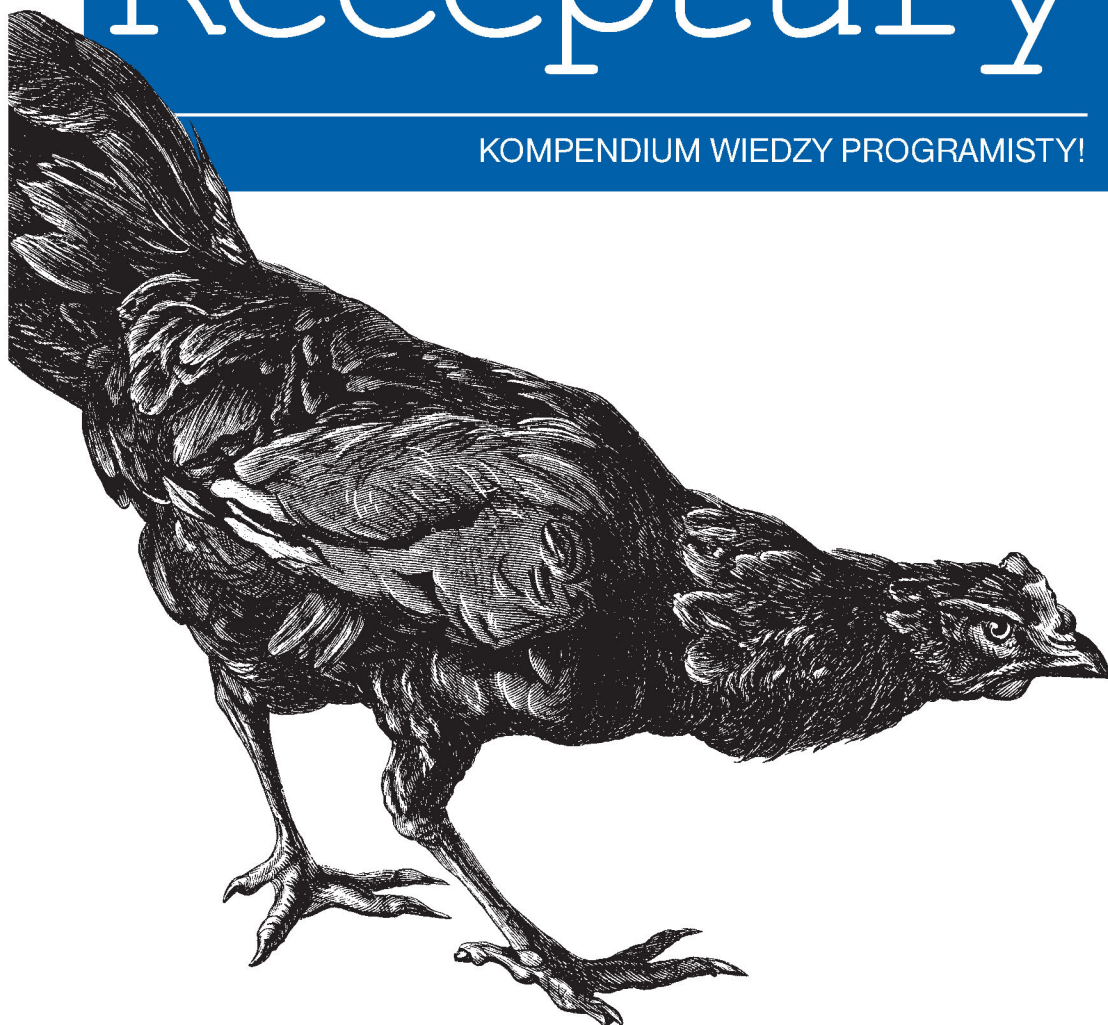


O'REILLY®

Wydanie III

Java Receptury

KOMPENDIUM WIEDZY PROGRAMISTY!



Helion 

Ian F. Darwin

Tytuł oryginału: Java Cookbook, Third Edition

Tłumaczenie: Piotr Rajca

ISBN: 978-83-246-9570-6

© 2015 Helion S.A.

Authorized Polish translation of the English edition of Java Cookbook,
ISBN 9781449337049 © 2014 RejmiNet Group, Inc.

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

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

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

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

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

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

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/jarec3>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:

<ftp://ftp.helion.pl/przyklady/jarec3.zip>

Printed in Poland.

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

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

Spis treści

Wstęp	13
1. Rozpoczynanie pracy: kompilacja, uruchamianie i testowanie	31
1.0. Wprowadzenie	31
1.1. Kompilacja i uruchamianie programów napisanych w Javie — JDK	32
1.2. Edycja i kompilacja programów przy użyciu edytorów wyposażonych w kolorowanie syntaktyczne	33
1.3. Kompilacja, uruchamianie i testowanie programów przy użyciu IDE	35
1.4. Efektywne wykorzystanie zmiennej środowiskowej CLASSPATH	42
1.5. Pobieranie przykładów dołączonych do tej książki i korzystanie z nich	44
1.6. Automatyzacja kompilacji przy użyciu programu Ant	50
1.7. Automatyzacja zależności, kompilacji, testowania i wdrażania przy użyciu programu Apache Maven	53
1.8. Automatyzacja zależności, kompilacji, testowania i wdrażania przy użyciu programu Gradle	56
1.9. Komunikaty o odrzuconych metodach	59
1.10. Testowanie warunkowe bez użycia dyrektywy #ifdef	61
1.11. Zapewnianie poprawności programu za pomocą asercji	63
1.12. Wykorzystanie programu uruchomieniowego	64
1.13. Testowanie jednostkowe — jak uniknąć konieczności stosowania programów uruchomieniowych?	66
1.14. Zarządzanie kodem z wykorzystaniem ciągłej integracji	69
1.15. Uzyskiwanie czytelnych komunikatów o wyjątkach	74
1.16. Poszukiwanie przykładowych kodów źródłowych: programy, szkielety i biblioteki	74
2. Interakcja ze środowiskiem	79
2.0. Wprowadzenie	79
2.1. Pobieranie wartości zmiennych środowiskowych	79
2.2. Pobieranie informacji z właściwości systemowych	81
2.3. Określanie używanej wersji JDK	82
2.4. Tworzenie kodu zależnego od używanego systemu operacyjnego	84
2.5. Stosowanie rozszerzających interfejsów programistycznych lub innych API	87
2.6. Analiza argumentów podanych w wierszu wywołania programu	87

