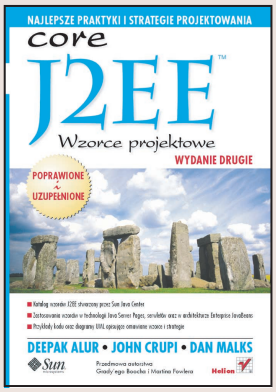
J2EE. Wzorce projektowe. Wydanie 2



Autor: Deepak Alur, John Crupi, Dan Malks
Tłumaczenie: Rafał Jońca
ISBN: 83-7361-344-7
Tytuł oryginału: Core J2EE Patterns. Best Practices and Design Strategies, 2nd Edition
Format: B5, stron: 518

Podstawowymi zagadnieniami opisywanymi w książce są wzorce, najlepsze techniki, strategie projektowe i sprawdzone rozwiązania wykorzystujące kluczowe technologie J2EE, czyli strony JSP, serwlety, komponenty EJB i interfejs JMS. Katalog wzorców J2EE zawiera 21 wzorców i o wiele więcej strategii, przy wykorzystaniu których powstają najlepsze rozwiązania programistyczne.

„J2EE. Wzorce projektowe. Wydanie drugie" zawiera opis następujących zagadnień:

- 21 wzorców projektowych J2EE — znane i w pełni sprawdzone oraz nowe wzorce zapewniające najlepsze rozwiązania dla aplikacji biznesowych
- strategie projektowe dla warstwy prezentacji, biznesowej i integracji
- opis zastosowania technologii JSP, EJB, JSM, usług sieciowych i serwletów
- przykłady niezalecanych rozwiązań stosujących technologię J2EE
- sposoby refaktoryzacji poprawiające istniejące projekty
- diagramy UML ilustrujące przedstawiane zagadnienia
- dużą liczbę przykładów zastosowania wzorców, strategii i refaktoryzacji

Deepak Alur jest architektem rozwiązań biznesowych Javy w Sun Java Center z 14-letnim doświadczeniem. Zajmuje się głównie architekturą, projektowaniem i implementacją dużych rozwiązań biznesowych z zastosowaniem technologii Java i J2EE.

John Crupi jest wybitnym inżynierem i szefem architektów Javy w Sun Java Center. Posiada 17-letnie doświadczenie w obliczeniach rozproszonych i zajmuje się przede wszystkim tworzeniem skalowalnych architektur J2EE wielokrotnego użytku.

Dan Malks jest głównym inżynierem w Sun Java Center. Posiada 16-letnie doświadczenie i zajmuje się technologiami obiektowymi oraz ich zastosowaniem w projektach biznesowych i usługach sieciowych. Pisze artykuły do czasopism branżowych, jest także współautorem książek o Javie, wzorcach i technologii J2EE.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

5

Omówienie wzorców J2EE

W tym rozdziale:

- Czym jest wzorzec?
- Podejście warstwowe.
- Wzorce J2EE.
- Wprowadzenie do katalogu.
- Związki między wzorcami J2EE.
- Związki z innymi znanymi wzorcami.
- Mapa wzorców.
- Podsumowanie.

Wzorce J2EE to zbiór rozwiązań dotyczących typowych problemów związanych z platformą J2EE. Są efektem wiedzy i doświadczeniu architektów z Sun Java Center, którzy brali udział w tworzeniu wielu udanych projektów J2EE. Sun Java Center jest organizacją konsultingową skupiającą się na tworzeniu rozwiązań opartych na technologii Java. Zajmuje się rozwiązaniami dla platformy J2EE od początku jej istnienia koncentrując się przede wszystkim na takich aspektach ogólnie pojętego QoS (*Quality of Service*) jak skalowalność, dostępność, wydajność, bezpieczeństwo, pewność i elastyczność.

Wzorce opisują typowe problemy napotykane podczas tworzenia aplikacji J2EE i wskazują możliwe rozwiązania. Rozwiązania te powstały dzięki doświadczeniu zdobytemu w trakcie prac nad wieloma różnymi projektami i wymianie informacji między programistami. Wzorce zawierają istotę tych rozwiązań, a poza tym zostały już zoptymalizowane przez ich użytkowników. Można powiedzieć inaczej: wzorce wydobywają najważniejsze aspekty problemu i oferują rozwiązania dobrze przemyślane i ujednolicone.

W niniejszej książce zajmujemy się wzorcami pod kątem J2EE, a w szczególności komponentami EJB, stronami JSP i serwletami. W trakcie pracy z klientami J2EE implementującymi różne komponenty poznaliśmy typowe problemy i znaleźliśmy dla nich optymalne rozwiązania. Opracowaliśmy też najlepsze praktyki wykorzystywane do tworzenia komponentów J2EE.

Opisane wzorce stosują te najlepsze praktyki. Przedstawiono je w taki sposób, by w prosty sposób móc wykorzystać je w dowolnej aplikacji uwzględniając konkretne wymagania. Wzorce w usystematyzowany sposób omawiają sprawdzone techniki. Dzięki nim łatwiej jest skorzystać ze zweryfikowanych rozwiązań. Innymi słowy, dzięki wykorzystaniu wzorców tworzenie systemów opartych na J2EE staje się prostsze i szybsze.

Czym jest wzorzec?

W rozdziale 1. przedstawiliśmy kilka definicji wzorca sformułowanych przez różnych ekspertów. Opisaliśmy także inne zagadnienia związane z wzorcami, na przykład zalety ich stosowania. Teraz jeszcze raz zajmiemy się tym tematem, ale pod kątem katalogu wzorców J2EE.

W rozdziale 1. pojawiła się następująca definicja wzorca: często stosowane *rozwiązanie* typowych *problemów* w pewnym *kontekście*.

Terminy kontekst, problem i rozwiązanie wymagają pewnego wyjaśnienia. Najpierw zajmijmy się kontekstem. Kontekst to środowisko, otoczenie, sytuacja lub pewne warunki, w których występuje problem. Czym jest problem? Problem to nierozstrzygnięte pytanie, czyli coś, co należy zbadać i rozwiązać. Zazwyczaj problem jest ograniczony kontekstem, w którym występuje. Rozwiązanie to odpowiedź na problem w danym kontekście pozwalająca go usunąć.

Czy jednak posiadanie rozwiązania problemu w pewnym kontekście oznacza, że mamy już wzorzec? Niekoniecznie. Ważnym elementem jest też powtarzalność stosowanego rozwiązania. Wzorzec jest użyteczny tylko wtedy, gdy może być stosowany wielokrotnie. Czy to wszystko? Niekoniecznie. Choć sama idea wzorca jest prosta, jego rzeczywista definicja nie jest już zadaniem łatwym.

Aby dowiedzieć się więcej o wzorcach, ich historii i dziedzinach zastosowania polecamy inne książki lub zasoby internetowe. W naszym katalogu wzorzec jest opisany zgodnie z jego główną charakterystyką: *problemem* i *rozwiązaniem*, ale wspominamy też o kilku innych aspektach, jak *siły* i *konsekwencje*. Poszczególne elementy szablonu wzorców omawia podrozdział „Szablon wzorców”.

