

## IDŹ DO

PRZYKŁADOWY ROZDZIAŁ



SPIS TREŚCI

## KATALOG KSIĄŻEK

KATALOG ONLINE

ZAMÓW DRUKOWANY KATALOG

## TWÓJ KOSZYK

DODAJ DO KOSZYKA

## CENNIK I INFORMACJE

ZAMÓW INFORMACJE  
O NOWOŚCIACH

ZAMÓW CENNIK

## CZYTELNIA

FRAGMENTY KSIĄŻEK ONLINE

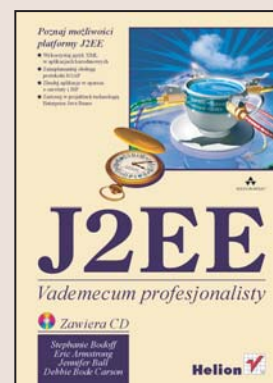
# J2EE. Vademecum profesjonalisty. Wydanie II

Autorzy: Stephanie Bodoff, Eric Armstrong,  
Jennifer Ball, Debbie Bode Carson

Tłumaczenie: Adam Bochenek, Piotr Rajca,  
Jaromir Senczyk, Przemysław Szeremiota  
ISBN: 83-7361-953-4

Tytuł oryginału: [The J2EE Tutorial, Second Edition](#)

Format: B5, stron: 1292



### Poznaj możliwości platformy J2EE

- Wykorzystaj język XML w aplikacjach bazodanowych
- Zaimplementuj obsługę protokołu SOAP
- Zbuduj aplikacje w oparciu o serwlety i JSP
- Zastosuj w projektach technologię Enterprise Java Beans

Platforma Java 2 Enterprise Edition zdobyła już mocną pozycję na rynku serwerów aplikacji. Dzięki niej możliwe stało się tworzenie aplikacji korporacyjnych zgodnych z podstawowym założeniem przyświecającym twórcom Javy – „pisz raz, uruchamiaj wszędzie”. Najnowsza wersja platformy J2EE została znacznie rozbudowana i zmodernizowana w porównaniu z poprzednimi. Dodano do niej możliwość obsługi usług WWW, rozszerzono i unowocześniono implementacje technologii serwletów i JSP oraz poprawiono wiele komponentów przydatnych przy tworzeniu aplikacji korporacyjnych.

„J2EE. Vademecum profesjonalisty. Wydanie II” to kompletny przewodnik po najnowszej wersji Java 2 Enterprise Edition, napisany przez członków zespołu zajmującego się rozwojem platformy J2EE, zatrudnionych w firmie Sun. Opisuje kluczowe komponenty Java 2 Platform, Enterprise Edition (J2EE) w wersji 1.4. Prezentuje rozwiązania konkretnych problemów napotykanymi przez programistów, zilustrowane licznymi przykładami. W książce przedstawiono nie tylko komponenty J2EE, ale również współpracujące z nią technologie: JavaServer Pages Standard Tag Library (JSTL) oraz JavaServer Faces.

- Tworzenie aplikacji internetowej wykorzystującej język XML
- Stosowanie parsera SAX
- Analiza i przetwarzanie hierarchii DOM w dokumentach
- Korzystanie z XPath
- Implementacja usług WWW oraz obsługi protokołu SOAP
- Tworzenie aplikacji z wykorzystaniem serwletów i JSP
- Stosowanie technologii JavaServer Faces
- Projektowanie z wykorzystaniem Enterprise Java Beans

Jeśli chcesz zaprojektować i stworzyć aplikację z wykorzystaniem J2EE, w tej książce znajdziesz wszystkie informacje, jakie mogą Ci być do tego potrzebne.



# Spis treści

	<b>O autorach .....</b>	<b>23</b>
	<b>Słowo wstępne .....</b>	<b>25</b>
	<b>O książce .....</b>	<b>27</b>
Rozdział 1.	<b>Przegląd .....</b>	<b>35</b>
	Rozproszone aplikacje wielowarstwowe .....	36
	Komponenty J2EE .....	37
	Klienci J2EE .....	37
	Komponenty internetowe .....	39
	Komponenty biznesowe .....	40
	Warstwa korporacyjnych systemów informacyjnych (EIS) .....	40
	Kontenery J2EE .....	41
	Usługi kontenerów .....	41
	Typy kontenerów .....	42
	Wsparcie ze strony usług internetowych .....	43
	XML .....	44
	Protokół transportowy SOAP .....	44
	Standardowy format WSDL .....	44
	Standardowe formaty UDDI oraz ebXML .....	45
	Pakowanie aplikacji .....	45
	Role procesu tworzenia aplikacji .....	46
	Dostawca produktu J2EE .....	47
	Dostawca narzędzi .....	47
	Dostawca komponentów aplikacji .....	47
	Konstruktor aplikacji .....	48
	Wdrożeniowiec oraz administrator aplikacji .....	48
	Interfejsy programowania aplikacji platformy J2EE 1.4 .....	49
	Technologia Enterprise JavaBeans .....	50
	Technologia Java Servlet .....	50
	Technologia JavaServer Pages .....	50
	Java Message Service API .....	50
	Java Transaction API .....	51
	JavaMail API .....	51
	JavaBeans Activation Framework .....	51
	Java API for XML Processing .....	51
	Java API for XML-Based RPC .....	52
	SOAP with Attachments API for Java .....	52
	Java API for XML Registries .....	52
	Architektura J2EE Connector .....	53
	JDBC API .....	53
	Java Naming and Directory Interface .....	54

	Java Authentication and Authorization Service .....	54
	Uproszczona integracja systemów .....	55
	Sun Java System Application Server Platform Edition 8 .....	55
	Technologie .....	56
	Narzędzia .....	57
	Uruchamianie i zatrzymywanie serwera .....	58
	Uruchamianie konsoli administracyjnej .....	58
	Uruchamianie narzędzia deploytool .....	59
	Uruchamianie i zatrzymywanie serwera bazy danych PointBase .....	59
	Uruchamianie aplikacji J2EE .....	59
<b>Rozdział 2.</b>	<b>Zrozumieć XML .....</b>	<b>63</b>
	Wprowadzenie do języka XML .....	63
	Czym jest XML? .....	63
	Dlaczego XML jest taki ważny? .....	67
	Jak można używać języka XML? .....	70
	Generacja danych XML .....	72
	Tworzenie prostego pliku XML .....	72
	Definiowanie elementu głównego .....	73
	Tworzenie instrukcji przetwarzania .....	77
	Błąd popełniony celowo .....	78
	Zastępowanie i wstawianie tekstu .....	79
	Tworzenie definicji typu dokumentu (DTD) .....	81
	Dokumenty i dane .....	86
	Definiowanie atrybutów i encji w DTD .....	86
	Odwołania do encji binarnych .....	92
	Definiowanie encji parametrów oraz sekcji warunkowych .....	94
	Rozwiązywanie konfliktów nazw .....	97
	Stosowanie przestrzeni nazw .....	98
	Projektowanie struktury danych XML .....	100
	Oszczędzanie pracy .....	101
	Atrybuty i elementy .....	101
	Normalizacja danych .....	103
	Normalizacja DTD .....	104
	Podsumowanie .....	105
<b>Rozdział 3.</b>	<b>Podstawowe sposoby tworzenia aplikacji internetowych .....</b>	<b>107</b>
	Cykl istnienia aplikacji internetowej .....	109
	Moduły internetowe .....	111
	Pakowanie modułów internetowych .....	112
	Wdrażania modułu internetowego .....	114
	Wyświetlanie listy wdrożonych modułów .....	117
	Aktualizacja modułów internetowych .....	117
	Usuwanie modułów internetowych .....	119
	Konfiguracja aplikacji internetowych .....	120
	Odwzorowania adresów URL na komponenty internetowe .....	120
	Deklarowanie plików powitalnych .....	122
	Określanie parametrów inicjalizacyjnych .....	123
	Kojarzenie błędów ze stronami informacyjnymi .....	123
	Deklarowanie odwołań do zasobów .....	124
	Przykład — Księgarnia Duke'a .....	124
	Korzystanie z baz danych w aplikacjach internetowych .....	125
	Zapisywanie informacji w przykładowej bazie danych .....	125
	Tworzenie źródła danych w serwerze Application Server .....	126
	Określanie odwołania do zasobu aplikacji .....	127
	Kojarzenie odwołania do zasobu ze źródłem danych .....	128
	Dodatkowe informacje .....	128

Rozdział 4.	<b>Interfejs API do przetwarzania danych XML .....</b>	<b>129</b>
	JAXP API .....	129
	Przegląd pakietów .....	130
	SAX API .....	131
	Pakiety SAX .....	133
	DOM API .....	133
	Pakiety DOM .....	134
	XSLT API .....	134
	Pakiety XSLT .....	135
	Stosowanie bibliotek JAXP .....	136
	Co dalej? .....	136
Rozdział 5.	<b>Prosty interfejs programowania aplikacji do obsługi XML-a .....</b>	<b>137</b>
	Kiedy używać mechanizmu SAX? .....	138
	Wyświetlanie zawartości pliku XML przy wykorzystaniu parsera SAX .....	139
	Tworzenie szkieletu aplikacji .....	140
	Importowanie klas .....	140
	Przygotowania do operacji wejścia-wyjścia .....	140
	Implementacja interfejsu ContentHandler .....	141
	Przygotowanie parsera .....	142
	Generacja danych wynikowych .....	143
	Formatowanie danych wynikowych .....	143
	Obsługa zdarzeń związanych z zawartością danych XML .....	144
	Kompilacja i uruchamianie programu .....	148
	Sprawdzanie wyników .....	148
	Identyfikacja zdarzeń .....	149
	Generacja bardziej zwartych danych wynikowych .....	151
	Sprawdzanie wyników .....	153
	Dokumenty i dane .....	154
	Dodawanie obsługi innych zdarzeń .....	154
	Określanie lokalizacji dokumentu .....	154
	Obsługa instrukcji przetwarzania .....	156
	Podsumowanie .....	157
	Obsługa błędów w przypadku stosowania parsera nieweryfikującego .....	157
	Obsługa wyjątków SAXParseException .....	158
	Obsługa wyjątków SAXException .....	159
	Poprawa obsługi wyjątków SAXParseException .....	160
	Obsługa wyjątków ParserConfigurationException .....	161
	Obsługa wyjątków IOException .....	162
	Obsługa błędów niekrytycznych .....	162
	Obsługa ostrzeżeń .....	163
	Wyświetlanie danych specjalnych oraz danych CDATA .....	164
	Obsługa znaków specjalnych .....	164
	Obsługa tekstu o składni przypominającej XML .....	164
	Obsługa danych CDATA oraz innych znaków .....	165
	Analiza syntaktyczna w przypadku zastosowania DTD .....	166
	Wpływ DTD na parser nieweryfikujący .....	166
	Znaki odstępu, które można ignorować .....	167
	Porządki .....	169
	Elementy puste raz jeszcze .....	169
	Generacja odwołań do encji .....	169
	Generacja encji zewnętrznych .....	170
	Podsumowanie informacji o encjach .....	170
	Wybór implementacji parsera .....	171

Wykorzystanie parsera weryfikującego .....	171
Konfiguracja klasy fabrykującej .....	171
Weryfikacja przy wykorzystaniu schematów XML .....	172
Eksperymenty z błędami weryfikacji .....	174
Obsługa błędów w parserach weryfikujących .....	176
Analiza sparametryzowanej DTD .....	176
Ostrzeżenia DTD .....	177
Obsługa zdarzeń leksykalnych .....	178
Sposób działania interfejsu LexicalHandler .....	179
Posługiwanie się interfejsem LexicalHandler .....	180
Zastosowanie interfejsów DTDHandler oraz EntityResolver .....	184
Interfejs DTDHandler .....	185
Interfejs EntityResolver .....	186
Dalsze informacje .....	186
<b>Rozdział 6. Obiektowy model dokumentu .....</b>	<b>187</b>
Kiedy używać DOM? .....	187
Dokumenty a dane .....	188
Model zawartości mieszanej .....	188
Prostszy model .....	189
Zwiększenie stopnia złożoności .....	190
Wybór modelu .....	192
Wczytywanie danych XML do DOM .....	193
Tworzenie programu .....	193
Dodatkowe informacje .....	197
Co dalej? .....	199
Wyświetlanie hierarchii DOM .....	199
Zmiana aplikacji DomEcho w aplikację GUI .....	199
Utworzenie adapterów w celu wyświetlenia DOM w komponencie JTree .....	204
Dopracowanie kodu aplikacji .....	212
Badanie struktury DOM .....	212
Wyświetlanie prostego drzewa DOM .....	212
Wyświetlanie bardziej złożonego drzewa DOM .....	214
Zakończenie .....	219
Prezentowanie struktury DOM w sposób optymalny .....	219
Kompresja zawartości drzewa .....	219
Operacje na zaznaczonych elementach drzewa JTree .....	224
Obsługa modyfikacji .....	232
Zakończenie .....	232
Tworzenie i modyfikowanie DOM .....	232
Pobieranie DOM z obiektu fabrykującego .....	233
Normalizacja DOM .....	236
Inne operacje .....	237
Zakończenie .....	240
Weryfikacja przy użyciu schematów XML .....	240
Ogólne informacje o procesie weryfikacji .....	241
Konfiguracja obiektu fabrykującego DocumentBuilder .....	241
Weryfikacja w przypadku stosowania wielu przestrzeni nazw .....	242
Dalsze informacje .....	245
<b>Rozdział 7. Przekształcenia rozszerzalnego języka arkuszy stylów .....</b>	<b>247</b>
Przedstawienie XSL, XSLT oraz XPath .....	247
Pakiety JAXP związane z obsługą przekształceń .....	248
Jak działa XPath? .....	249
Wyrażenia XPath .....	249
Model danych XSLT i XPath .....	250

