, *Patrz: interfejs typ*
 podpowiadanie, 39, 41, 54, 56, 91, 112, 210
 skalarny, 41

 string, 41
 zwracany, 49
typowanie danych, *Patrz: dane typowanie*

U

uczestnik, 67
UI, *Patrz: interfejs użytkownika*
ukrywanie informacji, *Patrz: informacja*
 ukrywanie
UML, 61, 65, 68, 69, 77
Unified Modeling Language, *Patrz: UML*
urządzenie mobilne, 28, 31, 126, 128, 133, 235, 269, 285
użytkownik, 205
 nazwa, 205, 209

V

Vlissides John, 19, 53, 65, 117, 164

W

Wenyuan Yao, 53
widoczność, 36, 42
 private, 42
 protected, 43
 public, 44
widok, 51
właściwość, 23, 25, 35
 abstrakcyjna, 35
wzorzec
 Adapter, 117, 119, 120, 133, 142
 kategorii klas, 122
 kategorii obiektów, 122, 126
 wykorzystujący kompozycję, 126
Budowniczy, 79
czynnościowy, , 62, 64, 157
 komunikacja, 158
Dekorator, 117, 135, 136, 137, 142
Fabryka Abstrakcyjna, 74, 79, 81, 82, 83
Fasada, 117
Interpreter, 157
Iterator, 157
Kompozyt, 117
Kontekst, 20
kreatywny, 62, 64, 79
kryteria wyboru, 63
Łańcuch Odpowiedzialności, 75, 76, 157,
 199, 245, 246, 247, 251, 255, 256, 287

Mediator, 157
Metoda Fabrykująca, 66, 67, 74, 79, 165, 166, 167, 256, 261
Metoda Szablonowa, 157, 159, 160, 163, 164, 165, 166, 167, 174
 hak, 169, 171, 172
Most, 117
Obserwator, 157, 199, 267, 268, 269, 278, 283, 287, 294
 interfejs wbudowany, 267
Opakowanie, 129, 135, 142
Pamiętka, 157
Polecenie, 157
Prototyp, 79, 99, 100, 107, 115
Proxy, 117, 199, 204, 209, 214
 inteligentna referencja, 205
 podmiot, 205
 prawdziwy podmiot, 205, 212
 wirtualny, 204
 zabezpieczający, 205
 zdalny, 204
Pylek, 74, 117
Singleton, 203
Stan, 157, 158, 175, 176, 178, 183, 197, 198, 218, 219
Strategia, 20, 31, 50, 71, 77, 157, 158, 199, 217, 218, 219, 222, 227, 242
strukturalny, 62, 64, 117, 204
Wizytator, 157
z parametrami, 90, 91

Z

zasada
 Hollywood, 162, 163, 164, 174
 wyjątki, 169
 jednej odpowiedzialności, 24
 odwróconej struktury kontroli,
 Patrz: zasada Hollywood
 Przedszkola, 164
 wzorców projektowych
 druga, 58
 pierwsza, 54
zestaw zachowań, 219
złożoność, 61, *Patrz:* obiekt złożoność
zmienna, 35, 38, 54, 114, 155
 boolowska, 172, 173
 globalna, 203
 observers, 271
 prywatna, 202, 224, 271
 statyczna, 202, 203, 204

Ż

żądanie, 245, 255, 283
 inicjowanie, 246

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**

PHP. Wzorce projektowe



Ciągły rozwój języka PHP sprawia, że za jego pomocą można już tworzyć bardzo zaawansowane aplikacje. Wszyscy programiści wiedzą, że wraz ze wzrostem skomplikowania tworzonego oprogramowania konieczne staje się zastosowanie sprawdzonych, przemyślanych i przetestowanych rozwiązań. Takim zbiorem najlepszych metod rozwiązywania typowych problemów są wzorce projektowe. Zawitały one również do świata PHP!

Na rynku znajdziesz wiele książek poświęconych wzorcom projektowym, jednak dotyczą one zazwyczaj języków Java lub C. Ta wyjątkowa książka zajmuje się wzorcami w języku PHP. W trakcie lektury poznasz podstawowe założenia programowania obiektowego, wzorce czynnościowe, kreatywne i strukturalne. Przekonasz się też, jak bardzo wzrośnie jakość Twojego kodu, jeżeli do komunikacji z bazą danych MySQL zastosujesz wzorzec Proxy. Książka ta jest obowiązkową lekturą każdego programisty piszącego w języku PHP. Zobacz, jak dobry może być Twój kod!

Wzorce projektowe w PHP to:

- najlepsze rozwiązania typowych problemów
- sposób na poprawę jakości Twojego kodu
- łatwiejsza komunikacja z bazą danych MySQL
- mniej błędów w Twoich aplikacjach

Przekonaj się, jak tworzyć kod wysokiej jakości!

helion.pl
księgarnia
internetowa

Nr katalogowy: 15007



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 978-83-246-7455-8



Cena 49,00 zł