Identyfikacja wzorca

W Sun Java Center zajmowaliśmy się wieloma projektami J2EE i wkrótce zauważyliśmy, iż niektóre problemy pojawiają się w prawie wszystkich projektach. Co ważniejsze, rozwiązania tych problemów także były podobne. Choć strategie implementacyjne się różniły, ogólny zarys rozwiązania był taki sam. Opiszemy teraz pokrótce, jak odkrywaliśmy wzorców.

Gdy zauważamy, że problem i jego rozwiązanie pojawiają się kilkakrotnie, staramy się zidentyfikować i udokumentować jego charakterystykę, stosując szablon wzorców. Najpierw rozważamy tę początkową dokumentację jako kandydata na wzorzec. Nie dodajemy kandydatów do katalogu wzorców aż do momentu ich wielokrotnego wystąpienia w wielu projektach. Sprawdzamy także, czy wzorzec ten nie został już zastosowany we wcześniejszych projektach.

Jako proces walidacji wzorca stosujemy *zasadę trzech* (*Rule of Three*); pod taką nazwą znana jest w społeczności programistów. Zasada ta określa kandydata na wzorzec może pojawić się w katalogu. Zgodnie z nią kandydat staje się pełnoprawnym wzorcem, gdy został zidentyfikowany w co najmniej trzech różnych systemach. Oczywiście zasada ta jest elastyczna, ale pomaga w identyfikacji wzorców.

Często podobne rozwiązania mogą dotyczyć jednego wzorca. W trakcie prac nad wzorcem należy zastanowić się nad tym, w jaki sposób najlepiej z jego pomocą przekazać rozwiązanie. Czasem wystarczy nadanie odpowiedniej nazwy, by wszyscy programiści od razu wiedzieli, o co chodzi. W takiej sytuacji warto spróbować udokumentować dwa podobne rozwiązania jako dwa różne wzorce. Z drugiej strony lepsze może się okazać przekazanie rozwiązania jako kombinacji wzorca i strategii.

Wzorce a strategie

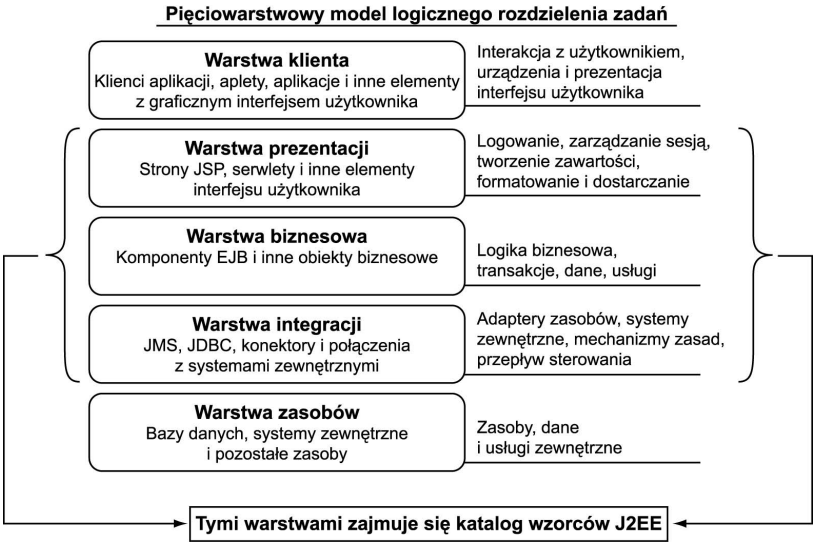
Gdy rozpoczęliśmy dokumentowanie wzorców J2EE, staraliśmy się używać dość wysokiego poziomu abstrakcji. Jednocześnie każdy wzorzec zawierał różne strategie dotyczące szczegółów implementacji. Dzięki strategiom wzorzec dokumentował rozwiązanie na różnym poziomie abstrakcji. Niektóre z tych strategii mogłyby stanowić osobne wzorce, ale sądziliśmy, że aktualna struktura szablonu najlepiej obrazuje relacje pomiędzy strategiami a wzorcami, w których się zawierają.

Choć cały czas toczyły się debaty dotyczące zamiany tych strategii na wzorce, ciągle opieraliśmy się takim rozwiązaniom, gdyż uważaliśmy istniejącą dokumentację za przejrzystą. Oto kilka spostrzeżeń dotyczących relacji pomiędzy wzorcami a strategiami:

- Wzorce istnieją na wyższym poziomie abstrakcji niż strategie.
- Strategie stanowią najbardziej zalecane lub najczęstsze implementacje wzorców.
- Strategie umożliwiają rozszerzenie wzorca. Programiści odkrywają nowe sposoby implementacji wzorca, tworząc nowe strategie dla dobrze znanych wzorców.
- Strategie ułatwiają komunikację, tworząc nazwy dla bardziej szczegółowych aspektów rozwiązania.

Podejście warstwowe

Ponieważ katalog opisuje wzorce pomagające budować aplikacje działające na platformie J2EE, a platforma ta jest systemem wielowarstwowym, przyglądamy się systemowi pod kątem *warstw*. Warstwy stanowią logiczny podział zadań w systemie. Poszczególne warstwy mają zdefiniowane właściwe zakresy odpowiedzialności. Warstwy są od siebie logicznie oddzielone. Każda z warstw jest luźno związana z warstwami sąsiednimi. Cały system można przedstawić jako stos warstw (patrz rysunek 5.1).



Rysunek 5.1. *Podejście warstwowe*

Warstwa klienta

Warstwa ta reprezentuje wszystkie urządzenia albo systemy klienckie mające dostęp do systemu lub aplikacji. Klientem może być przeglądarka internetowa, aplikacja Javy lub inna, aplet Javy, telefon komórkowy z WAP, aplikacja sieciowa albo inne urządzenie, dla którego jeszcze nie ma nazwy. Może to być nawet proces wsadowy.

Warstwa prezentacji

Warstwa ta zawiera całą logikę prezentacyjną wymaganą do obsługi klientów używających systemu. Warstwa ta odbiera żądania klienta, zapewnia system logowania, zarządza sesją, steruje dostępem do usług biznesowych, tworzy i dostarcza odpowiedzi do klienta. W tej warstwie znajdują się serwlety i strony JSP. Pamiętajmy, że choć nie są one elementami interfejsu użytkownika, to takowe elementy tworzą.

Warstwa biznesowa

Warstwa biznesowa udostępnia usługi biznesowe wymagane przez klientów aplikacji. Warstwa ta zawiera dane i logikę biznesową. Zazwyczaj wewnątrz tej warstwy wykonywana jest największa liczba zadań związanych z przetwarzaniem biznesowym. Możliwe, że z powodu istnienia systemów zewnętrznych część przetwarzania będzie się odbywała na warstwie zasobów. Preferowanym sposobem implementacji obiektów biznesowych są komponenty EJB.