Szablony i konteksty .....	250
Proste adresowanie przy użyciu XPath .....	250
Proste wyrażenia XPath .....	251
Łączenie adresów indeksowych .....	252
Znaki wieloznaczne .....	253
Adresowanie przy użyciu ścieżki rozszerzonej .....	253
Typy danych i operatory XPath .....	253
Wartość łańcuchowa elementu .....	254
Funkcje XPath .....	254
Podsumowanie .....	257
Zapis DOM w pliku XML .....	257
Odczyt XML .....	257
Tworzenie obiektu przekształcenia .....	259
Zapis XML .....	261
Zapis fragmentu drzewa DOM .....	261
Podsumowanie .....	263
Generacja XML na podstawie dowolnych danych .....	263
Tworzenie prostego pliku danych .....	263
Tworzenie prostego parsera .....	265
Modyfikacja parsera, by generował zdarzenia SAX .....	267
Użycie parsera jako obiektu SAXSource .....	272
Wykonanie konwersji .....	274
Przekształcanie danych XML przy użyciu XSLT .....	275
Definiowanie prostego typu dokumentu — <article> .....	275
Tworzenie dokumentu testowego .....	277
Tworzenie przekształcenia XSLT .....	278
Przetwarzanie podstawowych elementów struktury .....	279
Tworzenie prostego programu .....	282
Usuwanie znaków odstępu .....	284
Przetwarzanie pozostałych elementów struktury .....	286
Przetwarzanie elementów wewnątrzwierszowych (treści) .....	290
Drukowanie kodu HTML .....	294
Jakie są pozostałe możliwości XSLT? .....	294
Zastosowanie pakietu Xalan do wykonywania przekształceń z poziomu wiersza poleceń .....	296
Łączenie przekształceń przy użyciu łańcucha filtrów .....	296
Pisanie programu .....	296
Rozumienie sposobu działania łańcucha filtrów .....	299
Testowanie programu .....	300
Wnioski .....	302
Dalsze informacje .....	303
<b>Rozdział 8. Tworzenie usług WWW przy użyciu JAX-RPC .....</b>	<b>305</b>
Określanie portu .....	306
Tworzenie prostej usługi WWW oraz klienta przy wykorzystaniu JAX-RPC .....	306
Kodowanie interfejsu punktu końcowego usługi oraz klas implementacji .....	307
Przygotowanie usługi .....	308
Pakowanie usługi .....	310
Określanie adresu punktu końcowego .....	311
Wdrażanie usługi .....	311
Klient używający statycznego pieńka .....	312
Obsługiwane typy danych .....	314
Typy J2SE SDK .....	315
Typy podstawowe .....	315
Tablice .....	315
Typy wartości .....	315
Komponenty JavaBeans .....	316

Klienty usług WWW .....	316
Klient używający dynamicznego pośrednika .....	316
Klient DII .....	319
Klient aplikacji .....	322
Pozostałe klienty JAX-RPC .....	325
Współpraca usług WWW (WS-I) a JAX-RPC .....	326
Dodatkowe informacje .....	326
<b>Rozdział 9. Interfejs programowania aplikacji do obsługi protokołu SOAP z załącznikami .....</b>	<b>327</b>
Przegląd SAAJ .....	328
Komunikaty .....	328
Połączenia .....	331
Informator .....	332
Tworzenie i wysyłanie prostego komunikatu .....	333
Dodawanie zawartości do nagłówka .....	341
Dodawanie zawartości do obiektu SOAPPart .....	342
Dodawanie dokumentu do treści komunikatu SOAP .....	343
Operacje na zawartości komunikatów przy użyciu interfejsów programowania aplikacji SAAJ oraz DOM .....	343
Dodawanie załączników .....	343
Dodawanie atrybutów .....	346
Stosowanie usterek SOAP .....	350
Przykłady .....	354
Program Request.java .....	355
Program MyUddiPing.java .....	356
Program HeaderExample.java .....	362
Programy DOMExample.java oraz DOMSrcExample.java .....	363
Program Attachments.java .....	366
Program SOAPFaultTest.java .....	368
Dodatkowe informacje .....	369
<b>Rozdział 10. Interfejs programowania aplikacji do obsługi rejestrów XML .....</b>	<b>371</b>
Podstawowe informacje o JAXR .....	371
Czym jest rejestr? .....	371
Czym jest JAXR? .....	372
Architektura JAXR .....	373
Implementacja klienta JAXR .....	374
Nawiązywanie połączenia .....	375
Przeszukiwanie rejestru .....	380
Zarządzanie danymi rejestru .....	384
Zastosowanie typologii w klientach JAXR .....	390
Uruchamianie przykładowych klientów .....	394
Zanim skompilujemy przykłady .....	396
Kompilacja przykładów .....	398
Uruchamianie przykładów .....	398
Stosowanie klientów JAXR w aplikacjach J2EE .....	403
Tworzenie kodu klienta aplikacji — MyAppClient.java .....	403
Tworzenie kodu komponentu sesyjnego PubQuery .....	404
Kompilacja plików źródłowych .....	404
Importowanie certyfikatów .....	405
Uruchamianie serwera Application Server .....	406
Tworzenie zasobów JAXR .....	406
Tworzenie i pakowanie aplikacji .....	407
Wdrażanie aplikacji .....	409
Uruchamianie klienta aplikacji .....	409
Dodatkowe informacje .....	410

<b>Rozdział 11. Technologia serwetów Java .....</b>	<b>411</b>
Czym jest serwet? .....	411
Przykład serwetów .....	412
Rozwiązywanie problemów .....	415
Cykl życia serwetu .....	416
Obsługa zdarzeń związanych z cyklem życia serwetu .....	416
Obsługa błędów .....	418
Współdzielenie informacji .....	418
Wykorzystanie obiektów zakresu .....	418
Kontrola współbieżnego dostępu do współdzielonych zasobów .....	419
Dostęp do baz danych .....	420
Inicjalizacja serwetu .....	421
Implementacja metody usługowej .....	422
Pobieranie informacji z żądania .....	422
Tworzenie odpowiedzi .....	424
Filtrowanie żądań i odpowiedzi .....	426
Programowanie filtrów .....	427
Programowanie niestandardowych żądań i odpowiedzi .....	428
Specyfikacja odwzorowań filtrów .....	430
Wywoływanie innych zasobów .....	432
Dołączanie innych zasobów do odpowiedzi .....	432
Przekazywanie sterowania innemu komponentowi .....	433
Dostęp do kontekstu .....	434
Przechowywanie stanu klienta .....	435
Dostęp do sesji .....	435
Wiązanie obiektów z sesją .....	435
Zarządzanie sesjami .....	436
Śledzenie sesji .....	437
Finalizacja serwetu .....	437
Śledzenie żądań usługi .....	438
Powiadamianie metod o zakończeniu działania serwetu .....	439
Poprawna implementacja czasochłonnych metod .....	439
Dalsze informacje .....	440
<b>Rozdział 12. Technologia stron JSP .....</b>	<b>441</b>
Czym jest strona JSP? .....	441
Przykład .....	442
Przykład stron JSP .....	444
Cykl życia strony JSP .....	449
Tłumaczenie i kompilacja .....	450
Wykonanie .....	451
Tworzenie treści statycznej .....	453
Kodowanie strony i odpowiedzi .....	453
Tworzenie treści dynamicznej .....	454
Zastosowanie obiektów na stronach JSP .....	454
Język wyrażeń .....	455
Wyłączanie wartościowania wyrażeń .....	456
Używanie wyrażeń .....	456
Zmienne .....	457
Obiekty niejawne .....	458
Literały .....	459
Operatory .....	459
Słowa kluczowe .....	460
Przykłady .....	460
Funkcje .....	461



	Komponenty JavaBeans .....	462
	Konwencje projektowania komponentów JavaBeans .....	462
	Tworzenie i używanie komponentów JavaBeans .....	463
	Konfigurowanie właściwości komponentów JavaBeans .....	464
	Pobieranie właściwości komponentu JavaBeans .....	465
	Stosowanie znaczników niestandardowych .....	466
	Deklarowanie bibliotek znaczników .....	466
	Dołączanie implementacji biblioteki znaczników .....	468
	Ponowne użycie treści na stronach JSP .....	469
	Przekazywanie sterowania do innego komponentu .....	470
	Element <code>jsp:param</code> .....	470
	Dołączanie apletu .....	471
	Konfigurowanie właściwości grup stron JSP .....	473
	Wyłączanie wyznaczania wartości wyrażeń .....	473
	Deklarowanie kodowania strony .....	474
	Definiowanie niejawnego dołączania .....	474
	Dalsze informacje .....	475
Rozdział 13.	<b>Dokumenty JSP .....</b>	<b>477</b>
	Przykład dokumentu JSP .....	477
	Tworzenie dokumentu JSP .....	482
	Deklarowanie bibliotek znaczników .....	484
	Umieszczanie dyrektyw w dokumentach JSP .....	486
	Tworzenie treści statycznych i dynamicznych .....	487
	Zastosowanie elementu <code>jsp:root</code> .....	489
	Zastosowanie elementu <code>jsp:output</code> .....	490
	Identyfikacja dokumentu JSP przez kontener .....	493
Rozdział 14.	<b>Biblioteka JSTL .....</b>	<b>495</b>
	Przykład stron JSP .....	495
	Korzystanie z biblioteki JSTL .....	498
	Współpraca znaczników .....	499
	Biblioteka znaczników podstawowych .....	500
	Znaczniki obsługi zmiennych .....	500
	Znaczniki sterowania przepływem .....	501
	Znaczniki URL .....	504
	Pozostałe znaczniki .....	505
	Biblioteka znaczników XML .....	505
	Znaczniki podstawowe .....	507
	Znaczniki sterowania przepływem .....	508
	Znaczniki przekształceń .....	508
	Biblioteka znaczników internacjonalizacji .....	509
	Konfiguracja lokalizacji .....	509
	Znaczniki komunikatów .....	510
	Znaczniki formatowania .....	510
	Biblioteka znaczników SQL .....	511
	Interfejs Result znacznika query .....	513
	Funkcje .....	514
	Dalsze informacje .....	515
Rozdział 15.	<b>Niestandardowe znaczniki JSP .....</b>	<b>517</b>
	Czym jest niestandardowy znacznik? .....	518
	Przykład stron JSP .....	518
	Typy znaczników .....	522
	Znaczniki z atrybutami .....	522
	Znaczniki posiadające ciało .....	524

	Znaczniki definiujące zmienne .....	525
	Komunikacja między znacznikami .....	525
	Zastosowanie plików znaczników do hermetyzacji treści w celu wielokrotnego użycia ..	526
	Położenie plików znaczników .....	528
	Dyrektywy plików znaczników .....	528
	Przetwarzanie fragmentów przekazywanych plikom znaczników .....	534
	Przykłady .....	534
	Deskryptory bibliotek znaczników .....	537
	Elementy nadrzędne deskryptora biblioteki znaczników .....	538
	Deklarowanie plików znaczników .....	539
	Deklarowanie obiektów obsługi znaczników .....	541
	Deklarowanie atrybutów znacznika dla obiektu obsługi znacznika .....	542
	Deklarowanie zmiennych znacznika dla obiektu obsługi znacznika .....	543
	Implementacja obiektów obsługi znaczników prostych .....	545
	Dołączanie obiektów obsługi znaczników do aplikacji internetowych .....	545
	Wywołania obiektu obsługi znacznika prostego .....	545
	Obiekty obsługi dla znaczników prostych .....	546
	Obiekty obsługi dla znaczników posiadających atrybuty .....	546
	Obiekty obsługi znaczników prostych posiadających ciało .....	548
	Obiekty obsługi znaczników definiujących zmienne .....	549
	Współpraca znaczników .....	551
	Przykłady .....	553
<b>Rozdział 16.</b>	<b>Skrypty na stronach JSP .....</b>	<b>561</b>
	Przykład stron JSP .....	561
	Posługiwanie się elementami skryptów .....	563
	Wyłączanie skryptów .....	563
	Deklaracje .....	564
	Inicjalizacja i finalizacja strony JSP .....	564
	Skryptlety .....	564
	Wyrażenia .....	565
	Implementacja znaczników akceptujących elementy skryptów .....	566
	Elementy TLD .....	566
	Obiekty obsługi znaczników .....	566
	Znaczniki posiadające ciało .....	568
	Współpraca znaczników .....	570
	Znaczniki definiujące zmienne .....	571
<b>Rozdział 17.</b>	<b>Technologia JavaServer Faces .....</b>	<b>573</b>
	Korzyści związane ze stosowaniem technologii JavaServer Faces .....	574
	Czym jest aplikacja JavaServer Faces? .....	575
	Role użytkowników szkieletu .....	576
	Prosta aplikacja JavaServer Faces .....	577
	Etapy tworzenia aplikacji .....	577
	Tworzenie stron .....	579
	Definiowanie nawigacji .....	581
	Tworzenie komponentów .....	582
	Dodawanie deklaracji komponentów zarządzanych .....	583
	Model interfejsu użytkownika oparty na komponentach .....	584
	Klasy komponentów interfejsu użytkownika .....	585
	Model wyświetlania komponentów .....	587
	Model konwersji .....	590
	Model odbiornika i zdarzenia .....	591
	Model kontroli poprawności .....	592
	Model nawigacji .....	593
	Zarządzanie komponentami pomocniczymi .....	594
	Współpraca elementów aplikacji .....	597