3. Łańcuchy znaków i przetwarzanie tekstów	95
3.0. Wprowadzenie	95
3.1. Odczytywanie fragmentów łańcucha	97
3.2. Dzielenie łańcuchów na słowa	98
3.3. Łączenie łańcuchów znaków przy użyciu klasy <code>StringBuilder</code>	102
3.4. Przetwarzanie łańcucha znaków po jednej literze	104
3.5. Wyrównywanie łańcuchów znaków	105
3.6. Konwersja pomiędzy znakami Unicode a łańcuchami znaków	108
3.7. Odwracanie kolejności słów lub znaków w łańcuchu	110
3.8. Rozwijanie i kompresja znaków tabulacji	111
3.9. Kontrola wielkości liter	116
3.10. Wcinanie zawartości dokumentów tekstowych	117
3.11. Wprowadzanie znaków niedrukowalnych	118
3.12. Usuwanie odstępów z końca łańcucha	119
3.13. Przetwarzanie danych rozdzielonych przecinkami	120
3.14. Program — proste narzędzie do formatowania tekstów	125
3.15. Program — fonetyczne porównywanie nazwisk	127
 4. Dopasowywanie wzorców przy użyciu wyrażeń regularnych	 131
4.0. Wprowadzenie	131
4.1. Składnia wyrażeń regularnych	133
4.2. Wykorzystanie wyrażeń regularnych w języku Java — sprawdzanie występowania wzorca	140
4.3. Odnajdywanie tekstu pasującego do wzorca	143
4.4. Zastępowanie określonego tekstu	146
4.5. Wyświetlanie wszystkich wystąpień wzorca	147
4.6. Wyświetlanie wierszy zawierających fragment pasujący do wzorca	149
4.7. Kontrola wielkości znaków w metodach <code>match()</code> i <code>subst()</code>	150
4.8. Dopasowywanie znaków z akcentami lub znaków złożonych	152
4.9. Odnajdywanie znaków nowego wiersza	153
4.10. Program — analiza dziennika serwera Apache	155
4.11. Program — analizowanie danych	156
4.12. Program — pełna wersja programu <code>grep</code>	159
 5. Liczby	 165
5.0. Wprowadzenie	165
5.1. Sprawdzanie, czy łańcuch znaków stanowi poprawną liczbę	168
5.2. Zapisywanie dużych wartości w zmiennych „mniejszych” typów	170
5.3. Konwertowanie liczb na obiekty i na odwrót	171
5.4. Pobieranie ułamka z liczby całkowitej bez konwertowania go na postać zmiennoprzecinkową	172
5.5. Wymuszanie zachowania dokładności liczb zmiennoprzecinkowych	173
5.6. Porównywanie liczb zmiennoprzecinkowych	175

5.7. Zaokrąglanie wartości zmiennoprzecinkowych	177
5.8. Formatowanie liczb	178
5.9. Konwersje pomiędzy różnymi systemami liczbowymi — dwójkowym, ósemkowym, dziesiętnym i szesnastkowym	181
5.10. Operacje na grupie liczb całkowitych	182
5.11. Posługiwanie się cyframi rzymskimi	183
5.12. Formatowanie z zachowaniem odpowiedniej postaci liczby mnogiej	187
5.13. Generowanie liczb losowych	189
5.14. Obliczanie funkcji trygonometrycznych	192
5.15. Obliczanie logarytmów	192
5.16. Mnożenie macierzy	193
5.17. Operacje na liczbach zespolonych	195
5.18. Obsługa liczb o bardzo dużych wartościach	197
5.19. Program TempConverter	200
5.20. Program — generowanie liczbowych palindromów	201
6. Daty i godziny — nowy interfejs programowania aplikacji	205
6.0. Wprowadzenie	205
6.1. Określanie bieżącej daty	208
6.2. Wyświetlanie daty i czasu w zadanym formacie	209
6.3. Konwersja liczb określających datę i czas oraz ilości sekund	211
6.4. Analiza łańcuchów znaków i ich zamiana na daty	212
6.5. Obliczanie różnic pomiędzy dwiema datami	213
6.6. Dodawanie i odejmowanie dat	214
6.7. Stosowanie starych klas Date i Calendar	215
7. Strukturalizacja danych w języku Java	217
7.0. Wprowadzenie	217
7.1. Strukturalizacja danych przy użyciu tablic	218
7.2. Modyfikacja wielkości tablic	220
7.3. Szkielet kolekcji	221
7.4. Klasa podobna do tablicy, lecz bardziej dynamiczna	223
7.5. Stosowanie kolekcji ogólnych	225
7.6. Unikanie rzutowania dzięki zastosowaniu typów ogólnych	227
7.7. Jak przeglądać zawartość kolekcji? Wyliczenie dostępnych sposobów	230
7.8. Unikanie powtórzeń dzięki zastosowaniu zbioru	232
7.9. Iteratory lub wyliczenia — dostęp do danych w sposób niezależny od ich typów	233
7.10. Strukturalizacja danych z wykorzystaniem list połączonych	234
7.11. Odzworowywanie z wykorzystaniem klas Hashtable oraz HashMap	238
7.12. Zapisywanie łańcuchów znaków w obiektach Properties i Preferences	240
7.13. Sortowanie kolekcji	244
7.14. Unikanie konieczności sortowania danych	248
7.15. Odnajdywanie obiektu w kolekcji	250

7.16. Zamiana kolekcji na tablicę	252
7.17. Tworzenie własnego iteratora	253
7.18. Stos	256
7.19. Struktury wielowymiarowe	259
7.20. Program — porównanie szybkości działania	261
8. Techniki obiektowe	263
8.0. Wprowadzenie	263
8.1. Wyświetlanie obiektów — formatowanie obiektów przy użyciu metody toString()	266
8.2. Przesłanie metod equals() oraz hashCode()	267
8.3. Porządki w aplikacji przy użyciu metody addShutdownHook()	273
8.4. Wykorzystanie klas wewnętrznych	274
8.5. Tworzenie metod zwrotnych z wykorzystaniem interfejsów	276
8.6. Polimorfizm i metody abstrakcyjne	279
8.7. Przekazywanie wartości	281
8.8. Wartości wyliczeniowe bezpieczne dla typów	284
8.9. Wymuszanie użycia wzorca Singleton	288
8.10. Zgłaszanie własnych wyjątków	290
8.11. Wstrzykiwanie zależności	291
8.12. Program Plotter	294
9. Techniki programowania funkcyjnego: interfejsy funkcyjne, strumienie i kolekcje równoległe	299
9.0. Wprowadzenie	299
9.1. Stosowanie wyrażeń lambda lub domknięć zamiast klas wewnętrznych	301
9.2. Stosowanie predefiniowanych interfejsów lambda zamiast własnych	304
9.3. Upraszczenie przetwarzania z wykorzystaniem interfejsu Stream	306
9.4. Poprawianie przepustowości dzięki wykorzystaniu strumieni i kolekcji równoległych	308
9.5. Tworzenie własnych interfejsów funkcyjnych	309
9.6. Używanie istniejącego kodu w sposób funkcyjny dzięki wykorzystaniu odwołań do metod	311
9.7. Wstawianie istniejącego kodu metod	315
10. Wejście i wyjście	317
10.0. Wprowadzenie	317
10.1. Odczytywanie informacji ze standardowego strumienia wejściowego	320
10.2. Odczyt z konsoli lub okna terminala; odczyt hasła bez jego wyświetlania	323
10.3. Zapis danych w standardowym strumieniu wyjściowym lub w strumieniu błędów	325
10.4. Wyświetlanie tekstów przy użyciu klasy Formatter i metody printf	327
10.5. Analiza zawartości pliku przy użyciu klasy StringTokenizer	331
10.6. Analiza danych wejściowych przy użyciu klasy Scanner	335
10.7. Analiza danych wejściowych o strukturze gramatycznej	338
10.8. Otwieranie pliku o podanej nazwie	340
10.9. Kopiowanie plików	341