Warstwa integracji

Ta warstwa odpowiada za komunikację z zewnętrznymi systemami i źródłami danych, na przykład bazami danych i aplikacjami zewnętrznymi (*legacy applications*). Warstwa biznesowa wykorzystuje warstwę integracji za każdym razem, gdy obiekt biznesowy potrzebuje

danych lub zasobów znajdujących się w warstwie zasobów. Komponenty z tej warstwy używają JDBC, technologii konektorów J2EE (JCA) lub innego oprogramowania współpracującego z warstwą zasobów.

Warstwa zasobów

Jest to warstwa zawierająca dane biznesowe i zewnętrzne zasoby, takie jak komputery mainframe lub systemy zewnętrzne (*legacy systems*), systemy B2B albo usługi w rodzaju autoryzacji kart kredytowych.

Wzorce J2EE

Stosujemy podejście warstwowe, aby podzielić wzorce J2EE zgodnie z ich przeznaczeniem. Wzorce warstwy prezentacji są związane z serwletami i technologią JSP. Wzorce warstwy biznesowej zawierają wzorce związane z technologią EJB. Wzorce warstwy integracji dotyczą JMS i JDBC (patrz rysunek 5.2 w dalszej części rozdziału).

Wzorce warstwy prezentacji

Tabela 5.1 zawiera wzorce warstwy prezentacji wraz z krótkim omówieniem każdego z nich.

Tabela 5.1. *Wzorce warstwy prezentacji*

Nazwa wzorca	Opis
Intercepting Filter	Zajmuje się przetworzeniem żądania klienta po jego potrzymaniu i przy wysłaniu odpowiedzi.
Front Controller	Zapewnia scentralizowany kontroler zarządzający obsługą żądań.
Context Object	Hermetyzuje stan aplikacji w sposób niezależny od protokołu, w celu łatwego wykorzystania go w różnych warstwach aplikacji.
Application Controller	Centralizuje zarządzanie widokiem i akcjami aplikacji.
View Helper	Zawiera logikę niezwiązaną z formatowaniem prezentacji w komponentach pomocniczych.
Composite View	Tworzy złożony widok z wielu komponentów składowych.
Service to Worker	Łączy komponent `Dispatcher` z wzorcami *Front Controller* i *View Helper*.
Dispatcher View	Łączy komponent `Dispatcher` z wzorcami *Front Controller* i *View Helper*, wstrzymując wiele zadań do czasu przetworzenia widoku.

Wzorce warstwy biznesowej

Tabela 5.2 wymienia wzorce warstwy biznesowej wraz z krótkim omówieniem każdego z nich.

Tabela 5.2. *Wzorce warstwy biznesowej*

Nazwa wzorca	Opis
Business Delegate	Hermetyzuje dostęp do usług biznesowych.
Service Locator	Hermetyzuje kod wyszukiwania usługi i komponentów.
Session Façade	Hermetyzuje komponenty warstwy biznesowej i udostępnia klientom dobrze zdefiniowany zestaw usług biznesowych.
Application Service	Centralizuje i łączy podstawowe mechanizmy aplikacji w celu zapewnienia jednolitej warstwy usług.
Business Object	Oddziela logikę i dane biznesowe, tworząc podstawę modelu obiektowego aplikacji.
Composite Entity	Implementuje trwałe obiekty biznesowe, stosując lokalne komponenty Entity i zwykłe obiekty Javy (POJOs).
Transfer Object	Przenosi dane między warstwami.
Transfer Object Assembler	Tworzy obiekty *Transfer Object*, korzystając z wielu źródeł danych.
Value List Handler	Obsługuje wyszukiwanie, buforuje wyniki w pamięci podręcznej i umożliwia przeglądanie i wybieranie elementów z listy wartości.

Wzorce warstwy integracji

Tabela 5.3 wymienia wzorce warstwy integracji wraz z krótkim omówieniem każdego z nich.

Tabela 5.3. *Wzorce warstwy integracji*

Nazwa wzorca	Opis
Data Acces Object	Hermetyzuje dostęp do źródeł danych ukrywając szczegóły implementacji.
Service Activator	Odbiera komunikaty i w sposób asynchroniczny wywołuje usługi biznesowe.
Domain Store	Udostępnia mechanizmy utrwalania i odtwarzania obiektów biznesowych.
Web Service Broker	Udostępnia usługi, używając XML i protokołów sieciowych.

Wprowadzenie do katalogu

Aby pomóc w zrozumieniu i użyciu wzorców J2EE z katalogu, proponujemy zapoznanie się z tym podrozdziałem przed zagłębieniem się w szczegóły poszczególnych wzorców. Wyjaśnimy terminologię stosowaną we wzorcach, sposób stosowania diagramów UML, stereotypów i szablonu wzorców. W skrócie wyjaśnimy, w jaki sposób używać wzorców. Przedstawimy też mapę ułatwiającą poruszanie się po wzorcach z katalogu.

Terminologia

Osoby zatrudnione w firmach informatycznych, a w szczególności programiści tworzący systemy oparte na technologii Java, używają specyficznych terminów i akronimów. Choć wielu czytelników zapewne spotkało się z tymi pojęciami, niektóre z nich są wykorzystywane w różnych kontekstach. Aby uniknąć nieporozumień i zachować spójność, w tabeli 5.4 podajemy definicje wykorzystywanych terminów.

Tabela 5.4. *Terminologia*

Termin	Opis lub definicja	Używany
BMP (*Bean-managed Persistence*)	Strategia dotycząca komponentów Entity, w których programista implementuje logikę odpowiedzialną za trwałość komponentów Entity.	we wzorcach warstwy biznesowej
CMP (*Container-managed Persistence*)	Strategia dotycząca komponentów Entity, w których zarządzanie trwałością komponentów Entity jest zadaniem kontenera.	we wzorcach warstwy biznesowej
dyspozytor (*Dispatcher*)	Do zadań kontrolera należy między innymi przekazywanie żądań klientów do odpowiednich widoków. Serwlety korzystają w tym celu ze standardowej klasy `RequestDispatcher` Mechanizm ten można umieścić w osobnym komponencie, zwanym dyspozytorem.	we wzorcach *Dispatcher View* i *Service to Worker*
EJB	Komponent Enterprise JavaBeans; może to być instancja komponentu sesyjnego lub Entity. Gdy stosujemy ten skrót, oznacza to, iż jest to komponent sesyjny lub Entity.	w wielu miejscach w niniejszej książce
fabryka (abstrakcyjna lub metoda fabryki)	Wzorzec opisany w książce GoF dotyczący tworzenia obiektów lub rodzin obiektów.	we wzorcach warstwy biznesowej i we wzorcu *Data Access Object*.
fasada	Wzorzec dotyczący ukrywania złożoności, opisany w książce GoF.	we wzorcu *Session Façade*
GoF	Skrót od określenia Gang of Four (Banda Czworga), odnoszącego się do autorów popularnej książki o wzorcach projektowych (*Design Patterns: Elements of Reusable Object-Oriented Software*, autorzy: Erich Gamma, Richard Helm, Ralph Johnson i John Vlissides [GoF]).	w wielu miejscach w niniejszej książce
iterator	Wzorzec zapewniający dostęp do poszczególnych elementów kolekcji, opisany w książce GoF.	we wzorcu *Value List Handler*
komponent sesyjny (*Session Bean*)	Odnosi się do komponentu sesyjnego stanowego lub bezstanowego. Może się także odnosić w ogólności do interfejsu bazowego (*home*) i zdalnego (*remote*) oraz właściwej implementacji komponentu.	we wzorcach warstwy biznesowej
kontroler (*Controller*)	Współdziała z klientem, steruje i zarządza obsługą żądań.	we wzorcach warstwy prezentacji i warstwy biznesowej