	Cykl życia strony JavaServer Faces .....	599
	Scenariusze cyklu przetwarzania żądania .....	600
	Standardowy cykl przetwarzania żądania .....	601
	Dalsze informacje .....	605
<b>Rozdział 18.</b>	<b>Stosowanie technologii JavaServer Faces na stronach JSP .....</b>	<b>607</b>
	Przykład aplikacji JavaServer Faces .....	607
	Tworzenie strony .....	611
	Stosowanie znaczników podstawowych .....	613
	Stosowanie znaczników komponentów HTML .....	613
	Atrybuty znaczników komponentów interfejsu użytkownika .....	615
	Komponent UIForm .....	617
	Komponent UIColumn .....	618
	Komponent UICommand .....	618
	Komponent UIData .....	620
	Komponent UIGraphic .....	623
	Komponenty UIInput i UIOutput .....	623
	Komponent UIPanel .....	626
	Komponent UISelectBoolean .....	628
	Komponent UISelectMany .....	629
	Komponenty UIMessage i UIMessages .....	630
	Komponent UISelectOne .....	630
	Komponenty UISelectItem, UISelectItems i UISelectItemGroup .....	631
	Stosowanie zlokalizowanych komunikatów .....	634
	Odwołania do zestawu zasobów ResourceBundle .....	634
	Odwołania do zlokalizowanego komunikatu .....	635
	Stosowanie standardowych konwerterów .....	635
	Stosowanie konwertera DateTimeConverter .....	637
	Stosowanie konwertera NumberConverter .....	638
	Rejestracja odbiorników .....	639
	Rejestracja odbiornika zdarzeń zmiany wartości .....	639
	Rejestracja odbiornika zdarzeń akcji .....	640
	Stosowanie walidatorów standardowych .....	640
	Wartość wymagana .....	641
	Stosowanie walidatora LongRangeValidator .....	641
	Wiązanie wartości i instancji komponentów z zewnętrznymi źródłami danych .....	642
	Wiązanie wartości komponentu z właściwością .....	643
	Wiązanie wartości komponentu z obiektem niejawnym .....	644
	Wiązanie instancji komponentu z właściwością komponentu pomocniczego .....	645
	Odwołania do metody komponentu pomocniczego .....	646
	Odwołanie do metody nawigacji .....	647
	Odwołania do metody obsługi zdarzenia akcji .....	647
	Odwołania do metody kontroli poprawności .....	648
	Odwołania do metody obsługi zdarzenia zmiany wartości .....	648
	Stosowanie obiektów niestandardowych .....	649
	Stosowanie niestandardowego konwertera .....	650
	Stosowanie niestandardowego walidatora .....	650
	Stosowanie niestandardowego komponentu .....	651
<b>Rozdział 19.</b>	<b>Programowanie dla JavaServer Faces .....</b>	<b>653</b>
	Implementacja właściwości komponentu .....	653
	Implementacja właściwości wiązanych z wartościami komponentów .....	654
	Implementacja właściwości związanych z instancjami komponentów .....	661
	Lokalizacja .....	662
	Tworzenie zestawu zasobów .....	662
	Lokalizacja danych dynamicznych .....	662
	Lokalizacja komunikatów .....	663

Implementacja niestandardowego konwertera .....	665
Implementacja odbiornika zdarzeń .....	667
Implementacja odbiorników zmiany wartości .....	667
Implementacja odbiorników zdarzeń akcji .....	668
Implementacja niestandardowego walidatora .....	669
Implementacja interfejsu Validator .....	670
Implementacja niestandardowego znacznika .....	672
Implementacja metod komponentu pomocniczego .....	674
Implementacja metody nawigacji .....	674
Implementacja metody obsługi zdarzenia akcji .....	675
Implementacja metody kontroli poprawności .....	676
Implementacja metody obsługi zdarzenia zmiany wartości .....	677
<b>Rozdział 20. Implementacja niestandardowych komponentów</b>	
<b>interfejsu użytkownika .....</b>	<b>679</b>
Jak ustalić, czy potrzebujemy niestandardowego komponentu? .....	680
Kiedy używać niestandardowego komponentu? .....	680
Kiedy używać niestandardowego obiektu wyświetlania? .....	681
Kombinacje komponentów, obiektów wyświetlania i znaczników .....	682
Przykład mapy graficznej .....	683
Implementacja mapy graficznej przy użyciu technologii JavaServer Faces .....	683
Wyświetlanie strony HTML .....	683
Strona JSP .....	684
Konfigurowanie danych modelu .....	685
Podsumowanie klas aplikacji .....	687
Etapy implementacji niestandardowego komponentu .....	687
Implementacja obiektu obsługi znacznika komponentu .....	688
Definiowanie niestandardowego znacznika komponentu	
przez deskryptor biblioteki znaczników .....	692
Implementacja klas niestandardowych komponentów .....	693
Kodowanie .....	696
Dekodowanie .....	698
Umożliwianie wiązania wartości właściwościom komponentu .....	698
Przechowywanie i odtwarzanie stanu .....	699
Delegowanie wyświetlania do klasy wyświetlania .....	700
Implementacja klasy wyświetlania .....	701
Identyfikacja typu obiektu wyświetlania .....	702
Obsługa zdarzeń dla niestandardowych komponentów .....	702
<b>Rozdział 21. Konfigurowanie aplikacji JavaServer Faces .....</b>	<b>705</b>
Plik konfiguracyjny zasobów aplikacji .....	705
Konfigurowanie komponentów .....	706
Stosowanie elementu managed-bean .....	707
Inicjalizacja właściwości za pomocą elementu managed-property .....	708
Inicjalizacja map i list .....	713
Rejestracja komunikatów .....	714
Rejestracja niestandardowego walidatora .....	715
Rejestracja niestandardowego konwertera .....	716
Konfigurowanie reguł nawigacji .....	716
Rejestracja niestandardowej klasy wyświetlania w pakiecie wyświetlania .....	719
Rejestracja niestandardowego komponentu .....	720
Podstawowe wymagania aplikacji JavaServer Faces .....	721
Konfigurowanie aplikacji za pomocą programu deploytool .....	722
Dołączanie wymaganych plików JAR .....	725
Dołączanie klas, stron i innych zasobów .....	726

<b>Rozdział 22.</b>	<b>Internacjonalizacja i lokalizacja aplikacji internetowych .....</b>	<b>727</b>
	Klasy lokalizacji na platformie Java .....	727
	Przygotowywanie zlokalizowanych komunikatów i etykiet .....	728
	Określanie lokalizacji .....	728
	Wybór zestawu zasobów .....	729
	Pobieranie zlokalizowanych komunikatów .....	729
	Formatowanie dat i liczb .....	730
	Zbiory znaków i ich kodowanie .....	731
	Zbiory znaków .....	731
	Kodowanie znaków .....	731
	Dalsze informacje .....	734
<b>Rozdział 23.</b>	<b>Enterprise Beans .....</b>	<b>735</b>
	Czym jest komponent Enterprise Bean? .....	735
	Korzyści płynące z używania komponentów EJB .....	735
	Kiedy należy używać komponentów EJB? .....	736
	Rodzaje komponentów Enterprise Beans .....	736
	Komponenty sesyjne .....	736
	Tryby zarządzania stanem .....	737
	Kiedy używać komponentów sesyjnych? .....	738
	Komponenty encyjne .....	738
	Co odróżnia komponenty encyjne od sesyjnych? .....	739
	Trwałość zarządzana przez kontener .....	740
	Kiedy używać komponentów encyjnych? .....	743
	Komponenty sterowane komunikatami .....	743
	Jaka jest różnica między komponentami sterowanymi komunikatami a sesyjnymi i encyjnymi obiektami EJB? .....	743
	Kiedy używać komponentów sterowanych komunikatami? .....	744
	Definiowanie dostępu do komponentów za pomocą interfejsów .....	745
	Klient zdalny .....	745
	Klient lokalny .....	746
	Interfejsy lokalne a trwałość zarządzana przez kontener .....	746
	Wybór pomiędzy dostępem zdalnym i lokalnym .....	747
	Klient usługi internetowej .....	748
	Rodzaj dostępu a parametry metod .....	748
	Składniki komponentu EJB .....	749
	Konwencja nazw przyjęta dla komponentów EJB .....	750
	Cykl życia komponentów Enterprise Beans .....	750
	Cykl życia stanowego komponentu sesyjnego .....	751
	Cykl życia bezstanowego komponentu sesyjnego .....	752
	Cykl życia komponentu encyjnego .....	752
	Cykl życia komponentu sterowanego komunikatami .....	754
	Dalsze informacje .....	754
<b>Rozdział 24.</b>	<b>Komponenty Enterprise Beans — pierwsze kroki .....</b>	<b>755</b>
	Tworzenie aplikacji J2EE .....	756
	Tworzenie komponentu EJB .....	756
	Przygotowanie kodu źródłowego komponentu .....	756
	Kompilacja plików z kodem źródłowym .....	758
	Pakowanie komponentu EJB .....	758
	Tworzenie aplikacji klienta .....	759
	Kodowanie aplikacji klienta .....	760
	Kompilacja programu klienta .....	762
	Pakowanie aplikacji klienta .....	762
	Określenie nazwy komponentu, do którego odwołuje się klient .....	763

Tworzenie klienta sieciowego .....	764
Kodowanie klienta sieciowego .....	764
Kompilacja klienta sieciowego .....	765
Pakowanie klienta sieciowego .....	765
Określenie nazwy komponentu, do którego odwołuje się klient sieciowy .....	766
Przyporządkowanie komponentowi nazwy .....	767
Określenie ścieżki bazowej kontekstu klienta sieciowego .....	767
Wdrożenie aplikacji J2EE .....	768
Uruchomienie aplikacji klienta .....	769
Uruchomienie klienta sieciowego .....	769
Modyfikowanie aplikacji J2EE .....	770
Zmiany w pliku klasy komponentu .....	770
Dodanie pliku .....	771
Modyfikacja ustawień wdrożenia .....	771
<b>Rozdział 25. Przykłady komponentów sesyjnych .....</b>	<b>773</b>
Przykład CartBean .....	773
Klasa komponentu sesyjnego .....	774
Interfejs domowy .....	777
Interfejs zdalny .....	778
Klasy pomocnicze .....	779
Kompilacja przykładu CartBean .....	779
Przygotowanie aplikacji .....	779
Pakowanie komponentu EJB .....	779
Pakowanie programu klienta .....	780
Przykład usługi sieciowej — HelloServiceBean .....	782
Interfejs punktu końcowego usługi sieciowej .....	783
Klasa implementująca bezstanowy komponent sesyjny .....	783
Kompilacja HelloServiceBean .....	783
Tworzenie klienta usługi internetowej .....	786
Uruchomienie klienta usługi internetowej .....	787
Inne cechy komponentów EJB .....	787
Dostęp do parametrów zewnętrznych .....	787
Porównywanie komponentów EJB .....	788
Przekazanie referencji wskazującej komponent EJB .....	788
Usługa timera .....	789
Tworzenie timera .....	790
Wyłączanie i zapamiętywanie timerów .....	790
Pobranie informacji na temat timera .....	791
Timery a transakcje .....	791
Przykładowy komponent TimerSessionBean .....	791
Przygotowanie komponentu TimerSessionBean .....	792
Obsługa wyjątków .....	797
<b>Rozdział 26. Przykłady komponentów encyjnych</b>	
<b>    bezpośrednio zarządzających trwałością .....</b>	<b>799</b>
Przykładowy komponent SavingsAccountBean .....	799
Klasa komponentu encyjnego .....	800
Interfejs domowy .....	808
Interfejs zdalny .....	809
Uruchomienie przykładu SavingsAccountBean .....	810
Odwzorowanie relacji w komponentach bezpośrednio zarządzających trwałością .....	812
Relacje „jeden do jednego” .....	812
Relacje „jeden do wielu” .....	815
Relacje „wiele do wielu” .....	821

Klucze główne komponentów bezpośrednio zarządzających trwałością .....	823
Klasa klucza głównego .....	823
Klucze główne w klasie komponentu encyjnego .....	824
Pobranie klucza głównego .....	825
Rady dotyczące narzędzia deploytool i komponentów encyjnych bezpośrednio zarządzających trwałością .....	826
<b>Rozdział 27. Przykłady komponentów encyjnych</b>	
<b>o trwałości zarządzanej przez kontener .....</b>	<b>827</b>
Opis aplikacji RosterApp .....	827
Komponent PlayerBean .....	828
Klasa komponentu .....	828
Lokalny interfejs domowy .....	832
Interfejs lokalny .....	833
Funkcje realizowane przez aplikację RosterApp .....	834
Tworzenie obiektu gracza .....	834
Dopisanie gracza do zespołu .....	835
Usunięcie obiektu gracza .....	836
Usunięcie gracza z drużyny .....	836
Pobranie listy wszystkich graczy drużyny .....	837
Pobranie kopii listy graczy drużyny .....	839
Wyszukiwanie graczy na podstawie zajmowanych przez nich pozycji .....	840
Wyszukiwanie dyscyplin uprawianych przez zawodnika .....	841
Stworzenie i uruchomienie przykładowej aplikacji RosterApp .....	842
Tworzenie tabel bazy danych .....	842
Tworzenie źródła danych .....	843
Schemat przyporządkowania pól .....	843
Kompilacja komponentów EJB .....	844
Tworzenie aplikacji .....	844
Pakowanie komponentów EJB .....	844
Pakowanie aplikacji klienta .....	852
Wdrożenie aplikacji .....	853
Uruchomienie aplikacji klienta .....	853
Przewodnik po ustawieniach aplikacji RosterApp .....	854
RosterApp .....	854
RosterClient .....	855
RosterJAR .....	856
TeamJAR .....	856
Klucz główny a trwałość zarządzana przez kontener .....	861
Klasa klucza głównego .....	861
Zaawansowane zagadnienia związane z CMP — przykład OrderApp .....	864
Struktura aplikacji OrderApp .....	864
Relacje pomiędzy komponentami aplikacji OrderApp .....	865
Klucze główne komponentów encyjnych aplikacji OrderApp .....	867
Komponent encyjny przyporządkowany więcej niż jednej tabeli bazy danych .....	869
Metody wyszukiujące i metody selekcji .....	869
Metody domowe .....	870
Kaskadowe usuwanie w aplikacji OrderApp .....	870
Pola typu BLOB i CLOB w aplikacji OrderApp .....	870
Przygotowanie i uruchomienie przykładu OrderApp .....	871
Rady dotyczące narzędzia deploytool i komponentów encyjnych o trwałości zarządzanej przez kontener .....	877
Wybór pól stanu i abstrakcyjnego schematu trwałości .....	877
Definicja zapytań EJB QL związanych z metodami wyszukiującymi i metodami selekcji .....	878
Definicja relacji .....	878
Tworzenie tabel bazy danych w czasie wdrażania aplikacji .....	879