10.10. Odczytywanie zawartości pliku i zapisywanie jej w obiekcie String	347
10.11. Zmiana skojarzeń standardowych strumieni	347
10.12. Powielanie strumienia podczas realizacji operacji zapisu	348
10.13. Odczyt i zapis danych zakodowanych w innym zbiorze znaków	351
10.14. Te kłopotliwe znaki końca wiersza	352
10.15. Kod operujący na plikach w sposób zależny od systemu operacyjnego	353
10.16. Odczytywanie „podzielonych” wierszy tekstu	354
10.17. Odczytywanie i zapisywanie danych binarnych	358
10.18. Przejsie do określonego miejsca w pliku	359
10.19. Zapisywanie danych w strumieniu z wykorzystaniem języka C	360
10.20. Zapisywanie i odczytywanie obiektów	363
10.21. Unikanie wyjątków ClassCastException spowodowanych nieprawidłowymi wartościami SerialVersionUID	366
10.22. Odczytywanie i zapisywanie danych w archiwach JAR oraz ZIP	368
10.23. Odnajdywanie plików w sposób niezależny od systemu operacyjnego przy użyciu metod getResource() i getResourceAsStream()	371
10.24. Odczytywanie i zapisywanie skompresowanych plików	373
10.25. Poznawanie API do obsługi portów szeregowych i równoległych	374
10.26. Zapisywanie danych użytkownika na dysku	379
10.27. Program — zamiana tekstu do postaci PostScript	382
11. Operacje na katalogach i systemie plików	387
11.0. Wprowadzenie	387
11.1. Pobieranie informacji o pliku	388
11.2. Tworzenie pliku	390
11.3. Zmiana nazwy pliku	391
11.4. Usuwanie plików	392
11.5. Tworzenie plików tymczasowych	394
11.6. Zmiana atrybutów pliku	395
11.7. Tworzenie listy zawartości katalogu	397
11.8. Pobieranie katalogów głównych	399
11.9. Tworzenie nowych katalogów	400
11.10. Stosowanie klasy Path zamiast File	401
11.11. Stosowanie usługi WatchService do uzyskiwania informacji o zmianach pliku	402
11.12. Program Find	404
12. Multimedia: grafika, dźwięk i wideo	407
12.0. Wprowadzenie	407
12.1. Rysowanie przy użyciu obiektu Graphics	408
12.2. Testowanie komponentów graficznych	409
12.3. Wyświetlanie tekstu	410
12.4. Wyświetlanie wyśrodkowanego tekstu w komponencie	411
12.5. Rysowanie cienia	413

12.6. Wyświetlanie tekstu przy użyciu biblioteki grafiki dwuwymiarowej	415
12.7. Wyświetlanie tekstu przy użyciu czcionki aplikacji	417
12.8. Wyświetlanie obrazu	419
12.9. Odczyt i zapis obrazów przy użyciu pakietu javax.imageio	423
12.10. Odtwarzanie pliku dźwiękowego	424
12.11. Prezentacja ruchomego obrazu	426
12.12. Drukowanie w Javie	430
12.13. Program PlotterAWT	434
12.14. Program Grapher	435
13. Klienci sieciowe	439
13.0. Wprowadzenie	439
13.1. Nawiązywanie połączenia z serwerem	441
13.2. Odnajdywanie i zwracanie informacji o adresach sieciowych	443
13.3. Obsługa błędów sieciowych	445
13.4. Odczyt i zapis danych tekstowych	446
13.5. Odczyt i zapis danych binarnych	448
13.6. Odczyt i zapis danych serializowanych	450
13.7. Datagramy UDP	452
13.8. Program — klient TFIP wykorzystujący protokół UDP	454
13.9. URI, URL czy może URN?	458
13.10. Klient usługi internetowej REST	459
13.11. Klient usługi internetowej SOAP	461
13.12. Program — klient usługi Telnet	466
13.13. Program — klient pogawędek internetowych	468
13.14. Program — sprawdzanie odnośników HTTP	472
14. Graficzny interfejs użytkownika	475
14.0. Wprowadzenie	475
14.1. Wyświetlanie komponentów graficznego interfejsu użytkownika	477
14.2. Uruchamianie graficznego interfejsu użytkownika w wątku przekazywania zdarzeń	478
14.3. Projektowanie układu okna	480
14.4. Karty — nowe spojrzenie na świat	483
14.5. Obsługa czynności — tworzenie działających przycisków	484
14.6. Obsługa czynności z wykorzystaniem anonimowych klas wewnętrznych	486
14.7. Obsługa czynności z wykorzystaniem wyrażeń lambda	488
14.8. Kończenie programu przy użyciu przycisku Zamknij	489
14.9. Okna dialogowe — tego nie można zrobić później	494
14.10. Przechwytywanie i formatowanie wyjątków graficznego interfejsu użytkownika ...	496
14.11. Wyświetlanie wyników wykonania programu w oknie	499
14.12. Wybieranie wartości przy użyciu komponentu JSpinner	505
14.13. Wybieranie plików przy użyciu klasy JFileChooser	506