Tabela 5.4. *Terminologia (ciąg dalszy)*

Termin	Opis lub definicja	Używany
model	Fizyczna albo logiczna reprezentacja systemu lub podsystemu.	we wzorcach warstwy biznesowej i warstwy prezentacji
obiekt dostępu do danych (DAO)	Obiekt ukrywający szczegóły implementacji dostępu do źródeł danych i systemów zewnętrznych.	we wzorcach warstwy biznesowej i warstwy integracji
obiekt niezależny	Obiekt, który może istnieć niezależnie od innych i zarządza cyklem życia swych obiektów zależnych.	we wzorcu *Composite Entity*
obiekt transferowy (*Transfer Object*)	Serializowalny obiekt Javy (POJO) używany do przesyłania danych z jednego obiektu (warstwy) do innego. Nie zawiera żadnych metod biznesowych.	we wzorcach warstwy biznesowej
obiekt zależny	Obiekt, który nie może istnieć samodzielnie. Jego cyklem życia zarządza inny obiekt.	we wzorcach *Business Delegate* i *Composite Entity*
pomocnik (*Helper*)	Wykorzystywany przez kontroler i/lub widok. Na przykład kontroler lub widok może skorzystać z obiektu pomocnika w celu: pobrania zawartości, walidacji, zapisu modelu lub dostosowania go w celu wyświetlenia.	we wzorcach warstwy prezentacji i we wzorcu *Business Delegate*
proxy	Wzorzec w którym jeden obiekt ukrywa inny i steruje dostępem do niego, opisany w książce GoF.	w wielu miejscach niniejszej książki
przedstawiciel (*Delegate*)	Obiekt zastępujący inny — stanowi dla niego warstwę pośredniczącą. Przedstawiciel posiada cechy pośrednika oraz fasady.	we wzorcu *Business Delegate* i w wielu innych wzorcach
singleton	Wzorzec zapewniający tylko jedną instancję obiektu, opisany w książce GoF.	w wielu miejscach w niniejszej książce
skryptlet Javy	Logika aplikacji osadzona bezpośrednio na stronie JSP.	we wzorcach warstwy prezentacji
szablon	Szablon tekstowy odnosi się do stałego tekstu w widoku JSP. Poza tym szablon może dotyczyć konkretnego układu komponentów tworzących widok.	we wzorcach warstwy prezentacji
trwały magazyn (*Persistent Store*)	Reprezentuje trwały system przechowywania danych, np. RDBMS, ODBMS, systemy plików itp.	we wzorcach warstwy biznesowej i warstwy integracji
widok (*View*)	Widok odpowiada za graficzną i tekstową część wyświetlanego interfejsu. Współpracuje z pomocnikami, aby uzyskać dane potrzebne do utworzenia zawartości. Wykorzystuje także obiekty pomocników w celu wykonania dodatkowych zadań.	we wzorcach warstwy prezentacji
złożenie (*Composite*)	Obiekt złożony zawierający inne obiekty. Związany z wzorcem *Composite* z książki GoF (patrz dalsza część tabeli).	we wzorcach *Composite View* i *Composite Entity*

Stosowanie języka UML

W katalogu wzorców intensywnie korzystamy z diagramów UML, a w szczególności z następujących typów diagramów:

- *Diagramy klas* — używamy diagramów klas do przedstawienia struktury rozwiązania i struktury strategii implementacji. Stanowią statyczny obraz rozwiązania.
- *Diagramy przebiegu* (inaczej sekwencji lub interakcji) — te diagramy służą do przedstawiania wzajemnych oddziaływań elementów rozwiązania lub strategii. Stanowią dynamiczny obraz rozwiązania.
- *Stereotypy* — używamy stereotypów, aby wskazać różne typy obiektów i ich role w diagramach klas i interakcji. Lista stereotypów i ich znaczeń znajduje się w tabeli 5.5.

Tabela 5.5. *Stereotypy UML*

Stereotyp	Znaczenie
EJB	Reprezentuje komponent Enterprise JavaBean; związany jest z obiektem biznesowym. Rolę tą pełni zazwyczaj komponent sesyjny lub Entity.
komponent sesyjny	Reprezentuje komponent sesyjny jako całość bez określania jego interfejsów ani implementacji.
komponent Entity	Reprezentuje komponent Entity jako całość bez określania jego interfejsów, implementacji ani klucza głównego.
widok	Widok reprezentuje i wyświetla informacje przekazywane klientowi.
JSP	Strona JSP; widok jest zazwyczaj zaimplementowany jako strona JSP.
serwlet	Serwlet Javy; kontroler jest zazwyczaj zaimplementowany jako serwlet.
singleton	Klasa posiadająca tylko jedną instancję zgodnie z wzorcem *Singleton*.
własny znacznik (*Custom Tag*)	Własne znaczniki JSP (podobnie jak komponenty JavaBean) wykorzystuje się do implementacji obiektów pomocniczych. Obiekt pomocniczy jest odpowiedzialny za takie zadania, jak zbieranie informacji wymaganych przez widok, lub za dostosowanie modelu danych do wyświetlenia. Pomocnik może po prostu przekazać dane widokowi w postaci oryginalnej lub dokonać ich formatowania do postaci odpowiedniej do wyświetlenia jako strona WWW.

Wszystkie wzorce z katalogu zawierają diagram klas przedstawiający strukturę rozwiązania i diagram sekwencji prezentujący interakcje występujące we wzorcu. Poza tym wzorce zawierające strategie stosują osobne diagramy dla poszczególnych strategii.

Książki zawierające więcej informacji o języku UML wymieniono w bibliografii.

Stereotypy UML

W trakcie zapoznawania się z wzorcami i ich diagramami z pewnością natkniemy się na stereotypy. Stereotypy są terminami stosowanymi przez projektantów i architektów. Wykorzystujemy te terminy, by precyzyjnie i prosto zaprezentować diagramy. Część z tych stereotypów odwołuje się do przedstawionej wcześniej terminologii. Jako stereotypy uznajemy także nazwy wzorców i spełnianych przez nie ról, co pomaga w ich wyjaśnieniu.