<b>Rozdział 28. Przykład komponentu sterowanego komunikatami .....</b>	<b>881</b>
Ogólna charakterystyka przykładowej aplikacji .....	881
Program klienta .....	882
Klasa komponentu sterowanego komunikatami .....	882
Metoda onMessage .....	883
Metody ejbCreate i ejbRemove .....	884
Wdrożenie i uruchomienie aplikacji SimpleMessageApp .....	884
Stworzenie zasobów obsługujących komunikację .....	884
Wdrożenie aplikacji .....	885
Uruchomienie programu klienta .....	886
Usunięcie zasobów obsługujących komunikację .....	886
Rady dotyczące narzędzia deploytool i komponentów sterowanych komunikatami .....	886
Określenie typu komponentu .....	886
Konfiguracja właściwości komponentu sterowanego komunikatami .....	887
Rady dotyczące narzędzia deploytool i komponentów wysyłających komunikaty .....	888
Ustalenie nazwy zasobów .....	888
Ustalenie nazwy celu komunikatu .....	889
Ustalenie celu komunikatu .....	889
<b>Rozdział 29. Język zapytań EJB QL .....</b>	<b>891</b>
Terminologia .....	891
Uproszczona składnia EJB QL .....	892
Przykłady zapytań .....	892
Proste zapytania wyszukiwujące .....	893
Zapytania wyszukiwujące odwołujące się do komponentów będących w relacji .....	894
Wyrażenia warunkowe w zapytaniach wyszukiwujących .....	895
Zapytania selekcji .....	896
Kompletna składnia EJB QL .....	897
Symbole notacji BNF .....	897
Gramatyka języka EJB QL w notacji BNF .....	898
Klauzula FROM .....	900
Wyrażenia nawigujące .....	903
Klauzula WHERE .....	905
Klauzula SELECT .....	911
Klauzula ORDER BY .....	913
Ograniczenia języka SQL .....	914
<b>Rozdział 30. Transakcje .....</b>	<b>915</b>
Czym jest transakcja? .....	915
Transakcje zarządzane przez kontener .....	916
Atrybuty transakcji .....	916
Wycofywanie transakcji zarządzanej przez kontener .....	919
Synchronizacja zmiennych instancyjnych komponentu sesyjnego .....	920
Kompilacja przykładu BankBean .....	921
Pakowanie przykładu BankBean .....	922
Metody niedozwolone podczas posługiwania się transakcjami zarządzanymi przez kontener .....	925
Transakcje zarządzane przez komponent .....	925
Transakcje JDBC .....	926
Przygotowanie i uruchomienie przykładu WarehouseBean .....	927
Kompilacja przykładu WarehouseBean .....	927
Pakowanie przykładu WarehouseBean .....	927
Transakcje JTA .....	930
Przygotowanie i uruchomienie przykładu TellerBean .....	931
Kompilacja przykładu TellerBean .....	931
Pakowanie przykładu TellerBean .....	932



	Powrót z metody bez zatwierdzenia transakcji .....	935
	Metody niedozwolone podczas posługiwania się transakcjami zarządzanymi przez komponent .....	935
	Podsumowanie możliwości obsługi transakcji w komponentach encyjnycy .....	936
	Limity czasowe transakcji .....	936
	Poziomy izolacji .....	937
	Aktualizacja wielu baz danych .....	938
	Transakcje w komponentach sieciowych .....	939
<b>Rozdział 31.</b>	<b>Łączenie z zasobami .....</b>	<b>941</b>
	Usługa nazw JNDI .....	941
	Źródła danych i pule połączeń .....	942
	Połączenia z bazą danych .....	943
	Kodowanie połączenia z bazą .....	943
	Określenie nazwy obiektu .....	944
	Tworzenie źródła danych .....	944
	Połączenia z usługą pocztową .....	945
	Uruchomienie przykładu ConfirmBean .....	946
	Połączenia URL .....	948
	Uruchomienie przykładu HTMLReaderBean .....	949
	Dalsze informacje .....	950
<b>Rozdział 32.</b>	<b>Bezpieczeństwo .....</b>	<b>951</b>
	Podstawy .....	951
	Dziedziny, użytkownicy, grupy i role .....	952
	Zarządzanie użytkownikami .....	953
	Ustalanie ról bezpieczeństwa .....	953
	Przyporządkowanie ról użytkownikom i grupom .....	954
	Bezpieczeństwo warstwy sieciowej .....	955
	Ochrona zasobów sieciowych .....	956
	Konfigurowanie wymagań i reguł bezpieczeństwa .....	957
	Konfiguracja bezpiecznego połączenia .....	960
	Bezpieczeństwo programowe w warstwie sieciowej .....	960
	Metody uwierzytelnienia .....	962
	Podstawowe uwierzytelnienie HTTP .....	962
	Uwierzytelnienie na podstawie formularza .....	963
	Uwierzytelnienie na podstawie certyfikatu .....	964
	Uwierzytelnienie wzajemne .....	964
	Uwierzytelnienie typu Digest .....	966
	Konfiguracja uwierzytelnienia .....	966
	Przykład. Uwierzytelnienie za pomocą formularza .....	967
	Instalacja i konfiguracja protokołu SSL .....	974
	Na czym polega protokół bezpiecznej transmisji danych? .....	974
	Certyfikaty cyfrowe .....	975
	Konfiguracja łącznika SSL .....	980
	Bezpieczeństwo usług internetowych i XML .....	983
	Przykład. Podstawowe uwierzytelnienie aplikacji JAX-RPC .....	984
	Przykład. Uwierzytelnienie aplikacji JAX-RPC na podstawie certyfikatu i przy użyciu transmisji HTTP/SSL .....	990
	Bezpieczeństwo warstwy komponentów EJB .....	998
	Deklaracja praw dostępu do metody .....	998
	Konfiguracja bezpieczeństwa na poziomie adresu IOR .....	999
	Bezpieczeństwo programowe w warstwie komponentów EJB .....	1000
	Nieuwierzytelniona nazwa użytkownika .....	1001
	Bezpieczeństwo warstwy aplikacji klienta .....	1001

Bezpieczeństwo korporacyjnej warstwy informacyjnej .....	1002
Logowanie na poziomie kontenera .....	1002
Logowanie na poziomie komponentu .....	1002
Konfiguracja bezpieczeństwa adaptera zasobów .....	1003
Propagowanie tożsamości .....	1004
Konfiguracja propagowanej tożsamości komponentu .....	1004
Konfiguracja uwierzytelnienia klienta .....	1005
Czym jest JACC? .....	1005
Dalsze informacje .....	1005
<b>Rozdział 33. Interfejs Java Message Service .....</b>	<b>1007</b>
Wprowadzenie .....	1007
Na czym polega wymiana komunikatów w obrębie aplikacji? .....	1007
Czym jest interfejs JMS API? .....	1008
Kiedy można korzystać z interfejsu JMS? .....	1009
Jak interfejs JMS współdziała z platformą J2EE? .....	1010
Podstawowe elementy interfejsu JMS .....	1011
Architektura JMS API .....	1011
Dziedziny wymiany komunikatów .....	1012
Pobieranie komunikatów .....	1014
Model programistyczny interfejsu JMS .....	1014
Obiekty administracji .....	1015
Połączenia .....	1017
Sesje .....	1017
Wytwórcy komunikatów .....	1018
Konsumenty komunikatów .....	1019
Komunikaty .....	1021
Obsługa wyjątków .....	1023
Prosta aplikacja klienta JMS .....	1024
Przykład prostego synchronicznego odbioru komunikatów .....	1024
Przykład prostego asynchronicznego odbioru komunikatów .....	1033
Uruchamianie programów-klientów JMS w wielu systemach .....	1037
Niezawodność wymiany w aplikacjach JMS .....	1041
Podstawowe mechanizmy pewności wymiany .....	1042
Zaawansowane mechanizmy pewności wymiany .....	1048
Stosowanie interfejsu JMS w aplikacji J2EE .....	1058
Komponenty sesyjne i encyjne w synchronicznym odbiorze komunikatów .....	1058
Stosowanie komponentów sterowanych komunikatami .....	1060
Zarządzanie transakcjami rozproszonymi .....	1062
Stosowanie interfejsu JMS z klientami aplikacji i komponentami WWW .....	1064
Dalsze informacje .....	1065
<b>Rozdział 34. Interfejs JMS w aplikacjach J2EE — przykłady .....</b>	<b>1067</b>
Aplikacja J2EE korzystająca z JMS z komponentem sesyjnym .....	1068
Pisanie komponentów aplikacji .....	1068
Tworzenie i pakowanie aplikacji .....	1070
Wdrażanie aplikacji .....	1074
Uruchamianie klienta aplikacji .....	1074
Aplikacja J2EE korzystająca z JMS z komponentem encyjnym .....	1075
Aplikacja działu kadr .....	1076
Pisanie komponentów aplikacji .....	1077
Tworzenie i pakowanie aplikacji .....	1079
Instalacja aplikacji .....	1081
Uruchamianie klienta .....	1081
Aplikacja J2EE konsumująca komunikaty ze zdalnego serwera J2EE .....	1082
Przegląd aplikacji .....	1083
Programowanie komponentów aplikacji .....	1084

Tworzenie aplikacji i pakietów .....	1084
Wdrażanie aplikacji .....	1086
Uruchamianie klienta aplikacji .....	1087
Aplikacja J2EE wdrażająca komponent sterowany komunikatami	
na dwóch serwerach J2EE .....	1088
Przegląd aplikacji .....	1088
Programowanie komponentów aplikacji .....	1089
Tworzenie aplikacji i pakietów .....	1091
Wdrażanie aplikacji .....	1093
Uruchamianie klienta aplikacji .....	1094
<b>Rozdział 35. Aplikacja Coffee Break .....</b>	<b>1097</b>
Wspólna część kodu serwerów .....	1098
Usługa JAX-RPC dostawcy .....	1098
Interfejs usługi .....	1099
Implementacja usługi .....	1099
Rejestrowanie usługi dostawcy .....	1100
Usuwanie usługi z rejestru .....	1103
Usługa SAAJ dostawcy .....	1105
Klient SAAJ .....	1106
Usługa SAAJ .....	1112
Serwer Coffee Break .....	1117
Strony JSP .....	1118
Komponenty JavaBeans .....	1119
RetailPriceListServlet .....	1121
Serwer Coffee Break — wersja z JavaServer Faces .....	1121
Strony JSP .....	1122
Komponenty JavaBeans .....	1124
Konfiguracja zasobów .....	1125
Kompilacja, pakowanie, wdrażanie i uruchamianie aplikacji .....	1126
Konfiguracja portu .....	1126
Konfiguracja serwera rejestracji .....	1126
Stosowanie gotowych plików WAR przykładu .....	1127
Kompilacja klas wspólnych aplikacji .....	1127
Kompilacja, pakowanie i wdrażanie usługi JAX-RPC .....	1127
Kompilacja, pakowanie i wdrażanie usługi SAAJ .....	1129
Kompilacja, pakowanie i wdrażanie serwera Coffee Break .....	1130
Kompilacja, pakowanie i wdrażanie serwera Coffee Break	
w wersji wykorzystującej JavaServer Faces .....	1131
Uruchamianie klienta aplikacji Coffee Break .....	1132
Usuwanie aplikacji Coffee Break .....	1134
<b>Rozdział 36. Bezpieczna Kasa Duke'a .....</b>	<b>1135</b>
Komponenty korporacyjne .....	1136
Komponenty sesyjne .....	1136
Komponenty encyjne .....	1139
Klasy pomocnicze .....	1139
Tabele bazy danych .....	1140
Ochrona komponentów korporacyjnych .....	1141
Klient aplikacyjny .....	1142
Klasy i powiązania pomiędzy nimi .....	1143
Klasa BankAdmin .....	1144
Klasa EventHandle .....	1145
Klasa DataModel .....	1146
Klient WWW .....	1149
Strategie projektowe .....	1150
Składowe klienta WWW .....	1151

	Przetwarzanie ządania .....	1153
	Ochrona zasobów klienta WWW .....	1154
	Umiędzynarodowienie aplikacji .....	1156
	Kompilacja, pakowanie, wdrażanie i uruchamianie aplikacji .....	1157
	Konfigurowanie serwerów .....	1158
	Kompilowanie kodu aplikacji .....	1159
	Pakowanie i wdrażanie Bezpiecznej Kasy Duke'a .....	1159
	Przegląd nazw JNDI .....	1164
	Uruchamianie klientów .....	1165
	Uruchamianie klienta aplikacyjnego .....	1165
	Uruchamianie klienta WWW .....	1166
Dodatek A	<b>Kodowanie znaków w języku Java .....</b>	<b>1169</b>
	Dalsze informacje .....	1170
Dodatek B	<b>XML i reszta — alfabet specyfikacji .....</b>	<b>1171</b>
	Standardy podstawowe .....	1172
	SAX .....	1172
	StAX .....	1172
	DOM .....	1173
	JDOM i dom4j .....	1173
	DTD .....	1173
	Przestrzenie nazw .....	1174
	XSL .....	1174
	XSLT (i XPath) .....	1175
	Schematy .....	1175
	XML Schema .....	1176
	RELAX NG .....	1176
	SOX .....	1176
	Schematron .....	1177
	Standardy łączenia i prezentacji dokumentów .....	1177
	Łączenie dokumentów XML .....	1177
	XHTML .....	1178
	Standardy wiedzy .....	1178
	RDF .....	1178
	RDF Schema .....	1179
	XTM .....	1179
	Standardy oparte na XML-u .....	1179
	Standardy dokumentów specjalizowanych .....	1179
	Standardy handlu elektronicznego .....	1180
	Podsumowanie .....	1181
Dodatek C	<b>Wstęp do HTTP .....</b>	<b>1183</b>
	Żądania HTTP .....	1183
	Odpowiedzi HTTP .....	1184
Dodatek D	<b>Architektura Connector platformy J2EE .....</b>	<b>1185</b>
	Adaptory zasobów .....	1185
	Kontrakty adaptera zasobu .....	1186
	Kontrakty systemowe .....	1186
	Kontrakty wyjściowe .....	1188
	Kontrakty wejściowe .....	1188
	Interfejs CCI .....	1189
	Dalsze informacje .....	1190
Dodatek E	<b>Słowniczek .....</b>	<b>1191</b>
	<b>Skorowidz .....</b>	<b>1229</b>

## Rozdział 21.

# Konfigurowanie aplikacji JavaServer Faces

Do obowiązków architekta aplikacji należą:

- Rejestrowanie obiektów pomocniczych dla danej aplikacji w celu udostępnienia ich wszystkim częściom aplikacji.
- Konfigurowanie komponentów pomocniczych i komponentów modelu, aby odwołania do nich na stronie powodowały tworzenie instancji o odpowiednich wartościach.
- Definiowanie reguł nawigacji dla każdej ze stron aplikacji gwarantujące poprawną sekwencję kolejnych stron aplikacji.
- Pakowanie wszystkich stron, obiektów i innych plików aplikacji, aby aplikacja mogła zostać zainstalowana dla dowolnego kontenera.