14.14. Wybieranie koloru	509
14.15. Formatowanie komponentów przy użyciu kodu HTML	511
14.16. Wyświetlanie okna głównego pośrodku ekranu	512
14.17. Zmiana sposobów prezentacji programów pisanych z wykorzystaniem pakietu Swing	515
14.18. Korzystanie z rozszerzonych możliwości pakietu Swing w systemie Mac OS X	519
14.19. Tworzenie aplikacji z graficznym interfejsem użytkownika przy użyciu pakietu JavaFX	522
14.20. Program — własne narzędzie do wybierania czcionek	524
14.21. Program — własny menedżer układu	528
15. Tworzenie programów wielojęzycznych oraz lokalizacja	535
15.0. Wprowadzenie	535
15.1. Tworzenie przycisku w różnych wersjach językowych	535
15.2. Tworzenie listy dostępnych ustawień lokalnych	538
15.3. Tworzenie menu z wykorzystaniem zasobów wielojęzycznych	539
15.4. Tworzenie metod pomocniczych przydatnych podczas pisanie programów wielojęzycznych	539
15.5. Tworzenie okien dialogowych z wykorzystaniem zasobów wielojęzycznych	541
15.6. Tworzenie wiązki zasobów	543
15.7. Usuwanie łańcuchów znaków z kodu	544
15.8. Wykorzystanie konkretnych ustawień lokalnych	545
15.9. Określanie domyślnych ustawień lokalnych	546
15.10. Formatowanie komunikatów przy użyciu klasy MessageFormat	547
15.11. Program MenuIntl	549
15.12. Program BusCard	551
16. Programy Javy działające na serwerze — gniazda	555
16.0. Wprowadzenie	555
16.1. Tworzenie serwera	556
16.2. Zwracanie odpowiedzi (łańcucha znaków bądź danych binarnych)	558
16.3. Zwracanie informacji o obiektach	562
16.4. Obsługa wielu klientów	563
16.5. Serwer obsługujący protokół HTTP	567
16.6. Zabezpieczanie serwera WWW przy użyciu SSL i JSSE	570
16.7. Rejestracja operacji sieciowych	572
16.8. Rejestracja przez sieć przy użyciu SLF4J	574
16.9. Rejestracja przez sieć przy użyciu log4j	576
16.10. Rejestracja przez sieć przy użyciu pakietu java.util.logging	579
16.11. Znajdowanie interfejsów sieciowych	581
16.12. Program — serwer pogawędek w Javie	582

17. Java i poczta elektroniczna	587
17.0. Wprowadzenie	587
17.1. Wysyłanie poczty elektronicznej — wersja działająca w przeglądarkach	588
17.2. Wysyłanie poczty elektronicznej — właściwe rozwiązanie	592
17.3. Dodawanie możliwości wysyłania poczty do programu działającego na serwerze	594
17.4. Wysyłanie wiadomości MIME	599
17.5. Tworzenie ustawień poczty elektronicznej	602
17.6. Odczytywanie poczty elektronicznej	603
17.7. Program MailReaderBean	608
17.8. Program MailClient	611
18. Dostęp do baz danych	621
18.0. Wprowadzenie	621
18.1. Łatwy dostęp do bazy danych przy użyciu JPA oraz Hibernate	623
18.2. Konfiguracja i nawiązywanie połączeń JDBC	628
18.3. Nawiązywanie połączenia z bazą danych JDBC	631
18.4. Przesyłanie zapytań JDBC i pobieranie wyników	634
18.5. Wykorzystanie przygotowanych poleceń JDBC	637
18.6. Wykorzystanie procedur osadzonych w JDBC	641
18.7. Modyfikacja danych przy użyciu obiektu ResultSet	641
18.8. Zapisywanie wyników w obiektach RowSet	642
18.9. Modyfikacja danych przy użyciu poleceń SQL	644
18.10. Odnajdywanie metadanych JDBC	646
18.11. Program SQLRunner	650
19. Przetwarzanie danych w formacie JSON	661
19.0. Wprowadzenie	661
19.1. Bezpośrednie generowanie danych w formacie JSON	663
19.2. Analiza i zapisywanie danych JSON przy użyciu pakietu Jackson	664
19.3. Analiza i zapis danych w formacie JSON przy użyciu pakietu org.json	665
20. XML	669
20.0. Wprowadzenie	669
20.1. Konwersja obiektów na dane XML przy użyciu JAXB	672
20.2. Konwersja obiektów na dane XML przy użyciu serializatorów	675
20.3. Przekształcanie danych XML przy użyciu XSLT	676
20.4. Analiza składniowa XML przy użyciu API SAX	679
20.5. Analiza dokumentów XML przy użyciu modelu obiektów dokumentu (DOM)	681
20.6. Odnajdywanie elementów XML przy użyciu XPath	684
20.7. Weryfikacja poprawności struktury z wykorzystaniem DTD	686
20.8. Generowanie własnego kodu XML z wykorzystaniem DOM i obiektów przekształceń XML	689
20.9. Program xml2mif	691

21. Pakiety i ich tworzenie	693
21.0. Wprowadzenie	693
21.1. Tworzenie pakietu	694
21.2. Tworzenie dokumentacji klas przy użyciu programu Javadoc	696
21.3. Więcej niż Javadoc — adnotacje i metadane	700
21.4. Stosowanie programu archiwizującego jar	701
21.5. Uruchamianie programu zapisanego w pliku JAR	703
21.6. Tworzenie klasy w taki sposób, by była komponentem JavaBeans	704
21.7. Umieszczanie komponentów w plikach JAR	708
21.8. Umieszczanie serwetów w plikach JAR	709
21.9. „Zapisz raz, instaluj wszędzie”	710
21.10. „Napisz raz, instaluj na Mac OS X”	711
21.11. Java Web Start	713
21.12. Podpisywanie plików JAR	719
22. Stosowanie wątków w Javie	721
22.0. Wprowadzenie	721
22.1. Uruchamianie kodu w innym wątku	723
22.2. Animacja — wyświetlanie poruszających się obrazów	728
22.3. Zatrzymywanie działania wątku	732
22.4. Spotkania i ograniczenia czasowe	734
22.5. Synchronizacja wątków przy użyciu słowa kluczowego synchronized	735
22.6. Upraszczenie synchronizacji przy użyciu blokad	741
22.7. Komunikacja między wątkami — metody wait() oraz notifyAll()	745
22.8. Upraszczenie programu producent-konsument przy użyciu interfejsu Queue	750
22.9. Optymalizacja działania równoległego przy użyciu Fork/Join	753
22.10. Zapis danych w tle w programach edycyjnych	756
22.11. Wielowątkowy serwer sieciowy	758
22.12. Upraszczenie serwerów z wykorzystaniem klas pakietu java.util.concurrent	765
23. Introspekcja lub „klasa o nazwie Class”	769
23.0. Wprowadzenie	769
23.1. Pobieranie deskryptora klasy	770
23.2. Określanie oraz stosowanie metod i pól	771
23.3. Uzyskiwanie dostępu do prywatnych pól i metod za pomocą introspekcji	774
23.4. Dynamiczne ładowanie i instalowanie klas	775
23.5. Tworzenie nowej klasy od podstaw przy użyciu obiektu ClassLoader	778
23.6. Określanie efektywności działania	779
23.7. Wyświetlanie informacji o klasie	784
23.8. Wyświetlanie klas należących do pakietu	785
23.9. Stosowanie i definiowanie adnotacji	787
23.10. Zastosowanie adnotacji do odnajdywania klas pełniących rolę wtoczek	792
23.11. Program CrossRef	794
23.12. Program AppletViewer	796