Szablon wzorców

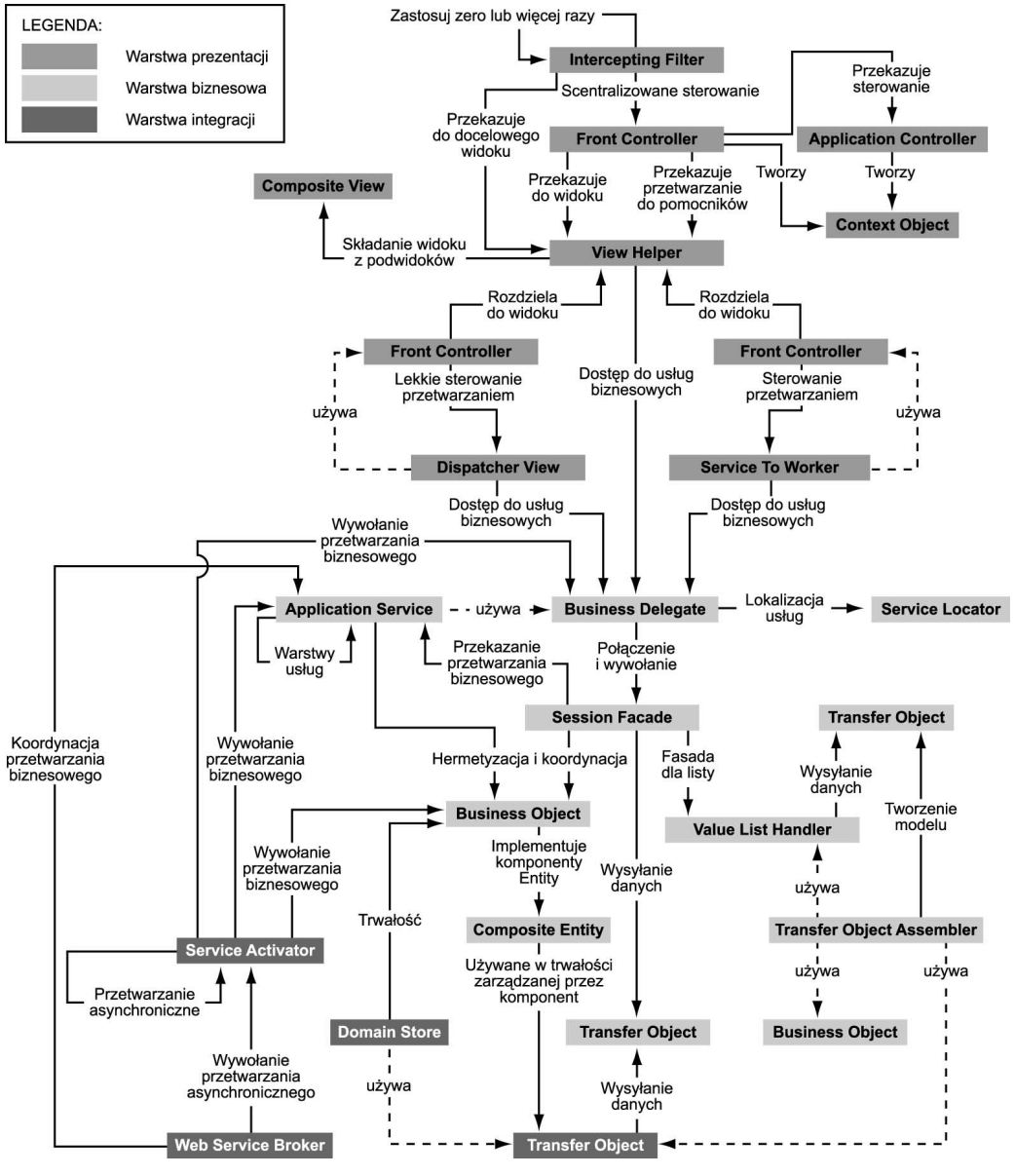
Wzorce J2EE są zdefiniowane zgodnie z szablonem wzorców. Szablon wzorców składa się z podrozdziałów prezentujących różne elementy danego wzorca. Warto zauważyć, że staraliśmy się nadać wszystkim wzorcom opisowe nazwy. Choć trudno jest zawrzeć całą istotę wzorca w jego nazwie, wydaje nam się, że udało się spełnić to zadanie. Podobnie jak nazwy w rzeczywistym świecie, nazwa wzorca informuje czytelnika, jak z niego korzystać.

Stosujemy szablon wzorców składający się z następujących części:

- *Problem* — opisuje zagadnienie projektowe, które musi rozstrzygnąć projektant.
- *Siły* — wymienia powody lub motywy wpływające na problem i jego rozwiązanie. Lista sił zawiera powody, dla których warto zastosować dany wzorzec.
- *Rozwiązanie* — pokrótce omawia ideę rozwiązania i bardziej szczegółowo jego konkretne elementy. Ta część składa się z dwóch podrozdziałów:
  - *Struktura* — za pomocą diagramów UML przedstawiana jest podstawowa struktura rozwiązania. Diagramy sekwencji przedstawiają dynamiczne elementy rozwiązania. Pojawia się szczegółowy opis elementów oraz występujących zależności.
  - *Strategie* — opisuje różne sposoby implementacji wzorca. Podrozdział „Wzorce a strategie" wyjaśnia potrzebę stosowania strategii. Jeżeli strategię można pokazać na przykładzie kodu, przedstawiony zostaje odpowiedni fragment. Jeżeli fragment kodu jest długi, umieszczamy go w sekcji *Przykładowy kod*.
- *Konsekwencje* — omawia wady i zalety danego wzorca. Ogólnie ta część skupia się na wynikach zastosowania wzorca lub strategii i konsekwencjach jego stosowania dla aplikacji.
- *Przykładowy kod* — ta część zawiera przykładowe implementacje i listingi dla wzorca lub strategii. Część jest opcjonalna, jeżeli cały kod został wcześniej opisany podczas omawiania strategii.
- *Powiązane wzorce* — zawiera listę innych wzorców z katalogu J2EE lub z innych źródeł, na przykład wzorce projektowe GoF. Poza nazwą wzorca podajemy krótki opis jego związku z omawianym wzorcem.

Związki między wzorcami J2EE

Na pewnym etapie odkrywania wzorców projektanci i architekci zauważyli brak dobrego zrozumienia zasad łączenia wzorców w celu uzyskania większych rozwiązań. Rozwiązujemy ten problem, podając graficzną reprezentację wzorców i związków między nimi. Diagram nosi nazwę diagramu związków wzorców J2EE i jest przedstawiony na rysunku 5.2. W epilogu przedstawiamy przykład użycia połączonych wzorców w celu realizacji przykładowych scenariuszy.



Rysunek 5.2. *Związki wzorców J2EE*

Poszczególne wzorce posiadają własny kontekst, problem i rozwiązanie konkretnych zagadnień. Warto się jednak najpierw ogarnąć i zrozumieć pełny obraz wzorców w celu lepszego ich zrozumienia i wykorzystania.