Zadaniem niniejszego rozdziału jest wyjaśnienie sposobu realizacji wymienionych zadań.

## Plik konfiguracyjny zasobów aplikacji

Technologia JavaServer Faces stosuje przenośny format plików konfiguracyjnych (w postaci dokumentu XML) umożliwiających konfigurowanie zasobów. Architekt aplikacji tworzy jeden lub więcej plików zwanych plikami konfiguracyjnymi zasobów aplikacji, które używają tego formatu do rejestrowania i konfigurowania obiektów, a także definiowania reguł nawigacji. Plik konfiguracyjny zasobów aplikacji nosi zwykle nazwę *faces-config.xml*.

Plik konfiguracyjny zasobów aplikacji musi być zgodny z plikiem *http://java.sun.com/dtd/web-facesconfig\_1\_0.dtd*. Dodatkowo każdy taki plik musi zawierać następujące elementy w poniższym porządku:

- Numer wersji XML:

```
<?xml version="1.0"?>
```

- Deklarację DOCTYPE:

```
<!DOCTYPE faces-config PUBLIC "-//Sun Microsystems, Inc.//DTD Javasever Faces  
Config 1.0//EN" "http://java.sun.com/dtd/web-facesconfig_1_0.dtd">
```

- Znacznik `faces-config` obejmujący wszystkie inne deklaracje:

```
<faces-config>
...
</faces-config>
```

Może istnieć więcej niż jeden plik konfiguracyjny zasobów aplikacji. Implementacja JavaServer Faces znajduje ten plik lub pliki, poszukując:

- Zasobu o nazwie `/META-INF/faces-config.xml` w dowolnym pliku JAR w katalogu aplikacji `/WEB-INF/lib` oraz za pomocą nadrzędnych classloaderów. Jeśli istnieje zasób o takiej nazwie, to zostaje załadowany jako zasób konfiguracji. Metoda ta jest użyteczna w przypadku spakowanej biblioteki zawierającej komponenty i obiekty wyświetlania.
- Parametru inicjalizacji kontekstu, `javax.faces.application.CONFIG_FILES`, który specyfikuje jedną lub więcej ścieżek (oddzielonych przecinkami) prowadzących do wielu plików konfiguracyjnych dla danej aplikacji. Metoda ta powinna być stosowana w przypadku rozbudowanych aplikacji, które delegują odpowiedzialność za konfigurację poszczególnych swoich części do różnych grup architektów aplikacji.
- Zasobu o nazwie `faces-config.xml` w katalogu `/WEB-INF/` aplikacji. W ten sposób używają plików konfiguracyjnych najprostsze aplikacje.

Dostęp do zasobów zarejestrowanych dla aplikacji wymaga od programisty aplikacji użycia instancji klasy `Application`, która jest automatycznie tworzona dla każdej aplikacji. Instancja `Application` działa jako scentralizowana fabryka zasobów zdefiniowanych w pliku XML.

Podczas uruchamiania aplikacji implementacja JavaServer Faces tworzy pojedynczą instancję klasy `Application` i konfiguruje ją, wykorzystując informacje zawarte w pliku konfiguracyjnym zasobów aplikacji.

## Konfigurowanie komponentów

Do tworzenia komponentów pomocniczych używanych przez aplikacje JavaServer Faces i umieszczenia ich w wybranym zakresie używamy mechanizmu tworzenia komponentów zarządzanych. Mechanizm ten konfigurujemy w pliku konfiguracyjnym zasobów aplikacji, używając elementu `managed-bean` do zdefiniowania każdego komponentu. Plik ten zostaje przetworzony podczas uruchamiania aplikacji. Gdy strona odwołuje się do komponentu, to implementacja JavaServer Faces inicjalizuje go zgodnie z konfiguracją zapisaną w pliku konfiguracyjnym zasobów aplikacji.

Korzystając z mechanizmu tworzenia komponentów zarządzanych, możemy:

- Tworzyć komponenty za pomocą jednego, scentralizowanego pliku dostępnego dla całej aplikacji, zamiast warunkowo tworzyć instancje komponentów w różnych częściach aplikacji.
- Indywidualizować właściwości komponentów bez tworzenia dodatkowego kodu.

- Indywidualizować wartości właściwości komponentów podczas tworzenia komponentu bezpośrednio za pomocą pliku konfiguracyjnego.
- Za pomocą elementów `value` nadawać właściwości jednego komponentu zarządzanego wartością będącą wynikiem opracowania wyrażenia wiążącego wartość dla innego komponentu.

W niniejszym podrozdziale przedstawimy sposób inicjalizacji komponentów pomocniczych za pomocą mechanizmu tworzenia komponentów zarządzanych. Sposób implementacji właściwości komponentów pomocniczych wyjaśniliśmy w podrozdziale „Implementacja właściwości komponentu” w rozdziale 19. W podrozdziale „Implementacja metod komponentu pomocniczego”, zamieszczonym w rozdziale 19., omówiliśmy sposób implementacji metod komponentu pomocniczego. Sposób odwołań znaczników komponentów do komponentów zarządzanych przedstawiliśmy w podrozdziale „Wiązanie wartości i instancji komponentów z zewnętrznymi źródłami danych” (rozdział 18.).

## Stosowanie elementu `managed-bean`

Komponenty pomocnicze tworzymy za pomocą elementu `managed-bean` reprezentującego instancję klasy komponentu, która musi istnieć dla danej aplikacji. Element `managed-bean` jest przetwarzany przez implementację JavaServer Faces podczas działania aplikacji. Gdy strona aplikacji odwołuje się do komponentu, to implementacja JavaServer Faces tworzy jego instancję (jeśli nie istnieje żadna instancja komponentu) zgodnie z konfiguracją odpowiedniego elementu.

Poniżej przedstawiamy przykład konfiguracji komponentu zarządzanego dla aplikacji Księgarnia Duke’a:

```
<managed-bean>
  <managed-bean-name> NA </managed-bean-name>
  <managed-bean-class>
    model.ImageArea
  </managed-bean-class>
  <managed-bean-scope> application </managed-bean-scope>
  <managed-property>
    <property-name>shape</property-name>
    <value>poly</value>
  </managed-property>
  ...
</managed-bean-name>
</managed-bean>
```

Element `managed-bean-name` definiuje klucz komponentu używany w danym zakresie. Jeśli tworzymy odwzorowanie komponentu interfejsu użytkownika na komponent pomocniczy, to atrybut `value` znacznika komponentu musi odpowiadać elementowi `managed-bean-name` aż do pierwszego znaku kropki. Na przykład poniższe wyrażenie odpowiada właściwości `shape` instancji `NA` klasy `ImageArea`:

```
value="#{NA.shape}"
```

Część wyrażenia przed znakiem kropki odpowiada elementowi `managed-bean-name` dla komponentu pomocniczego `ImageArea`. Więcej przykładów użycia atrybutu `value` do wiązania komponentów z właściwościami można znaleźć w podrozdziale „Stosowanie znaczników komponentów HTML” zamieszczonym w rozdziale 18.

Element `managed-bean-class` definiuje pełną nazwę klasy komponentu `JavaBean` używaną do tworzenia instancji komponentu. Obowiązkiem programisty aplikacji jest zapewnienie, żeby klasa ta odpowiadała konfiguracji komponentu w pliku konfiguracyjnym zasobów aplikacji. Przykładowo, definicje właściwości muszą zgadzać się ze skonfigurowanymi dla danego komponentu.

Element `managed-bean-scope` definiuje zakres, w którym przechowywany będzie komponent. Możliwe są cztery zakresy: `none`, `request`, `session` lub `application`. Jeśli zdefiniujemy komponent o zakresie `none`, to za każdym odwołaniem do tego komponentu będzie tworzona jego nowa instancja, która nie jest przechowywana w żadnym zakresie. Jedną z sytuacji, w której używamy zakresu `none`, jest odwołanie jednego komponentu zarządzanego do innego komponentu zarządzanego. Jeśli komponent ten powinien być tworzony tylko wtedy, gdy występuje odwołanie do niego, to powinien właśnie posiadać zakres `none`. Przykład inicjalizacji właściwości komponentu zarządzanego można znaleźć w podpunkcie „Inicjalizacja właściwości komponentów zarządzanych”.

Element `managed-bean` może zawierać zero lub więcej elementów `managed-property`, z których każdy odpowiada jednej właściwości zdefiniowanej w klasie komponentu. Elementy są używane do zainicjalizowania wartości właściwości komponentu pomocniczego. Jeśli nie chcemy, aby określona właściwość została zainicjalizowana podczas tworzenia instancji komponentu pomocniczego, to nie umieszczamy dla niej definicji `managed-property` w pliku konfiguracyjnym zasobów aplikacji.

Jeśli element `managed-bean` nie zawiera innych elementów `managed-bean`, to może zawierać jeden element `map-entries` lub `list-entries`. Element `map-entries` konfiguruje zbiór komponentów będących instancjami typu `Map`. Element `list-entries` konfiguruje zbiór komponentów będących instancjami typu `List`.

Aby stworzyć odwzorowanie na właściwość zdefiniowaną za pomocą elementu `managed-property`, musimy zagwarantować, że część wyrażenia umieszczonego w atrybucie `value` poprzedzająca znak kropki zgadza się z elementem `property-name` elementu `managed-property`. We wcześniejszym przykładzie właściwość `shape` została zainicjalizowana wartością `poly`. W następnym podrozdziale omówimy bardziej szczegółowo posługiwanie się elementem `managed-property`.

## Inicjalizacja właściwości za pomocą elementu `managed-property`

Element `managed-property` musi zawierać element `property-name`, który musi zgadzać się z nazwą danej właściwości w klasie komponentu. Element `managed-property` musi zawierać również jeden ze zbioru elementów (przedstawionych w tabeli 21.1) definiujący wartość właściwości. Wartość ta musi być takiego samego typu, jaki został zdefiniowany dla tej właściwości w klasie komponentu. Wybór elementu definiującego wartość zależy od typu właściwości zdefiniowanego w klasie komponentu. Wszystkie elementy używane do inicjalizacji wartości przedstawione zostały w tabeli 21.1.

Przykład inicjalizacji właściwości typu `String` za pomocą elementu `value` przedstawiliśmy w punkcie „Stosowanie elementu `managed-bean`”. Elementu podrzędnego `value` możemy również używać do inicjalizowania wartości typów prostych i innych typów referencyjnych.



**Tabela 21.1.** Elementy podrzędne elementu *managed-property* definiujące wartość właściwości

Element	Wartość, którą definiuje
<code>list-entries</code>	Definiuje wartości listy
<code>map-entries</code>	Definiuje wartości mapy
<code>null-value</code>	Jawnie nadaje właściwości wartość <code>null</code>
<code>value</code>	Definiuje pojedynczą wartość, na przykład typu <code>String</code> lub <code>int</code> , bądź wyrażenie JavaServer Faces

Pozostała część tego podrozdziału opisuje sposób użycia elementu podrzędnego `value` i innych elementów podrzędnych do inicjalizacji właściwości typu `java.util.Map` tablicy oraz `Collection`, a także parametrów inicjalizacji.

## Odwołania do parametru inicjalizacji

Kolejną ważną zaletą mechanizmu tworzenia komponentów zarządzanych jest możliwość odwołań do obiektów niejawnych przez właściwości komponentu.

Załóżmy, że pewna strona akceptuje dane wprowadzane przez użytkownika, w tym jego adres. Załóżmy również, że większość użytkowników mieszka w rejonie o takim samym kodzie pocztowym. Komponent kodu pocztowego może wyświetlić ten kod, przechodząc go w obiekcie niejawnym i odwołując się do niego podczas wyświetlania strony.

Kod możemy zapamiętać jako początkową, domyślną wartość niejawnego obiektu kontekstu `initParam`, dodając parametr kontekstu do aplikacji i nadając mu wartość za pomocą programu *deploytool*. Na przykład, aby nadać parametrowi kontekstu o nazwie `defaultAreaCode` wartość `650`, uruchomimy *deploytool*, wybierzemy naszą aplikację z drzewa, wybierzemy zakładkę `Context`, dodamy nowy parametr kontekstu i wprowadzimy `defaultAreaCode` w polu *Coded Parameter* i `650` w polu *Value*.

Następnie musimy stworzyć deklarację *managed-bean*, która skonfiguruje właściwość odwołującą się do tego parametru:

```
<managed-bean>
  <managed-bean-name>customer</managed-bean-name>
  <managed-bean-class>CustomerBean</managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
  <managed-property>
    <property-name>areaCode</property-name>
    <value-ref>initParam.defaultAreaCode</value-ref>
  </managed-property>
  ...
</managed-bean>
```

Dostęp do kodu podczas wyświetlania strony wymaga odwołania się do właściwości przez atrybut `value` znacznika komponentu `area`:

```
<h:inputText id=area value="#{customer.areaCode}"
```

Pobieranie wartości z innych obiektów niejawnych odbywa się w podobny sposób. Listę obiektów niejawnych zawiera tabela 18.9.