24. Wykorzystywanie Javy wraz z innymi językami programowania	803
24.0. Wprowadzenie	803
24.1. Uruchamianie zewnętrznego programu	804
24.2. Wykonywanie programu i przechwytywanie jego wyników	808
24.3. Wywoływanie kodu napisanego w innych językach przy użyciu javax.script	811
24.4. Tworzenie własnego mechanizmu skryptowego	813
24.5. Łączenie języków Java i Perl	817
24.6. Dołączanie kodu rodzimego	820
24.7. Wywoływanie kodu Javy z kodu rodzimego	825
 Posłowie	829
 A Java kiedyś i obecnie	831
 Skorowidz	849

Techniki programowania funkcyjnego: interfejsy funkcyjne, strumienie i kolekcje równoległe

9.0. Wprowadzenie ⑧

Java jest językiem programowania obiektowego. Doskonale o tym wiemy. Obecnie coraz większym zainteresowaniem cieszy się programowanie funkcyjne (ang. *functional programming*, FP). Być może nie ma aż tak wielu definicji programowania funkcyjnego jak języków, które umożliwiają stosowanie tego stylu programowania, choć ich liczba może być podobna. A oto co na temat programowania funkcyjnego napisano w Wikipedii:

„(...) paradygmat programowania, styl tworzenia struktury i elementów programów komputerowych, traktujący obliczenia jako przetwarzanie funkcji matematycznych i unikający przechowywania stanu oraz danych podlegających zmianom. Programowanie funkcyjne kładzie nacisk na funkcje, których wyniki zależą wyłącznie od danych wejściowych, a nie od stanu programu, innymi słowy, na funkcje o charakterze matematycznym. Jest to paradygmat programowania deklaratywnego, co oznacza, że programowanie bazuje na wykorzystaniu wyrażeń. W kodzie funkcyjnym wyniki zwracane przez funkcję są zależne wyłącznie od przekazanych do niej argumentów, a zatem dwukrotne wywołanie funkcji f z tym samym argumentem x w obu przypadkach spowoduje zwrócenie tego samego wyniku $f(x)$. Wyeliminowanie efektów ubocznych, takich jak zmiany stanu, które nie zależą od danych przekazanych do funkcji, znacznie ułatwia zrozumienie i określenie sposobu działania programu i stanowi jeden z kluczowych powodów rozwoju programowania funkcyjnego (...).”

— http://en.wikipedia.org/wiki/Functional_programming

z października 2014

W jaki sposób możemy wykorzystać zasady programowania funkcyjnego? Jednym z nich mogłoby być użycie odpowiedniego funkcyjnego języka programowania, a do wiodących języków tego typu należą Haskell OCaml, Erlang oraz rodzina języków LISP. Jednak wyagałoby to odejścia z ekosystemu języka Java. Można by się ewentualnie zastanowić nad wykorzystaniem języków Scala (<http://www.scala-lang.org/>) lub Clojure (<http://clojure.org/>), które działają w oparciu o wirtualną maszynę Javy i zapewniają wsparcie dla programowania funkcyjnego w kontekście języka obiektowego.

Jednak niniejsza książka jest poświęcona Javie, zatem można sobie wyobrazić, że będziemy dążyć do skorzystania z zalet, jakie daje programowanie funkcyjne z wykorzystaniem wyłącznie tego języka. Do cech programowania funkcyjnego należą:

- Funkcje czyste (ang. *pure functions*), czyli funkcje, które nie mają żadnych efektów ubocznych i których wyniki zależą wyłącznie od przekazanych argumentów, a nie od stanu programu, który może ulegać zmianom.
- Funkcje pierwszej klasy, czyli możliwość traktowania funkcji jako danych.
- Dane niezmiennie.
- Częste stosowanie rekurencji i przetwarzania leniwego.

Funkcje czyste są całkowicie niezależne; ich działanie zależy wyłącznie od przekazanych danych wejściowych oraz ich wewnętrznej logiki, a nie od zmiennego „stanu” innych części programu — w rzeczywistości w programowaniu funkcyjnym nie ma czegoś takiego jak „zmiennie globalne”, a jedynie „stałe globalne”. Choć dla osób przyzwyczajonych do stosowania języków imperatywnych, takich jak Java, może to być sporym zaskoczeniem, to jednak takie rozwiązania mogą znacznie ułatwić testowanie programów oraz zapewnienie prawidłowości ich działania! Oznacza to bowiem, że niezależnie do tego, co się dzieje w pozostałych częściach programu (nawet w niezależnie działających wątkach), wywołanie metody, takie jak `computeValue(27)`, zawsze i bezwarunkowo zwróci tę samą wartość (wyjątkami są tu funkcje zwracające elementy stanu globalnego, np. aktualną datę i godzinę, wartość losową itd.).

W tym rozdziale terminów *funkcja* oraz *metoda* będę używał wymiennie, choć zapewne nie jest to do końca poprawne. Osoby związane z programowaniem funkcyjnym używają terminu „funkcja”, mając na myśli matematyczną definicję funkcji, natomiast w języku Java „metody” są jedynie „kodem, który można wywołać” (z obiektowego punktu widzenia „wywołanie metody” w Javie określa się także jako *przesłanie sygnału* do obiektu).

„Traktowanie funkcji jako danych” oznacza, że można utworzyć obiekt będący funkcją, przekazać go do innej funkcji, napisać funkcję, która będzie zwracać inną funkcję, i tak dalej — a to wszystko bez konieczności stosowania jakiegokolwiek szczególnej składni, gdyż funkcje *są* danymi.