Pamiętajmy o zdaniu Christophera Alexandra z rozdziału 1., w którym twierdzi, iż wzorzec nie istnieje w izolacji i musi być wspierany przez inne wzorce, by nabrał odpowiedniego znaczenia. W zasadzie wszystkie wzorce z katalogu są powiązane z innymi wzorcami. Zrozumienie tych związków w trakcie projektowania rozwiązania jest pomocne, gdyż:

- Umożliwia zastanowienie się, jakie inne problemy mogą się pojawić, jeśli do rozwiązania problemu zastosujemy przedstawiony wzorzec. Jest to efekt domina: jakie problemy spowoduje modyfikacja architektury w celu wprowadzenia wzorca? Warto zidentyfikować takie problemy przed rozpoczęciem pisania kodu.
- Umożliwia zapoznanie się ze związkami między wzorcami w celu poznania alternatywnych rozwiązań. Po identyfikacji problemów sprawdzamy związki między wzorcami i zastanawiamy się nad rozwiązaniami alternatywnymi. Być może nowe problemy można rozwiązać, wybierając inny wzorzec lub też stosując wybrany wzorzec w połączeniu z innym.

Rysunek 5.2 przedstawia związki między wzorcami.

Wzorzec *Intercepting Filter* przechwytuje nadchodzące żądania klientów, wysyła odpowiedzi i zajmuje się filtrowaniem. Filtry można dodawać i usuwać w dowolnym momencie (filtry deklarowane są w deskryptorze wdrożenia), stosując różne ich kombinacje. Po zakończeniu przetwarzania wstępnego lub końcowego ostatni filtr z grupy przekazuje sterowanie do właściwego obiektu docelowego. W przypadku żądań jest to najczęściej kontroler, ale może to być także widok.

Front Controller zawiera logikę przetwarzania występującą wewnątrz warstwy prezentacji, która w innym przypadku zostałaby błędnie umieszczona w widoku. Kontroler obsługuje żądania i zarządza pobieraniem zawartości, bezpieczeństwem, widokiem, nawigacją, wykorzystując obiekt dyspozytora w celu wybrania odpowiedniego widoku.

Wzorzec *Application Controller* centralizuje sterowanie, pobieranie i wywoływanie widoku oraz wykonywanie poleceń. Podczas gdy wzorzec *Front Controller* działa jak centralny punkt dostępowy dla nadchodzących żądań, *Application Controller* jest odpowiedzialny za identyfikację i wywoływanie poleceń aplikacji oraz wybór odpowiedniego widoku.

Wzorzec *Context Object* hermetyzuje stan aplikacji w sposób niezależny od protokołu, aby mógł być bez przeszkód wymieniany między różnymi elementami aplikacji. Ułatwia to testowanie, gdyż zmniejsza się liczbę zależności od konkretnego kontenera.

Wzorzec *View Helper* służy oddzieleniu kodu związanego z formatowaniem od reszty logiki biznesowej. Sugeruje stosowanie komponentów pomocniczych do wykonywania zadań związanych z pobieraniem zawartości, jej walidacją oraz dostosowaniem do potrzeb wyświetlania. Komponent widoku zawiera wtedy tylko elementy związane z tworzeniem prezentacji. Komponenty pomocnicze zazwyczaj komunikują się z usługami biznesowymi przez obiekty *Business Deletate* lub *Application Service*, natomiast sam widok może składać się z wielu mniejszych komponentów tworzących ogólny szablon.

Wzorzec *Composite View* dotyczy tworzenia widoku z wielu elementów jednostkowych. Mniejsze widoki (statyczne lub dynamiczne) łączy się w celu uzyskania jednego szablonu. Innymi przykładami połączonych wzorców są wzorce *Service to Worker* i *Dispatcher View*. Oba wzorce posiadają podobną strukturę, w skład której wchodzi kontroler współpracujący z dyspozytorem, widokami oraz obiektami pomocniczymi. Oba wzorce pełnią podobne role, ale różnią się wewnętrznym podziałem zadań. W odróżnieniu od wzorca *Service to Worker*, wzorzec *Dispatcher View* wstrzymuje przetwarzanie biznesowe aż do zakończenia przetwarzania widoku.

Wzorzec *Business Delegate* zmniejsza zależności między oddzielonymi warstwami i stanowi punkt dostępu do zdalnych usług warstwy biznesowej. Wzorzec ten może również stosować buforowanie danych w celu zwiększenia wydajności. *Business Delegate* istnieje w relacji z wzorcem *Session Façade* ukrywając szczegóły związane z korzystaniem z fasady. Wzorzec *Application Service* wykorzystuje wzorzec *Business Delegate* w celu dostępu do fasady.

Wzorzec *Service Locator* ukrywa szczegóły implementacji mechanizmów wyszukiwania komponentów oraz usług biznesowych. Wzorzec *Business Delegate* używa go w celu połączenia z odpowiednią *Session Façade*. Pozostali klienci, którzy muszą się połączyć z *Session Façade* lub innymi usługami warstwy biznesowej, również korzystają z wzorca *Service Locator*.

Wzorzec *Session Façade* udostępnia klientom usługi biznesowe, ukrywając złożoność implementacji tych usług. Wzorzec ten może korzystać z obiektów *Business Object* oraz odwoływać się do różnych implementacji wzorca *Application Service*. Może także wykorzystywać wzorzec *Value List Handler*.

Wzorzec *Application Service* zawiera wybrane mechanizmy udostępniając je w jednolitej formie usługom warstwy biznesowej. Wzorzec ten może wchodzić w interakcję z innymi usługami lub obiektami biznesowymi. Wzorzec może wywoływać inne implementacje *Application Service*, tworząc tym samym jedną z warstw usług aplikacji.

Wzorzec *Business Object* tworzy model domenowy aplikacji przy wykorzystaniu modelu obiektowego. Wzorzec ten oddziela dane i logikę biznesową, tworząc osobną warstwę aplikacji. Wzorzec zazwyczaj reprezentuje obiekty trwałe, które mogą być utrwalane i odtwarzane przy użyciu wzorca *Domain Store*.

Wzorzec *Composite Entity* implementuje obiekty *Business Object*, stosując lokalne komponenty Entity i zwykłe obiekty Javy (POJO). Wzorzec ten korzysta z wzorca *Data Access Object* gdy stosuje się trwałość zarządzaną przez komponenty (BMP).

Wzorzec *Transfer Object* pozwala na łatwą wymianę danych pomiędzy warstwami. Pozwala zredukować obciążenie sieci i zminimalizować liczbę wywołań między warstwami.

Wzorzec *Transfer Object Assembler* tworzy złożone obiekty *Transfer Object*, korzystając z wielu źródeł danych. Źródła te mogą być komponentami EJB, obiektami DAO lub zwykłymi obiektami Javy. Wzorzec jest najbardziej użyteczny, gdy klient musi pobrać dane właściwe dla modelu aplikacji lub jego części.

Wzorzec *Value List Handler* używa wzorca iteratora [GoF] w celu wykonania i przetworzenia wyników zapytań do baz danych. Wzorzec buforuje wyniki zapytań i w razie potrzeby przesyła klientowi podzbiór wyników. Stosując ten wzorzec, unikamy narzutu związanego z wyszukiwaniem dużej liczby komponentów Entity. Wzorzec używa obiektów DAO w celu wykonania zapytań i pobrania wyników z trwałego magazynu danych.