## Inicjalizacja właściwości będących mapą

Element `map-entries` użyty wewnątrz elementu `managed-property` służy inicjalizacji wartości właściwości komponentu pomocniczego, która posiada typ `java.util.Map`. Poniżej przedstawiamy definicję elementu `map-entries` pochodzącą z pliku `web-facesconfig_1_0.dtd` umieszczonego pod adresem [http://java.sun.com/dtd/web-facesconfig\\_1\\_0.dtd](http://java.sun.com/dtd/web-facesconfig_1_0.dtd) i definiującego plik konfiguracyjny zasobów aplikacji:

```
<!ELEMENT map-entries (key-class?, value-class?, map-entry*) >
```

Definicja ta pokazuje, że element `map-entries` może zawierać opcjonalny element `key-class`, opcjonalny element `value-class` oraz zero lub więcej elementów `map-entry`.

Element `map-entry` zdefiniowany został następująco:

```
<!ELEMENT map-entry (key, (null-value|value )) >
```

Poniżej przedstawiamy przykład użycia elementu `map-entries`:

```
<managed-bean>
...
<managed-property>
  <property-name>prices</property-name>
  <map-entries>
    <map-entry>
      <key>My Early Years: Growing Up on *7</key>
      <value>30.75</value>
    </map-entry>
    <map-entry>
      <key>Web Servers for Fun and Profit</key>
      <value>40.75</value>
    </map-entry>
  </map-entries>
</managed-property>
</managed-bean>
```

Mapa utworzona przez powyższy element `map-entries` zawiera dwie pozycje. Domyślnie wszystkie klucze i wartości mapy przekształcane są na typ `java.lang.String`. Chcąc używać kluczy innego typu, musimy zagnieździć element `key-class` wewnątrz elementu `map-entries`:

```
<map-entries>
  <key-class>java.math.BigDecimal</key-class>
  ...
</map-entries>
```

Powyższa deklaracja spowoduje przekształcenie wszystkich kluczy na typ `java.math.BigDecimal`. Oczywiście sami musimy zadbać o to, by przekształcenie takie było możliwe. Klucz użyty w poprzednim przykładzie jest typu `String` i nie może zostać przekształcony na typ `java.math.BigDecimal`.

Jeśli chcemy, aby również wszystkie wartości na mapie były innego typu, to po elemencie `key-class` umieszczamy element `value-class`:

```
<map-entries>
  <key-class>int</key-class>
  <value-class>java.math.BigDecimal</value-class>
  ...
</map-entries>
```

Zwróćmy uwagę, że element ten konfiguruje typ tylko wszystkich elementów podrzędnych `value`.

Pierwszy element `map-entry` w ostatnim przykładzie zawiera element podrzędny `value`. Element ten definiuje pojedynczą wartość, która zostanie przekształcona na typ określony przez komponent.

Drugi element `map-entry` definiuje element `value`, który odwołuje się do właściwości innego komponentu. Odwołanie do innego komponentu przez właściwość komponentu pozwala tworzyć system złożony z precyzyjnie zdefiniowanych obiektów. Na przykład obiekt obsługi formularza należący do zakresu żądania może posiadać wskaźnik na obiekt obsługi bazy danych umieszczony w zakresie aplikacji. Oba te obiekty razem mogą obsługiwać formularz. Zauważmy przy tym, że umieszczenie odwołania do innego komponentu spowoduje utworzenie jego instancji, jeśli ona jeszcze nie istnieje.

Zamiast używać elementu `map-entries`, można również przypisać całą mapę, wykorzystując element `value` zawierający wyrażenie, którego typem jest mapa.

## Inicjalizacja właściwości będących tablicami lub listami

Element `values` używany jest do zainicjalizowania wartości właściwości typu tablicowego lub typu `List`. Każda pojedyncza wartość tablicy lub listy zostaje zainicjalizowana za pomocą elementu `value` lub `null-value`. Oto przykład:

```
<managed-bean>
  ...
  <managed-property>
    <property-name>books</property-name>
    <values>
      <value-type>java.lang.String</value-type>
      <value>Web Servers for Fun and Profit</value>
      <value>#{myBooks.bookId[3]}</value>
      <null-value/>
    </values>
  </managed-property>
</managed-bean>
```

Powyższy przykład inicjalizuje właściwość typu tablicowego lub `List`. Typ utworzonej struktury danych jest określony przez typ odpowiadającej jej właściwości komponentu. Element `values` definiuje listę wartości należących do tablicy lub listy. Element `value` określa pojedynczą wartość tablicy bądź listy i może odwoływać się do właściwości innego komponentu. Element `null-value` spowoduje wywołanie metody `setBooks` z argumentem równym `null`. Wartość `null` nie może zostać podana dla właściwości będącej typu prostego języka Java, na przykład `int` lub `boolean`.

## Inicjalizacja właściwości komponentów zarządzanych

Czasami może zdarzyć się sytuacja, w której będziemy chcieli stworzyć komponent odwołujący się do innych komponentów zarządzanych, tworząc w ten sposób graf lub drzewo komponentów. Załóżmy na przykład, że chcemy stworzyć komponent reprezentujący informacje o kliencie, w tym adres do korespondencji i adres zameldowania, które także są komponentami. Przedstawiona poniżej deklaracja managed-bean tworzy instancję klasy CustomerBean, która posiada dwie właściwości typu AddressBean: jedną reprezentującą adres do korespondencji i drugą reprezentującą adres zameldowania. W ten sposób powstaje proste drzewo, którego korzeniem jest komponent CustomerBean, a liśćmi dwa komponenty AddressBean.

```
<managed-bean>
  <managed-bean-name>customer</managed-bean-name>
  <managed-bean-class>
    com.mycompany.mybeans.CustomerBean
  </managed-bean-class>
  <managed-bean-scope> request </managed-bean-scope>
  <managed-property>
    <property-name>mailingAddress</property-name>
    <value>addressBean</value>
  </managed-property>
  <managed-property>
    <property-name>streetAddress</property-name>
    <value>addressBean</value>
  </managed-property>
  <managed-property>
    <property-name>customerType</property-name>
    <value>New</value>
  </managed-property>
</managed-bean>
<managed-bean>
  <managed-bean-name>addressBean</managed-bean-name>
  <managed-bean-class>
    com.mycompany.mybeans.AddressBean
  </managed-bean-class>
  <managed-bean-scope> none </managed-bean-scope>
  <managed-property>
    <property-name>street</property-name>
    <null-value/>
  <managed-property>
    ...
</managed-bean>
```

Pierwsza deklaracja komponentu CustomerBean (czyli zawierająca element managed-bean-name customer) tworzy komponent CustomerBean dostępny w zakresie żądania. Komponent ten posiada dwie właściwości: mailingAddress i streetAddress. Właściwości te używają elementu value, odwołując się do komponentu o nazwie addressBean.

Druga deklaracja komponentu zarządzanego definiuje komponent klasy AddressBean, ale nie tworzy go, ponieważ element managed-bean-scope definiuje zakres none. Przypomnijmy, że zakres none oznacza, że komponent tworzony jest tylko wtedy, gdy wystąpi odwołanie do niego. Ponieważ zarówno właściwość mailingAddress, jak i streetAddress odwołują się do addressBean za pomocą elementu value, to podczas tworzenia komponentu klasy CustomerBean zostają również utworzone dwie instancje klasy AddressBean.

Tworząc obiekt odwołujący się do innego obiektu, należy unikać odwoływania się do obiektów o krótszym okresie życia, ponieważ uniemożliwia to odzyskanie zasobów zakresu, który przestaje istnieć. Na przykład obiekt należący do zakresu sesji nie może odwoływać się do obiektu istniejącego w zakresie żądania. Obiekty należące do zakresu `none` nie mają określonego czasu istnienia i dlatego mogą odwoływać się jedynie do obiektów tego samego zakresu. Dozwolone odwołania przedstawione zostały w tabeli 21.2.

**Tabela 21.2.** *Dozwolone odwołania pomiędzy obiektami różnych zakresów*

Obiekt należący do zakresu	Może wskazywać obiekt należący do zakresu
<code>none</code>	<code>none</code>
<code>application</code>	<code>none</code> , <code>application</code>
<code>session</code>	<code>none</code> , <code>application</code> , <code>session</code>
<code>request</code>	<code>none</code> , <code>application</code> , <code>session</code> , <code>request</code>

## Inicjalizacja map i list

Oprócz konfigurowania właściwości typu `Map` i `List` można również konfigurować bezpośrednio obiekty typu `Map` oraz `List` i odwoływać się do nich bezpośrednio ze znacznika zamiast do właściwości obudowujących obiekty typu `Map` i `List`.

Aplikacja Księgarnia Duke'a konfiguruje obiekt typu `List`, aby zainicjalizować listę darmowych biuletynów, spośród których użytkownik może subskrybować wybrane na stronie `bookcashier.jsp`:

```
<managed-bean>
...
<managed-bean-name>newsletters</managed-bean-name>
  <managed-bean-class>
    java.util.ArrayList
  </managed-bean-class>
  <managed-bean-scope>application</managed-bean-scope>
  <list-entries>
    <value-class>javax.faces.model.SelectItem</value-class>
    <value>#{newsletter0}</value>
    <value>#{newsletter1}</value>
    <value>#{newsletter2}</value>
    <value>#{newsletter3}</value>
  </list-entries>
</managed-bean>
<managed-bean>
  <managed-bean-name>newsletter0</managed-bean-name>
  <managed-bean-class>
    javax.faces.model.SelectItem
  </managed-bean-class>
  <managed-bean-scope>none</managed-bean-scope>
  <managed-property>
    <property-name>label</property-name>
    <value>Duke's Quarterly</value>
  </managed-property>
</managed-bean>
```

```

    <property-name>value</property-name>
    <value>200</value>
  </managed-property>
</managed-bean>
...

```

Powyższa konfiguracja inicjalizuje obiekt typu `List` o nazwie `newsletters`. Lista ta składa się z instancji typu `SelectItem` będących również komponentami zarządzanymi. Więcej informacji na temat klasy `SelectItem` można znaleźć w punkcie „Komponenty `UISelectItem`, `UISelectItems` i `UISelectItemGroup`” zamieszczonym w rozdziale 18. Zwróćmy uwagę, że, w przeciwieństwie do przykładu zamieszczonego w podpunkcie „Inicjalizacja właściwości będących mapą”, tym razem lista biuletynów nie jest właściwością komponentu zarządzanego (nie jest obudowana elementem `managed-property`). Lista ta jest teraz sama komponentem zarządzanym.

## Rejestracja komunikatów

Jeśli chcemy używać komunikatów niestandardowych, to musimy udostępnić je aplikacji podczas jej uruchamiania. Możemy to osiągnąć na dwa sposoby: umieszczając komunikat w kolejce instancji `FacesContext` (patrz podrozdział „Lokalizacja” w rozdziale 19.) lub rejestrując komunikat w pliku konfiguracyjnym zasobów aplikacji.

Poniżej przedstawiamy fragment pliku rejestrującego komunikaty dla aplikacji Księgarnia Duke’a:

```

<application>
  <message-bundle>
    resource.ApplicationMessages
  </message-bundle>
  <locale-config>
    <default-locale>en</default-locale>
    <supported-locale>de</supported-locale>
    <supported-locale>fr</supported-locale>
    <supported-locale>es</supported-locale>
  </locale-config>
</application>

```

Powyższy zbiór elementów spowoduje umieszczenie w instancji `Application` komunikatów znajdujących się w podanym zestawie `ResourceBundle` o nazwie `resources.ApplicationMessages`.

Element `message-bundle` reprezentuje zbiór zlokalizowanych komunikatów i musi zawierać pełną ścieżkę dostępu do zestawu `ResourceBundle` zawierającego zlokalizowane komunikaty, `resource.ApplicationMessages` w tym przypadku.

Element `locale-config` zawiera listę obsługiwanych lokalizacji, w tym lokalizacji domyślnej. Element ten umożliwi systemowi ustalenie właściwej lokalizacji na podstawie konfiguracji języka dla przeglądarki. Aplikacja Księgarnia Duke’a umożliwi ręczny wybór lokalizacji i w związku z tym zastosowanie elementu `locale-config` nie jest w jej przypadku konieczne.

Elementy `supported-locale` i `default-locale` akceptują dwuliterowe kody zdefiniowane przez standard ISO-639 (patrz <http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt>). Warto zawsze sprawdzić, czy rzeczywiście umieściliśmy w zestawie `ResourceBundle` komunikaty dla lokalizacji, które określiliśmy za pomocą tych elementów.

Dostęp do zlokalizowanego komunikatu umieszczonego w zestawie zasobów wymaga od programisty aplikacji użycia odpowiedniego klucza (patrz podrozdział „Lokalizacja” w rozdziale 19.).

## Rejestracja niestandardowego walidatora

Jeśli programista aplikacji dostarcza własnej implementacji interfejsu `Validator`, to musi zarejestrować niestandardowy walidator w pliku konfiguracyjnym zasobów aplikacji za pomocą elementu `validator`:

```
<validator>
  ...
  <validator-id>FormatValidator</validator-id>
  <validator-class>validators.FormatValidator</validator-
class>
  <attribute>
    ...
    <attribute-name>formatPatterns</attribute-name>
    <attribute-class>java.lang.String</attribute-class>
  </attribute>
</validator>
```

Elementy podrzędne `validator-id` i `validator-class` są obowiązkowe. Element `validator-id` reprezentuje identyfikator, pod którym zostanie zarejestrowana klasa walidatora. Identyfikator ten jest używany przez klasę znacznika odpowiadającą niestandardowemu znacznikowi `validator`.

Element `validator-class` reprezentuje pełną nazwę klasy implementującej interfejs `Validator`.

Element `attribute` identyfikuje atrybut związany z implementacją interfejsu `Validator`. Posiada obowiązkowe elementy podrzędne `attribute-name` i `attribute-class`. Element `attribute-name` odnosi się do nazwy atrybutu określonej przez znacznik `validator`, natomiast `attribute-class` identyfikuje typ wartości związanej z atrybutem.

Sposób implementacji interfejsu `Validator` został omówiony w podrozdziale „Implementacja niestandardowego walidatora” (rozdział 19.).