Jednym z rozwiązań wprowadzonych w nowej wersji języka — Java 8 — mającym na celu udostępnienie możliwości programowania funkcyjnego są „interfejsy funkcyjne”. *Interfejsem funkcyjnym* w Javie nazywamy interfejs, który definiuje tylko jedną metodę. Przykładami takich interfejsów mogą być bardzo popularny interfejs `Runnable` definiujący metodę `run()` oraz powszechnie używany w bibliotece Swing interfejs `ActionListener`, który definiuje metodę `actionPerformed(ActionEvent)`. Okazuje się, że także te nowe interfejsy języka Java 8 mogą posiadać metody zadeklarowane za pomocą słowa kluczowego `default`, którego użycie w tym kontekście jest nowością. Takie domyślne metody stają się dostępne i mogą być używane w każdej klasie implementującej dany interfejs. Jeśli się nad tym zastanowimy, to stanie się jasne, że działanie takich metod nie może zależeć od stanu konkretnej klasy, gdyż nie miałyby one możliwości odwołania się do tego stanu w czasie kompilacji programu.

A zatem nieco precyzyjniej rzecz ujmując, interfejs funkcyjny jest interfejsem definiującym jedną, niedomyślną metodę. W języku Java można korzystać z funkcyjnego stylu programowania, jeśli zastosujemy interfejsy funkcyjne oraz jeśli kod umieszczany w metodach będzie korzystał wyłącznie ze sfinalizowanych zmiennych oraz pól obiektów. Jednym ze sposobów spełnienia tych wymagań jest korzystanie z metod domyślnych. Kilka pierwszych receptur tego rozdziału jest poświęconych właśnie interfejsom funkcyjnym.

Kolejnym nowym rozwiązaniem umożliwiającym stosowanie funkcyjnego stylu programowania są „wyrażenia lambda”. Lambda to wyrażenie, którego typem jest interfejs funkcyjny i które może być używane jako dana (czyli można je przypisywać zmiennym lub zwracać jako wynik wywołania metody itd.). Poniżej podałem dwa krótkie przykłady wyrażen lambda.

```
ActionListener x = (e -> System.out.println("Uaktywniono " + e.getSource()));
```

```
public class RunnableLambda {  
    public static void main(String[] args) {  
        new Thread(() -> System.out.println("Witam w wątku")).start();  
    }  
}
```

Kolejną nowością wprowadzoną w Java 8 są klasy Stream. Przypominają one nieco potok, w którym można coś umieścić, następnie wykonać na nim jakieś operacje, po czym przekazać jego zawartość dalej — czyli coś, co można by uznać za połączenie uniksowych potoków oraz opracowanego przez Google modelu programowania rozproszonego MapReduce (którego przykładem może być projekt Hadoop, <http://hadoop.apache.org/>), lecz działające w obrębie jednej wirtualnej maszyny Javy, czyli jednego programu. Obiekty Stream mogą działać szeregowo lub równolegle; przy czym te drugie zostały zaprojektowane w celu wykorzystania możliwości przetwarzania równoległego, jakie zapewniają platformy sprzętowe (zwłaszcza serwery, które bardzo często są wyposażane w procesory dysponujące dwunastoma lub szesnastoma rdzeniami). Także klasom Stream poświęciłem kilka receptur zamieszczonych w tym rozdziale.

Z klasami Stream powiązany jest interfejs Splitator, stanowiący pochodną (pod względem logicznym, a nie dziedziczenia) iteratorów, lecz zaprojektowany w celu wykorzystania w przetwarzaniu równoległym. *Większość osób nie będzie musiała tworzyć własnych implementacji tego interfejsu, a nawet nie będzie musiała zbyt często jawnie wywoływać jego metod, dlatego też nie poświęcę mu w tej książce wiele uwagi.*

Patrz także

Ogólne informacje na temat programowania funkcyjnego można znaleźć w książce *Functional Thinking* (<http://shop.oreilly.com/product/0636920029687.do>).

Dostępna jest także książka Richarda Warburtona *Java 8 Lambdas* (<http://shop.oreilly.com/product/0636920030713.do>), która została w całości poświęcona wyrażeniom lambda oraz związanym z nimi narzędziom.

9.1. Stosowanie wyrażen lambda lub domknięć zamiast klas wewnętrznych

Problem ⑧

Chcemy uniknąć pisania rozbudowanego kodu, którego wymaga stosowanie klas wewnętrznych.

Rozwiązanie

Należy skorzystać z wyrażen lambda.

Analiza

Symbol lambda (λ) to jedenasta litera alfabetu greckiego, a zatem jest on tak stary jak cała cywilizacja zachodnioeuropejska. Rachunek lambda (http://pl.wikipedia.org/wiki/Rachunek_lambda) jest równie stary jak sama informatyka. W tym kontekście wyrażenia lambda są *niewielkimi fragmentami obliczeń, do których można się odwoływać*. Są one funkcjami, które można traktować jako dane. W tym sensie są one bardzo podobne do anonimowych funkcji wewnętrznych, choć chyba lepszym rozwiązaniem byłoby wyobrażenie ich sobie jako *anonimowych metod*. Są one zasadniczo stosowane jako zamienniki klas wewnętrznych w kodzie wykorzystującym *interfejsy funkcyjne* — czyli interfejsy definiujące tylko jedną metodę („funkcję”). Doskonałym przykładem takiego interfejsu funkcyjnego jest `ActionListener`, powszechnie stosowany w kodzie obsługi interfejsu użytkownika. Interfejs ten definiuje tylko jedną metodę:

```
public void actionPerformed(ActionEvent);
```

Przykłady wykorzystania wyrażen lambda w obsłudze interfejsu użytkownika można znaleźć w rozdziale 14. Poniżej, aby rozbudzić zainteresowanie Czytelnika, zamieściłem jeden z nich:

```
quitButton.addActionListener(e -> System.exit(0));
```

Jednak obecnie już nie wszyscy tworzą aplikacje z graficznym interfejsem użytkownika, dlatego zacznę od przykładu, który nie jest z nimi w żaden sposób związany. Założmy, że dysponujemy zbiorem obiektów deskryptorów aparatów fotograficznych, które *zostały już wczytane z bazy danych i zapisane w pamięci*, a teraz chcemy napisać interfejs programistyczny ogólnego przeznaczenia pozwalający na ich przeszukiwanie, którego moglibyśmy używać w pozostałych miejscach naszej aplikacji.

Pierwszym pomysłem mogłoby być stworzenie następującego interfejsu:

```
public interface CameraInfo {  
    public List<Camera> findByMake();  
    public List<Camera> findByModel();  
    ...  
}
```

Jednak być może Czytelnik już zauważył problem wiążący się z takim rozwiązaniem. Otóż wraz ze zwiększaniem się stopnia złożoności naszej aplikacji konieczne byłoby także zaimplementowanie metod `findByPrice()`, `findByMakeAndModel()`, `findByYearIntroduced()` i tak dalej.