Wzorzec *Data Access Object* pozwala na oddzielenie warstwy biznesowej i zasobów. Ukrywa całą logikę dostępu do danych związaną z tworzeniem, pobieraniem, usuwaniem i aktualizacją danych z trwałego magazynu. Wzorzec do wysyłania i odbierania danych używa obiektów *Transfer Object*.

Wzorzec *Service Activator* pozwala na asynchroniczne przetwarzanie z wykorzystaniem JMS. Wzorzec może korzystać z wzorców: *Application Service*, *Session Façade* i *Business Object*. Można zastosować kilka obiektów *Service Activator*, aby zapewnić współbieżne przetwarzanie asynchroniczne dla zadań długoterminowych.

Wzorzec *Domain Store* stanowi mechanizm zapewniania trwałości dla modelu obiektowego. Stosuje i łączy kilka innych wzorców, między innymi *Data Access Object*.

Wzorzec *Web Service Broker* udostępnia jedną lub kilka usług aplikacji zewnętrznym klientom jako *web services*, stosując język XML i standardowe protokoły. Wzorzec współpracuje z wzorcami *Application Service* i *Session Façade*. Wykorzystuje jeden lub kilka wzorców *Service Activator* do przeprowadzenia asynchronicznego przetwarzania żądania.

Związki z innymi znanymi wzorcami

Istnieje znaczna ilość dokumentacji związanej z wzorcami projektowymi. Wzorce w poszczególnych książkach opisywane są na różnym poziomie abstrakcji. Istnieją wzorce architektury, projektowe, analizy i programowania. Najbardziej znaną książką dotyczącą wzorców jest *Design Patterns: Elements of Reusable Object-Oriented Software* [GoF]. Zawarte w niej wzorce stanowią rozwiązania dotyczące projektowania obiektowego. Odnosimy się także do wzorców z książki *Patterns of Enterprise Application Architecture* [PEAA] autorstwa Martina Fowlera.

Nasz katalog wzorców zawiera wzorce opisujące strukturę aplikacji i elementy projektu. Wspólnym mianownikiem jest opis wzorców pod kątem platformy J2EE. W pewnych przypadkach wzorce z katalogu bazują na wzorcach istniejących w literaturze lub są z nimi związane. W takiej sytuacji nazwa powszechnie przyjętego wzorca stanowi część nazwy wzorca J2EE lub też nazwę tę podajemy w podrozdziałach „Powiązane wzorce". Na przykład pewne wzorce bazują na wzorcach z książki GoF, ale są omawiane w kontekście platformy J2EE. W takich przypadkach nazwa wzorca J2EE zawiera także nazwę wzorca z książki GoF, a odpowiednie odwołanie znajduje się w podrozdziale „Powiązane wzorce".

Mapa wzorców

W tym podrozdziale przedstawimy listę typowych wymagań co do aplikacji napotykanych w trakcie tworzenia rozwiązań wykorzystujących platformę J2EE. W skrócie podajemy wymagania lub motywacje, zamieszczając obok jeden lub kilka wzorców związanych z tym zagadnieniem. Choć nie jest to lista wyczerpująca wszystkie możliwości, pozwala łatwo określić, jaki wzorzec zastosować w celu rozwiązania konkretnego problemu.

Tabela 5.6 zawiera zagadnienia związane na ogół z wzorcami warstwy prezentacji wraz z zaznaczeniem, którego wzorca należy użyć.

Tabela 5.6. *Wzorce warstwy prezentacji*

Jeśli szukamy sposobu na ...	Oto rozwiązanie...
wstępne lub końcowe przetwarzanie żądań	wzorzec *Intercepting Filter*
dodanie logowania, testowania lub innego zachowania wykonywanego dla wszystkich żądań	wzorzec *Front Controller* wzorzec *Intercepting Filter*
centralizację sterowania obsługą żądań	wzorzec *Front Controller* wzorzec *Intercepting Filter* wzorzec *Application Controller*
utworzenie ogólnego interfejsu poleceń lub obiektu kontekstu w celu zmniejszenia zależności między komponentami sterującymi i pomocniczymi	wzorzec *Front Controller* wzorzec *Application Controller* wzorzec *Context Object*
implementację kontrolera jako serwletu lub strony JSP	wzorzec *Front Controller*
tworzenie widoku z wielu podwidoków	wzorzec *Composite View*
implementację widoku jako serwletu lub strony JSP	wzorzec *View Helper*
podział modelu i widoku	wzorzec *View Helper*
ukrycie logiki przetwarzania danych, związanej z warstwą prezentacji	wzorzec *View Helper*
implementację komponentów pomocniczych jako JavaBeans lub znaczniki własne	wzorzec *View Helper*
łączenie kilku wzorców warstwy prezentacji	wzorzec *Intercepting Filter* wzorzec *Dispatcher View*
miejsce implementacji logiki związanej z nawigacją i zarządzania widokiem, czyli z wyborem właściwego widoku i pobieraniem go	wzorzec *Service to Worker* wzorzec *Dispatcher View*
miejsce przechowywania stanu sesji	zagadnienie projektowe — podrozdział „Stan sesji u klienta” w rozdziale 2 zagadnienie projektowe — podrozdział „Stan sesji w warstwie prezentacji” w rozdziale 2 zagadnienie projektowe — podrozdział „Stan sesji w warstwie biznesowej” w rozdziale 3
sterowanie dostępem klienta do pewnych widoków lub podwidoków	zagadnienie projektowe — podrozdział „Kontrola dostępu klienta” w rozdziale 2 refaktoryzacja — podrozdział „Ukrywanie zasobów przed klientem” w rozdziale 4
sterowanie otrzymywaniem żądań klientów	zagadnienie projektowe — podrozdział „Duplikacja formularzy” w rozdziale 2 zagadnienie projektowe — podrozdział „Wprowadzenie tokenu synchronizującego” w rozdziale 4

Tabela 5.6. *Wzorce warstwy prezentacji (ciąg dalszy)*

Jeśli szukamy sposobu na ...	Oto rozwiązanie...
wykrywanie powielonych formularzy	zagadnienie projektowe — podrozdział „Duplikacja formularzy" w rozdziale 2 refaktoryzacja — podrozdział „Wprowadzenie tokenu synchronizującego" w rozdziale 4
wykorzystanie automatycznego wypełniania właściwości dzięki `<jsp:setProperty>`	zagadnienie projektowe — podrozdział „Właściwości klas pomocniczych — integralność i spójność" w rozdziale 2
redukcję powiązań między warstwą prezentacji i biznesową	refaktoryzacja — podrozdział „Ukrycie szczegółów warstwy prezentacji przed warstwą biznesową" w rozdziale 4 refaktoryzacja — podrozdział „Wprowadzenie obiektów *Business Delegate*" w rozdziale 4
rozdzielenie kodu dostępu do danych	refaktoryzacja — podrozdział „Wydzielenie kodu dostępu do danych" w rozdziale 4

Tabela 5.7 zawiera zagadnienia związane na ogół z wzorcami warstwy biznesowej wraz z zaznaczeniem, którego wzorca należy użyć.