Sposób odwołań do walidatora na stronach aplikacji wyjaśniliśmy w punkcie „Stosowanie niestandardowego walidatora” (rozdział 18.).

## Rejestracja niestandardowego konwertera

Podobnie jak w przypadku niestandardowego walidatora również niestandardowy konwerter wymaga rejestracji. Poniżej przedstawiamy konfigurację elementu `converter` dla konwertera `CreditCardConverter` używanego przez aplikację Księgarnia Duke'a:

```
<converter>
  <description>
    Rejestruje implementację konwertera,
    converters.CreditCardConverter używając
    identyfikatora creditcard.
  </description>
  <converter-id>creditcard</converter-id>
  <converter-class>
    converters.CreditCardConverter
  </converter-class>
</converter>
```

Element `converter` reprezentuje implementację interfejsu `Converter` i zawiera obowiązkowe elementy podrzędne `converter-id` i `converter-class`.

Element `converter-id` określa identyfikator używany przez atrybut `converter` znacznika komponentu interfejsu użytkownika w celu zastosowania konwertera do przekształcenia danych komponentu. Przykład odwołania do niestandardowego konwertera przez znacznik komponentu zamieściliśmy w punkcie „Stosowanie niestandardowego konwertera” w rozdziale 18.

Element `converter-class` identyfikuje klasę implementującą interfejs `Converter`.

Sposób implementacji niestandardowego konwertera omówiliśmy w podrozdziale „Implementacja niestandardowego konwertera” w rozdziale 19.

## Konfigurowanie reguł nawigacji

W podrozdziale „Model nawigacji” zamieszczonym w rozdziale 17. wyjaśniliśmy, że konfigurowanie nawigacji polega na utworzeniu zbioru reguł umożliwiających wybór kolejnej strony na skutek kliknięcia przycisku lub łącza. Reguły nawigacji definiuje się w pliku konfiguracyjnym zasobów aplikacji.

Każda reguła nawigacji określa sposób przejścia od określonej strony do zbioru innych stron. Implementacja `JavaServer Faces` wybiera odpowiednią regułę nawigacji dla aktualnie wyświetlanej strony.

Po wybraniu właściwej reguły nawigacji wybór kolejnej strony zależy od metody akcji wywołanej na skutek kliknięcia komponentu oraz wyniku logicznego zwróconego przez tę metodę lub podanego przez znacznik komponentu.

Wynik logiczny może być dowolnie zdefiniowany przez programistę aplikacji. W tabeli 21.3 przedstawione zostały najczęściej używane wyniki.



**Tabela 21.3.** *Typowe łańcuchy wyników logicznych*

Wynik	Znaczenie
success	Wszystko przebiegło poprawnie. Przejdź do kolejnej strony.
failure	Wystąpił błąd. Przejdź do strony błędu.
logon	Użytkownik powinien się najpierw zalogować. Przejdź do strony logowania.
no results	Brak wyników wyszukiwania. Przejdź ponownie do strony wyszukiwania.

Zwykle metoda akcji przetwarza dane formularza bieżącej strony. Może na przykład sprawdzić, czy nazwa użytkownika i hasło wprowadzone w formularzu zgadzają się z zapisanymi w pliku. Jeśli tak, to metoda zwraca wynik `success`. W przeciwnym razie zwraca `failure`. Przykład ten pokazuje, że na ustalenie następnej strony ma wpływ metoda przetwarzająca akcję oraz zwracany przez nią wynik logiczny.

Poniżej przedstawiamy regułę nawigacji, która mogłaby zostać użyta w omówionym właśnie przykładzie:

```
<navigation-rule>
  <from-view-id>/logon.jsp</from-view-id>
  <navigation-case>
    <from-action>#{LogonForm.logon}</from-action>
    <from-outcome>success</from-outcome>
    <to-view-id>/storefront.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-action>#{LogonForm.logon}</from-action>
    <from-outcome>failure</from-outcome>
    <to-view-id>/logon.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```

Reguła ta definiuje możliwe sposoby nawigacji dla strony *logon.jsp*. Każdy element `navigation-case` definiuje osobny przypadek ścieżki nawigacji dla tej strony. Pierwszy element `navigation-case` oznacza, że jeśli metoda `LogonForm.logon` zwróci wynik `success`, to należy przejść do strony *storefront.jsp*. Drugi element `navigation-case` sugeruje, że gdy metoda `LogonForm.logon` zwróci wynik `failure`, to wyświetlona zostanie strona *logon.jsp*.

Konfiguracja nawigacji dla danej aplikacji jest zbiorem reguł nawigacji. Każda taka reguła jest zdefiniowana przez element `navigation-rule` w pliku *faces-config.xml*.

Reguły nawigacji zastosowane dla aplikacji Księgarnia Duke'a są bardzo proste. Poniżej przedstawiamy dwie bardziej skomplikowane reguły, które mogłyby zostać użyte dla tej aplikacji:

```
<navigation-rule>
  <from-view-id>/catalog.jsp</from-view-id>
  <navigation-case>
    <from-outcome>success</from-outcome>
    <to-view-id>/bookcashier.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>out of stock</from-outcome>
    <from-action>
```

```

        #{catalog.buy}
    </from-action>
    <to-view-id>/outofstock.jsp</to-view-id>
</navigation-case>
<navigation-case>
    <from-outcome>error</from-outcome>
    <to-view-id>/error.jsp</to-view-id>
</navigation-case>
</navigation-rule>

```

Pierwsza reguła nawigacji w tym przykładzie oznacza, że aplikacja może przejść od strony *catalog.jsp* do strony:

- *bookcashier.jsp*, jeśli zamawiana pozycja jest na stanie,
- *outofstock.jsp*, gdy zamawianej pozycji nie ma na stanie.

Druga reguła nawigacji oznacza, że wystąpienie błędu spowoduje przejście od dowolnej strony do strony *error.jsp*.

Każdy element `navigation-rule` odpowiada jednemu identyfikatorowi drzewa komponentów zdefiniowanemu przez opcjonalny element `from-view-id`. Oznacza to, że każda reguła definiuje wszystkie możliwe sposoby nawigacji dla określonej strony aplikacji. Jeśli element `from-view-id` nie został zdefiniowany, to reguły nawigacji zdefiniowane przez element `navigation-rule` odnoszą się do wszystkich stron aplikacji. Element `from-view-id` umożliwia również stosowanie wzorców. Na przykład przedstawiony poniżej element `from-view-id` oznacza, że reguła nawigacji odnosi się do wszystkich stron umieszczonych w katalogu *books*:

```
<from-view-id>/books/*</from-view-id>
```

Przedstawiony przykład reguł nawigacji pokazuje, że element `navigation-rule` może zawierać zero lub więcej elementów `navigation-case`. Element `navigation-case` definiuje zbiór kryteriów. Jeśli zostaną one spełnione, to aplikacja przejdzie do strony zdefiniowanej przez element `to-view-id` zagnieżdżony w tym samym elemencie `navigation-case`.

Kryteria nawigacji definiowane są za pomocą opcjonalnych elementów `from-outcome` i `from-action`. Element `from-outcome` definiuje wynik logiczny, na przykład `success`. Element `from-action` używa wyrażenia wiążącego metodę do odwołania się do metody akcji zwracającej łańcuch `String` będący wynikiem logicznym.

Elementy `navigation-case` porównywane są z wynikiem i wyrażeniem wiążącym metodę w następującej kolejności:

- Przypadki określające wartość `from-outcome` i `from-action`. Oba te elementy mogą zostać użyte jeśli metoda akcji zwraca różne wyniki w oparciu o wykonywane przez nią operacje.
- Przypadki określające jedynie wartość `from-outcome`. Element `from-outcome` musi zgadzać się albo z wynikiem zdefiniowanym przez atrybut `action` komponentu `UICommand` albo z wynikiem zwróconym przez metodę, do której odwołuje się ten komponent.
- Przypadki określające jedynie wartość `from-action`. Wartość ta musi zgadzać się z wyrażeniem `action` określonym przez znacznik komponentu.

Jeśli spełniony zostanie jeden z tych przypadków, to do wyświetlania zostanie wybrane drzewo komponentów zdefiniowane przez element `to-view-id`.

Sposób użycia atrybutu `action` znacznika komponentu odwołującego się do metody akcji wyjaśniliśmy w punkcie „Odwołanie do metody nawigacji” (rozdział 18.). Sposób implementacji metody akcji przedstawiliśmy w punkcie „Implementacja metody nawigacji” w rozdziale 19.

## Rejestracja niestandardowej klasy wyświetlania w pakiecie wyświetlania

Pakiet wyświetlania definiuje dla każdego obsługiwanego komponentu interfejsu użytkownika zbiór obiektów `Renderer` wyświetlających komponent w różny sposób u klienta obsługiwanego przez ten pakiet. Na przykład standardowy komponent klasy `UISelectOne` definiuje komponent, który umożliwia użytkownikowi wybór jednego elementu ze zbioru elementów. Komponent ten może być wyświetlany przez klasę wyświetlania `Listbox`, `Menu` lub `Radio`. Każda z tych klas tworzy inny wygląd komponentu. Klasa `Listbox` tworzy menu, które może wyświetlać cały zbiór wartości. Klasa `Menu` wyświetla jednocześnie tylko podzbiór wszystkich elementów. Klasa `Radio` wyświetla zbiór przycisków wyboru.

Gdy programista aplikacji tworzy niestandardową klasę wyświetlania (patrz „Delegowanie wyświetlania do klasy wyświetlania” w rozdziale 20.), to musi zarejestrować ją we właściwym pakiecie wyświetlania. Ponieważ niestandardowy komponent mapy graficznej został zaimplementowany jako komponent HTML, to klasa wyświetlania `AreaRenderer` (a także `MapRenderer`) powinna być zarejestrowana w pakiecie wyświetlania HTML.

Klasę wyświetlania rejestrujemy w pliku konfiguracyjnym zasobów aplikacji za pomocą elementu `render-kit`. Poniżej przedstawiamy przykład konfiguracji takiego elementu dla klasy `AreaRenderer` należącej do aplikacji Księgarnia Duke’a:

```
<render-kit>
  <renderer>
    <renderer-type>DemoArea</renderer-type>
    <renderer-class>
      renderkit.AreaRenderer
    </renderer-class>
    <attribute>
      <attribute-name>onmouseout</attribute-name>
      <attribute-class>java.lang.String</attribute-class>
    </attribute>
    <attribute>
      <attribute-name>onmouseover</attribute-name>
      <attribute-class>java.lang.String</attribute-class>
    </attribute>
    <attribute>
      <attribute-name>styleClass</attribute-name>
      <attribute-class>java.lang.String</attribute-class>
    </attribute>
    <supported-component-class>
      <component-class>
        components.AreaComponent
      </component-class>
    </supported-component-class>
  </renderer>
</render-kit>
```

```

        </component-class>
    </supported-component-class>
</renderer>
...

```

Element `render-kit` reprezentuje implementację `RenderKit`. Jeśli element ten nie zostanie wyspecyfikowany, to domyślnie zakładany jest pakiet wyświetlania HTML. Element `renderer` reprezentuje implementację `Renderer`. Zagnieżdżając element `renderer` wewnątrz elementu `render-kit`, rejestrujemy klasę wyświetlania w pakiecie wyświetlania `RenderKit` związanym z elementem `render-kit`.

Sposób użycia elementu `renderer-type` przez obiekt obsługi znacznika wyjaśniamy w następnym podrozdziale. Element `renderer-class` określa pełną nazwę klasy implementującej interfejs `Renderer`.

Elementy `component-family` i `render-type` są używane przez komponent do wyszukania odpowiedniego obiektu wyświetlania. Identyfikator `component-family` musi zgadzać się z wartością zwracaną przez metodę `getFamily` klasy komponentu. Identyfikator `render-type` musi zgadzać się z wartością zwracaną przez metodę `getRendererType` klasy obsługi znacznika. Element `attribute` nie jest używany podczas wykonania aplikacji, a jedynie dostarcza programom narzędziowym informacji o atrybutach obsługiwanych przez implementację `Renderer`.

## Rejestracja niestandardowego komponentu

Niestandardowe komponenty również wymagają rejestracji w pliku konfiguracyjnym zasobów aplikacji. Poniżej przedstawiamy element `component` rejestrujący komponent klasy `AreaComponent`:

```

<component>
  <component-type>DemoArea</component-type>
  <component-class>
    components.AreaComponent
  </component-class>
  <property>
    <property-name>alt</property-name>
    <property-class>java.lang.String</property-class>
  </property>
  <property>
    <property-name>coords</property-name>
    <property-class>java.lang.String</property-class>
  </property>
  <property>
    <property-name>shape</property-name>
    <property-class>java.lang.String</property-class>
  </property>
  <component-extension>
    <component-family>Area</component-family>
    <renderer-type>DemoArea</renderer-type>
  </component-extension>
</component>

```

Element `component-type` określa nazwę, pod którą zarejestrowany zostanie komponent. Inne obiekty, odwołując się do komponentu, będą używać właśnie tej nazwy. Element `component-class` określa pełną nazwę klasy komponentu. Elementy `property` specyfikują właściwości komponentu i ich typy.

Element `component-extension` identyfikuje zbiór komponentów i ich obiektów wyświetlania. Identyfikator `component-family` musi zgadzać się z wartością zwracaną przez metodę `getFamily` komponentu. Identyfikator `renderer-type` musi zgadzać się z wartością zwracaną przez metodę `getRendererType` obiektu obsługi znacznika. Takie rozwiązanie umożliwia wyświetlanie komponentu przez różne obiekty wyświetlania i wyświetlanie różnych komponentów przez ten sam obiekt wyświetlania.

## Podstawowe wymagania aplikacji JavaServer Faces

Oprócz poprawnego skonfigurowania aplikacji musimy spełnić także inne wymagania związane z odpowiednim spakowaniem wymaganych plików i dostarczeniem pliku deskryptora instalacji. W niniejszym podrozdziale omówimy sposób wykonania tych zadań.

Aplikacja JavaServer Faces musi być zgodna ze specyfikacją Servlet 2.3 (lub nowszą) oraz specyfikacją JavaServer Pages 1.2 (lub nowszą). Wszystkie aplikacje zgodne z tymi specyfikacjami są pakowane w pliku WAR, który musi spełniać określone wymagania, aby możliwe było jego wykonanie przez różne kontenery. Plik WAR aplikacji JavaServer Faces musi zawierać co najmniej:

- deskryptor instalacji aplikacji internetowej o nazwie *web.xml* konfigurujący zasoby wymagane przez aplikację internetową,
- zbiór plików JAR zawierających podstawowe klasy,
- zbiór klas aplikacji, stron JavaServer Faces i innych wymaganych zasobów takich jak pliki graficzne,
- plik konfiguracyjny zasobów aplikacji.

Plik WAR ma zwykle następującą strukturę katalogów:

```
index.html
strony JSP
WEB-INF/
  web.xml
  faces-config.xml
  deskryptory bibliotek znaczników (opcjonalne)
  classes/
    pliki klas
    pliki Properties
  lib/
    pliki JAR
```

Plik *web.xml* (deskryptor instalacji), zbiór plików JAR i zbiór plików aplikacji muszą być umieszczone w katalogu *WEB-INF* pliku WAR. Zwykle do skompilowania klas aplikacji używa się programu *asant*. Pliki pakuje się do pliku WAR za pomocą programu *deploytool*, który umożliwia również instalację pliku WAR.

Programy *asant* i *deploytool* są dołączone do serwera Sun Java System Application Server Platform Edition 8. Sposób tworzenia pliku WAR przez program *asant* konfigurujemy za pomocą pliku *build.xml*. Każdy z przykładów omawianych w tej książce posiada własny plik *build.xml*, na którym można się wzorować, tworząc własny plik.

## Konfigurowanie aplikacji za pomocą programu *deploytool*

Aplikacje internetowe są konfigurowane za pomocą elementów umieszczonych w deskrytorze instalacji aplikacji internetowej. Program *deploytool* generuje taki deskrytor, gdy tworzymy plik WAR i umieszcza w nim elementy, gdy tworzymy komponenty aplikacji i związane z nimi klasy. Elementy te możemy modyfikować za pomocą inspektorów związanych z plikami WAR.

Deskrytor instalacji dla aplikacji JavaServer Faces musi konfigurować:

- serwlet używany do przetwarzania żądań JavaServer Faces,
- odwzorowanie serwletu przetwarzającego żądania,
- ścieżkę dostępu do pliku konfiguracyjnego zasobów jeśli nie został on umieszczony w domyślnym katalogu.

Deskrytor instalacji może również:

- określać miejsce przechowywania stanu komponentów,
- ograniczać dostęp do stron zawierających znaczniki JavaServer Faces,
- włączać kontrolę poprawności dokumentów XML,
- weryfikować niestandardowe obiekty.

W tym podrozdziale przedstawimy więcej szczegółów związanych z realizacją wymienionych zadań za pomocą programu *deploytool*.

### Identyfikacja serwletu przetwarzającego żądania

Jedno z wymagań odnośnie aplikacji JavaServer Faces polega na tym, aby wszystkie żądania dotyczące komponentów aplikacji przechodziły przez serwlet `FacesServlet`. Instancja klasy `FacesServlet` zarządza cyklem przetwarzania żądań przez aplikację internetową i inicjalizuje zasoby wymagane przez technologię JavaServer Faces. Aby spełnić te wymagania, należy:

1. W oknie dialogowym *Edit Contents* kreatora *Web Component* dodać do pliku WAR plik *jsf-api.jar* znajdujący się w katalogu `<J2EE_HOME>/lib/`. Plik ten jest konieczny dla zapewnienia dostępu do instancji `FacesServlet` podczas konfigurowania aplikacji za pomocą programu *deploytool*.
2. W oknie dialogowym *Choose Component Type* kreatora *Web Component* wybrać przycisk *Servlet* i kliknąć *Next*.
3. Z listy rozwijalnej *Servlet Class* wybrać klasę `FacesServlet`.
4. Na liście rozwijalnej *Startup Load Sequence Position* wprowadzić wartość 1 oznaczającą, że klasa `FacesServlet` powinna zostać załadowana podczas uruchamiania aplikacji. Klikamy *Finish*.

5. Z drzewa wybieramy komponent `FacesServlet`.
6. Wybieramy zakładkę *Aliases* i klikamy *Add*.
7. W polu *Aliases* wprowadzamy ścieżkę dostępu do `FacesServlet`. Użytkownicy aplikacji będą musieli wstawić tę ścieżkę do łańcucha URL. W przypadku aplikacji `guessNumber` ścieżką tą jest `/guess/*`.

Zanim aplikacja JavaServer Faces uruchomi pierwszą stronę JSP, kontener musi wywołać instancję `FacesServlet`, aby rozpocząć cykl życia stron omówiony w podrozdziale „Cykl życia strony JavaServer Faces”, rozdział 17.

Aby zagwarantować wywołania instancji `FacesServlet`, tworzymy odwzorowanie w sposób opisany wyżej w punktach od 5. do 7.

Odwzorowanie to używa przedrostka pozwalającego zidentyfikować strony JSP posiadające zawartość JavaServer Faces. Dlatego też adres URL pierwszej strony musi zawierać to odwzorowanie. Istnieją dwa sposoby osiągnięcia tego celu:

- Autor strony może dołączyć do aplikacji stronę HTML, która posiada URL pierwszej strony JSP. Musi on zawierać ścieżkę do `FacesServlet` tak jak poniższy znacznik, który używa odwzorowania zdefiniowanego dla aplikacji `guessNumber`:

```
<a href="guess/greeting.jsp">
```

- Użytkownicy aplikacji mogą włączyć ścieżkę do `FacesServlet` od adresu URL pierwszej strony, wprowadzając go w przeglądarce, co ilustruje poniższy przykład dla aplikacji `guessNumber`:

```
http://localhost:8080/guessNumber/guess/greeting.jsp
```

Drugi sposób umożliwia użytkownikom rozpoczęcie pracy z aplikacją od pierwszej strony JSP zamiast od strony HTML. Wymaga jednak od użytkowników określenia pierwszej strony JSP. Jeśli wybrany zostanie pierwszy sposób, użytkownicy muszą jedynie wprowadzić:

```
http://localhost:8080/guessNumber
```

Zamiast odwzorowania za pomocą przedrostka `/guess/*` można również zdefiniować rozszerzenie za pomocą rozszerzenia, na przykład `*.faces`. Jeśli serwer otrzyma żądanie skierowane do strony JSP o rozszerzeniu `.faces`, to kontener wyśle żądanie do instancji `FacesServlet`, który będzie oczekiwać istnienia odpowiedniej strony JSP o tej samej nazwie. Na przykład, jeśli łańcuch URL żądania będzie miał postać `http://localhost/bookstore6/bookstore.faces`, to instancja `FacesServlet` odwzoruje go na stronę `bookstore.jsp`.

## Określenie ścieżki dostępu do pliku konfiguracyjnego zasobów aplikacji

W podrozdziale „Plik konfiguracyjny zasobów aplikacji” wyjaśniliśmy, że aplikacja może posiadać wiele plików konfiguracyjnych zasobów aplikacji. Jeśli pliki te nie zostaną umieszczone w domyślnie przeszukiwanych katalogach lub nie będą posiadać nazwy `faces-config.xml`, należy określić ścieżki dostępu do tych plików. Ścieżki te określamy za pomocą programu `deploytool` w sposób następujący:

1. Wybieramy z drzewa plik WAR.
2. Wybieramy panel *Context* i klikamy *Add*.
3. W polu *Coded Parameter* wprowadzamy `javax.faces.application.CONFIG_FILES`.
4. W polu *Value* wprowadzamy ścieżkę dostępu do pliku konfiguracyjnego zasobów aplikacji. Na przykład ścieżka dostępu do pliku konfiguracyjnego zasobów aplikacji `guessNumber` ma postać `/WEB-INF/faces-config.xml`.
5. Powtarzamy punkty od 2. do 4. dla każdego pliku konfiguracyjnego zasobów aplikacji.

## Określenie miejsca przechowywania stanu

Implementując metody interfejsu `StateHolder` (patrz punkt „Przechowywanie i odtwarzanie stanu” w rozdziale 20.) określamy w deskrytorze instalacji, czy stan komponentu będzie przechowywany u klienta czy na serwerze. W tym celu konfigurujemy parametr kontekstu za pomocą programu *deploytool*:

1. Wybieramy aplikację z drzewa programu *deploytool*.
2. Wybieramy panel *Context* i klikamy *Add*.
3. W polu *Coded Parameter* wprowadzamy `javax.faces.STATE_SAVING_METHOD`.
4. W polu *Value* wprowadzamy `client` lub `server` w zależności od tego, czy chcemy, by stan był przechowywany u klienta czy na serwerze.

Jeśli stan jest przechowywany u klienta, to stan całego widoku zostaje umieszczony na stronie w ukrytym polu. Implementacja `JavaServer Faces` domyślnie przechowuje stan u klienta. Aplikacja `Księgarnia Duke'a` również przechowuje swój stan u klienta.

## Ograniczanie dostępu do komponentów `JavaServer Faces`

Oprócz identyfikacji instancji `FacesServlet` i stworzenia jej odwzorowania musimy również zapewnić, że wszystkie aplikacje używają instancji `FacesServlet` do przetwarzania komponentów `JavaServer Faces`. W tym celu konfigurujemy ograniczenia.

1. Wybieramy plik WAR z drzewa programu *deploytool*.
2. Wybieramy panel *Security*.
3. Klikamy *Add Constraints* i w polu *Security Constraints* wprowadzamy `Restrict Access to JSP Pages`.
4. Klikamy *Add Collections* i w polu *Web Resource Collections* wprowadzamy `Restrict Access to JSP Pages`.
5. Klikamy *Edit Collections*.
6. W oknie dialogowym *Edit Collections of Web Resource Collections* klikamy *Add URL Pattern* i wprowadzamy ścieżkę dostępu do strony, do której chcemy ograniczyć dostęp, na przykład `/response.jsp`.
7. W sposób opisany w punkcie 6. wprowadzamy ścieżki dostępu do wszystkich stron JSP aplikacji i klikamy *OK*.



## Włączanie kontroli poprawności plików XML

Aplikacja internetowa zawiera jeden lub więcej plików konfiguracyjnych zasobów aplikacji napisanych w języku XML. Możemy zarządzać od implementacji JavaServer Faces, aby sprawdzała poprawność tych plików. W tym celu nadajemy fladze `validateXML` wartość `true`:

1. Wybieramy plik WAR z drzewa programu *deploytool*.
2. Wybieramy panel *Context* i klikamy *Add*.
3. W polu *Coded Parameter* wprowadzamy `com.sun.faces.validateXML`.
4. W polu *Value* wprowadzamy `true`. Wartością domyślną jest `false`.

## Weryfikacja niestandardowych obiektów

Jeśli aplikacja zawiera niestandardowe obiekty takie jak komponenty, konwertery, walidatory i obiekty wyświetlania, to podczas uruchamiania aplikacji możemy zweryfikować możliwość utworzenia tych obiektów. W tym celu nadajemy fladze `verifyObjects` wartość `true`:

1. Wybieramy plik WAR z drzewa programu *deploytool*.
2. Wybieramy panel *Context* i klikamy *Add*.
3. W polu *Coded Parameter* wprowadzamy `com.sun.faces.verifyObjects`.
4. W polu *Value* wprowadzamy `true`. Wartością domyślną jest `false`.

Flaga ta powinna mieć w normalnych warunkach wartość `false`, ponieważ weryfikacja obiektów wiąże się z dodatkowym kosztem.

## Dołączanie wymaganych plików JAR

Aplikacje JavaServer Faces wymagają do prawidłowego działania kilku plików JAR. Należą do nich:

- *jsf-api.jar* (zawiera klasy `javax.faces.*`),
- *jsf-impl.jar* (zawiera klasy implementacji JavaServer Faces),
- *jstl.jar* (wymagany, gdy używamy znaczników JSTL oraz używany przez klasy implementacji JavaServer Faces),
- *standard.jar* (wymagany, gdy używamy znaczników JSTL oraz używany przez klasy referencyjnej implementacji JavaServer Faces),
- *commons-beanutils.jar* (definiowanie i dostęp do właściwości komponentów JavaBeans),
- *commons-digester.jar* (przetwarzanie dokumentów XML),
- *commons-collections.jar* (rozszerzenia szkieletu kolekcji Java 2),
- *commons-logging.jar* (ogólny mechanizm zapisu informacji do dziennika).

Pliki *jsf-api.jar* i *jsf-impl.jar* znajdują się w katalogu `<J2EE_HOME>/lib/`. Plik *jstl.jar* został włączony do pliku *appserv-jstl.jar*. Pozostałe z wymienionych plików JAR włączono do pliku *appserv-rt.jar* również umieszczonego w katalogu `<J2EE_HOME>/lib/`.

Pakując i instalując aplikację JavaServer Faces za pomocą programu *deploytool*, nie musimy pakować żadnego z plików JAR z wyjątkiem pliku *jsf-api.jar*. Plik *jsf-api.jar* musi być spakowany razem z aplikacją, aby umożliwić dostęp do instancji `FacesServlet` i konfigurację jej odwzorowania.

## Dołączanie klas, stron i innych zasobów

Podczas pakowania aplikacji internetowej za pomocą programu *deploytool* zauważymy, że program ten automatycznie pakuje wiele plików aplikacji we właściwych katalogach pliku WAR. Wszystkie strony JSP zostają umieszczone na najwyższym poziomie pliku WAR. Pliki TLD i *web.xml* tworzone przez program *deploytool* zostają umieszczone w katalogu *WEB-INF*. Wszystkie pakiety trafiają do katalogu *WEB-INF/classes*. Jednak program *deploytool* nie kopiuje pliku *faces-config.xml* w katalogu *WEB-INF*. Pakując aplikację, powinniśmy sami skopiować plik *faces-config.xml* do katalogu *WEB-INF*.