Mozna by sobie wyobrazić metodę „znajdź na podstawie przykładu”, do której byłby przekazywany obiekt `Camera`, a metoda odnajdywałaby inne obiekty, używając do porównania wszystkich pól argumentu o wartościach różnych od `null`. Ale w jaki sposób należałoby zaimplementować wyszukiwanie wszystkich aparatów z wymiennymi obiektywami o cenie poniżej 1500 złotych?¹

¹ Gdybyśmy kiedyś musieli wykonywać tego typu operacje na informacjach przechowywanych w bazie danych, wykorzystując przy tym Java Persistence API (patrz receptura 18.1), to warto się zainteresować projektem Apache DeltaSpike (<http://deltaspike.apache.org/>), który pozwala na definiowanie interfejsów zawierających metody o nazwach takich jak `findCameraByInterchangeableTrueAndPriceLessThan(double price)` i jest w stanie sam je zaimplementować. W serwisie GitHub dostępne są wzorce projektów korzystających z CDI oraz DeltaSpike: Java SE (<https://github.com/os890/javase-cdi-ds-project-template>) oraz Java Web (<https://github.com/os890/java-web-cdi-ds-project-template>).

A zatem można sądzić, że lepszym sposobem na wykonanie takiego porównania byłoby zastosowanie „funkcji zwrotnej”. Pozwoliłoby ono na stworzenie anonimowej klasy wewnętrznej, która odpowiadałaby za wykonanie odpowiednich poszukiwań. Chcielibyśmy, żeby funkcja zwrotna mogła wyglądać w następujący sposób:

```
public boolean choose(Camera c) {  
    return c.isIlc() && c.getPrice() < 500;  
}
```

W tym celu musielibyśmy stworzyć interfejs o poniższej postaci²:

//functional/CameraAcceptor.java

```
/** Interfejs wybiera (akceptuje) niektóre elementy kolekcji. */  
public interface CameraAcceptor {  
    boolean choose(Camera c);  
}
```

Teraz aplikacja wyszukująca aparaty mogłaby udostępnić metodę:

```
public List<Camera> search(CameraAcceptor acc);
```

którą można by wywołać, używając następującego fragmentu kodu (zakładając, że potrafimy posługiwać się anonimowymi klasami wewnętrznymi):

```
results = searchApp.search(new CameraAcceptor() {  
    public boolean choose(Camera c) {  
        return c.isIlc() && c.getPrice() < 500;  
    }  
})
```

Gdyby ktoś nie potrafił posługiwać się anonimowymi klasami wewnętrznymi, musiałby użyć następującego rozwiązania:

```
class MyIlcPriceAcceptor implements CameraAcceptor {  
    public boolean choose(Camera c) {  
        return c.isIlc() && c.getPrice() < 500;  
    }  
}  
  
CameraAcceptor myIlcPriceAcceptor = new MyIlcPriceAcceptor();  
results = searchApp.search(myIlcPriceAcceptor);
```

To naprawdę masa kodu do napisania — i to tylko po to, by przekazać jedną metodę do mechanizmu wyszukiwania. O wyposażenie języka Java w wyrażenia lambda bądź w domknięcia (ang. *closure*) postulowano na wiele (i to dosłownie) lat, zanim eksperci doszli do porozumienia, jak należy to zrobić. A efekt jest niewiarygodnie prosty. Jednym ze sposobów wyobrażenia sobie wyrażen lambda w Javie jest potraktowanie ich jako *metod implementujących interfejs funkcyjny*. Z wykorzystaniem wyrażen lambda powyższy kod można zapisać w następującej postaci:

```
results = searchApp.search(c -> c.isIlc() && c.getPrice() < 500);
```

² Czytelnikom, którzy nie interesują się aparatami fotograficznymi, wyjaśniam, że określenie „aparat z wymienionym obiektywem” obejmuje dwie kategorie aparatów, które obecnie, w 2014 roku, można znaleźć w sklepach: tradycyjne lustrzanki cyfrowe (ang. *Digital Single Lens Reflection*, DSLR) oraz nową kategorię „aparatów kompaktowych”, takich jak Nikon 1, Sony ILCE (znany wcześniej pod nazwą NEX) oraz Canon EOS-M, które są mniejsze i lżejsze od starszych aparatów DSLR.

Zapis ze strzałką (->) oznacza kod, który należy wykonać. Jeśli jest to proste wyrażenie, takie jak w powyższym przykładzie, to można je zapisać w przedstawionej postaci. Jeśli jednak w kodzie występuje instrukcja warunkowa lub inne instrukcje, to podobnie jak w zwyczajnym kodzie pisanym w Javie, trzeba będzie zastosować blok kodu:

```
results = searchApp.search(c -> {  
    if (c.isIllc() && c.getPrice() < 500)  
        return true;  
    else  
        return false;  
});
```

Pierwsze c umieszczone w nawiasach odpowiada parametrowi Camera c jawnie zaimplementowanej metody choose(): typ można pominąć, ponieważ kompilator go zna! Jeśli metoda ma więcej niż jeden argument, to należy je zapisać w nawiasach. Załóżmy, że dysponujemy metodą porównującą, która wymaga przekazania dwóch obiektów Camera i zwraca wartość liczbowa (życzę powodzenia komuś, kto spróbowałby doprowadzić do porozumienia dwóch fotografików odnośnie do sposobu działania *takiego* algorytmu!):

```
double goodness = searchApp.compare((c1, c2) -> {  
    // tu byłby zapisany nasz magiczny kod  
});
```

Można sądzić, że taki sposób zapisu wyrażeń lambda zapewnia ogromne możliwości — i faktycznie tak jest! Będzie można znaleźć bardzo wiele przykładów takich rozwiązań, jak tylko Java 8 zyska popularność.

Do tej pory dla każdej metody, która miała być stosowana w formie wyrażeń lambda, konieczne było napisanie odpowiedniego interfejsu. W następnej recepturze przedstawię kilka predefiniowanych interfejsów, których można użyć, by jeszcze bardziej uprościć (czyli także skrócić) swój kod.

No i koniecznie należy pamiętać, że dostępnych już jest całkiem dużo takich „funkcyjnych” interfejsów, jak na przykład interfejs ActionListener stosowany w aplikacjach z graficznym interfejsem użytkownika. Co ciekawe, zintegrowane środowisko programistyczne IntelliJ (patrz receptura 1.3) jest w stanie automatycznie rozpoznawać definicje klas wewnętrznych, które można by zastąpić wyrażeniami lambda, i w przypadku korzystania z opcji „zwijania kodu” (ang. *code folding*, możliwości pozwalającej na reprezentację definicji całej metody w jednym wierszu kodu) zastępuje taką klasę wewnętrzną wyrażeniem lambda! Rysunki 9.1 oraz 9.2 pokazują kod w jego początkowej postaci oraz po zwinięciu.