Tabela 5.7. *Wzorce warstwy biznesowej*

Jeśli szukamy sposobu na...	Oto rozwiązanie...
minimalizację związków między warstwą prezentacji i biznesową	wzorzec *Business Delegate*
buforowanie usług biznesowych	wzorzec *Business Delegate*
ukrycie szczegółów implementacji dostępu, wyszukiwania i tworzenia usług biznesowych	wzorzec *Business Delegate* wzorzec *Service Locator*
izolację zależności od technologii i konkretnego producenta związanych z wyszukiwaniem usług	wzorzec *Service Locator*
zapewnienie jednolitej metody wyszukiwania i tworzenia usług biznesowych	wzorzec *Service Locator*
ukrycie złożoności i zależności związanych z wyszukiwaniem komponentów EJB oraz JMS	wzorzec *Service Locator*
przekazywanie danych między obiektami biznesowymi a klientami w ramach wielu warstw	wzorzec *Transfer Object*
zapewnienie prostszego, jednolitego interfejsu dla zdalnych klientów	wzorzec *Business Delegate* wzorzec *Session Façade* wzorzec *Application Service*
redukcję ilości zdalnych wywołań metod poprzez utworzenie ogólnych metod dostępu do usług warstwy biznesowej	wzorzec *Session Façade*

Tabela 5.7. *Wzorce warstwy biznesowej (ciąg dalszy)*

Jeśli szukamy sposobu na...	Oto rozwiązanie...
zarządzanie związkami między komponentami EJB i ukrycie złożoności interakcji między nimi	wzorzec *Session Façade*
ochronę komponentów warstwy biznesowej przed bezpośrednim dostępem klienta	wzorzec *Session Façade* wzorzec *Application Service*
utworzenie jednolitego sposobu dostępu do komponentów warstwy biznesowej	wzorzec *Session Façade* wzorzec *Application Service*
implementację złożonego modelu domenowego w oparciu o model obiektowy	wzorzec *Business Object*
identyfikację nierozdrobnionych obiektów i obiektów zależnych w celu utworzenia obiektów biznesowych i poprawnego wykorzystania komponentów Entity	wzorzec *Business Object* wzorzec *Composite Entity*
projektowanie nierozdrobnionych komponentów Entity	wzorzec *Composite Entity*
redukcję lub eliminację zależności między klientami komponentów Entity a schematem bazy danych	wzorzec *Composite Entity*
redukcję lub eliminację wzajemnych związków między komponentami Entity	wzorzec *Composite Entity*
redukcję liczby komponentów Entity i poprawę sposobu zarządzania nimi	wzorzec *Composite Entity*
pobranie modelu danych aplikacji z różnych komponentów warstwy biznesowej	wzorzec *Transfer Object Assembler*
tworzenie „w locie” modelu danych aplikacji	wzorzec *Transfer Object Assembler*
ukrycie złożoności konstrukcji modelu danych przed klientem	wzorzec *Transfer Object Assembler*
utworzenie mechanizmów zarządzania zapytaniami i ich obsługą wewnątrz warstwy biznesowej	wzorzec *Value List Handler*
minimalizację narzutu powodowanego przez metody wyszukiwania komponentów EJB	wzorzec *Value List Handler*
buforowanie wyników zapytań na serwerze wraz z możliwością ich wielokrotnego przeglądania	wzorzec *Value List Handler*
wybór pomiędzy komponentami sesyjnymi stanowymi i bezstanowymi	zagadnienie projektowe — podrozdział „Komponent sesyjny — stanowy a bezstanowy” w rozdziale 3
ochronę komponentów Entity przed bezpośrednim dostępem klienta	refaktoryzacja — podrozdział „Ukrycie komponentów Entity za komponentami sesyjnymi” w rozdziale 4
hermetyzację dostępu do usług biznesowych w celu ukrycia szczegółów implementacyjnych związanych z warstwą biznesową	refaktoryzacja — podrozdział „Wprowadzenie obiektów *Business Delegate*” w rozdziale 4

Tabela 5.7. *Wzorce warstwy biznesowej (ciąg dalszy)*

Jeśli szukamy sposobu na...	Oto rozwiązanie...
umieszczenie logiki biznesowej w komponentach Entity	zagadnienie projektowe — podrozdział „Logika biznesowa w komponentach Entity" w rozdziale 3 refaktoryzacja — podrozdział „Przeniesienie logiki biznesowej do warstwy komponentów sesyjnych" w rozdziale 4
utworzenie z komponentów sesyjnych reprezentujących ogólne usługi biznesowe	refaktoryzacja — podrozdział „Łączenie komponentów sesyjnych" w rozdziale 4 refaktoryzacja — podrozdział „Ukrycie komponentów Entity za komponentami sesyjnymi" w rozdziale 4
minimalizację lub wyeliminowanie narzutu sieci i obciążenia kontenera podczas komunikacji pomiędzy komponentami Entity	refaktoryzacja — podrozdział „Redukcja komunikacji między komponentami Entity" w rozdziale 4
wydzielenie kodu związanego z dostępem do danych	refaktoryzacja — podrozdział „Wydzielenie kodu dostępu do danych" w rozdziale 4

Tabela 5.8 zawiera zagadnienia związane na ogół z wzorcami warstwy integracji wraz z zaznaczeniem, którego wzorca należy użyć w konkretnej sytuacji.

Tabela 5.8. *Wzorce warstwy integracji*

Jeśli szukamy sposobu na...	Oto rozwiązanie...
redukcję zależności między warstwą biznesową a warstwą zasobów	wzorzec *Data Access Object*
centralizację dostępu do warstwy zasobów	wzorzec *Data Access Object*
minimalizację złożoności dostępu do zasobów z poziomu komponentów warstwy biznesowej	wzorzec *Data Access Object*
zapewnienie asynchronicznego przetwarzania dla aplikacji biznesowych	wzorzec *Service Activator*
wysłanie asynchronicznego żądania do usługi biznesowej	wzorzec *Service Activator*
asynchroniczne przetwarzanie żądania jako zbioru współbieżnie wykonywanych zadań	wzorzec *Service Activator*
przezroczyste utrwalanie modelu obiektowego	wzorzec *Domain Store*
implementację własnego mechanizmu trwałości obiektów	wzorzec *Domain Store*
udostępnienie usług sieciowych (*Web Services*) przy zastosowaniu XML oraz standardowych protokołów Web	wzorzec *Web Service Broker*
udostępnienie istniejących usług jako usług sieciowych	wzorzec *Web Service Broker*

Podsumowanie

Poznaliśmy podstawowe koncepcje dotyczące wzorców J2EE, opisaliśmy sposób ich kategoryzacji z uwzględnieniem podziału na warstwy oraz przedstawiliśmy mapę pomagającą ustalić, które wzorce stosować w jakich sytuacjach. W kolejnych rozdziałach opisujemy poszczególne wzorce.

Następne rozdziały prezentują wzorce warstwy prezentacji, biznesowej i integracji. W celu odnalezienia potrzebnego wzorca wystarczy zajrzeć do tych rozdziałów.