9.2. Stosowanie predefiniowanych interfejsów lambda zamiast własnych

Problem ⑧

Chcemy stosować wyrażenia lambda, używając przy tym nie własnych, lecz predefiniowanych interfejsów.

Rozwiązanie

Należy skorzystać z interfejsów funkcyjnych języka Java 8 zdefiniowanych w interfejsie `java.util.function`.

```

Wootage.java x
package Wootage;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 * Created with IntelliJ IDEA.
 * User: Ian
 * Date: 3/2/14
 * Time: 5:15pm
 * To change this template use File | Settings | File Templates.
 */
public class Wootage {
    public static void main(String[] args) {
        System.out.println("Witamy w aplikacji Java.");
    }

    ActionListener listener = new ActionListener() {
        public void actionPerformed(ActionEvent evt) {
            System.out.println("Źródło akcji " + evt.getSource());
        }
    };
}

```

Rysunek 9.1. Kod w środowisku IntelliJ w pełnej postaci

```

Wootage.java x
package Wootage;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 * Created with IntelliJ IDEA.
 * User: Ian
 * Date: 3/2/14
 * Time: 5:15pm
 * To change this template use File | Settings | File Templates.
 */
public class Wootage {
    public static void main(String[] args) {
        System.out.println("Witamy w aplikacji Java.");
    }

    ActionListener listener = (evt) -> { System.out.println("Źródło akcji " + evt.getSource()); };
}

```

Rysunek 9.2. Ten sam kod po zwinięciu

Analiza

W recepturze 9.1 stosowaliśmy metodę `acceptCamera()` zdefiniowaną w interfejsie `CameraAcceptor`. Metody o podobnym charakterze i działaniu są stosowane dosyć często, dlatego też pakiet `java.util.function` zawiera interfejs `Predicate<T>`, którego możemy użyć zamiast interfejsu `CameraAcceptor`. Interfejs ten definiuje tylko jedną metodę — `bool test(T t)`:

```

interface Predicate<T> {
    boolean test(T t);
}

```

Pakiet ten zawiera około 50 najczęściej stosowanych interfejsów funkcyjnych, takich jak `IntUnaryOperator`, który pobiera jeden argument typu `int` i zwraca wartość tego samego typu, lub `LongPredicate`, który pobiera wartość typu `long` i zwraca wynik typu `boolean` — i tak dalej.

Jak zawsze w przypadku stosowania typów ogólnych, aby skorzystać z interfejsu `Predicate`, należy użyć `Camera` (oczywiście w naszym przypadku) jako parametru typu, co da nam typ `Predicate<Camera>` przedstawiony w poniższym przykładzie (choć nie musimy go jawnie umieszczać w kodzie):

```
interface Predicate<Camera> {  
    boolean test(Camera c);  
}
```

A zatem naszą aplikację wyszukiującą możemy aktualnie zmienić tak, by udostępniała następującą metodę:

```
public List<Camera> search(Predicate p);
```

Na szczęście z punktu widzenia metod anonimowych implementowanych przez wyrażenia *lambda* ma ona taką samą sygnaturę co nasz interfejs `CameraAcceptor`, dzięki czemu pozostałe elementy naszego kodu ulegają zmianie! A zatem poniższa instrukcja wciąż stanowi poprawne wywołanie metody `search()`:

```
results = searchApp.search(c -> c.isIlc() && c.getPrice() < 500);
```

Poniżej została przedstawiona implementacja metody `search()`:

//functional/CameraSearchPredicate.java

```
public List<Camera> search(Predicate<Camera> tester) {  
    List<Camera> results = new ArrayList<>();  
    privateListOfCameras.forEach(c -> {  
        if (tester.test(c))  
            results.add(c);  
    });  
    return results;  
}
```

Załóżmy, że na każdym elemencie listy musi zostać wykonana tylko jedna operacja, a następnie cała lista zostanie usunięta. Po chwili zastanowienia dojdziemy do wniosku, że takiej listy wcale nie trzeba zwracać, a jedynie określić poszczególne elementy spełniające zadane warunki.

9.3. Upraszczanie przetwarzania z wykorzystaniem interfejsu `Stream`

Problem ⑧

Chcemy przetworzyć dane z wykorzystaniem mechanizmu przypominającego potoki.

Rozwiązanie

Należy użyć interfejsu `Stream` oraz jego metod.

Analiza

Strumienie (ang. *Streams*) są nowym mechanizmem wprowadzonym w języku Java 8, pozwalającym kolekcjom na przysyłanie swojej zawartości kolejno, element po elemencie, przez mechanizm przypominający potoki, gdzie mogą być przetwarzane, i to w sposób (w różnym stopniu) równoległy. Można wyróżnić trzy grupy metod związanych z wykorzystaniem strumieni:

- metody wytwarzające strumienie (patrz receptura 7.3);
- metody przekazujące, które operują na strumieniu i zwracają odwołanie do niego, pozwalając na tworzenie sekwencji wywołań; należą do nich takie metody jak: `distinct()`, `filter()`, `limit()`, `map()`, `peek()`, `sorted()`, `unsorted()` itd.;
- metody kończące działanie strumieni, które stanowią zakończenie wykonywanych na nich operacji; należą do nich takie metody jak: `count()`, `findFirst()`, `max()`, `min()`, `reduce()` czy też `sum()`.

Listing 9.1 przedstawia listę obiektów `Hero` reprezentujących superbohaterów w różnym wieku. Użyjemy metod interfejsu `Stream`, by wybrać tylko tych spośród nich, którzy są dorośli, i zsynchronizować ich wiek, a następnie w podobny sposób posortujemy ich imiona alfabetycznie.

W obu przypadkach rozpoczniemy od użycia generatora strumienia (metody `Arrays.stream()`), wykonamy na strumieniu kilka operacji, w tym operację odwzorowania (nie należy jej mylić z klasą `java.util.Map()`), która powoduje przesłanie do potoku innej wartości, a na samym końcu wywołamy operację kończącą. Operacje odwzorowywania oraz filtrowania niemal zawsze są kontrolowane przez wyrażenia lambda (stosowanie klas wewnętrznych w przypadku korzystania z tego stylu programowania byłoby zbyt męczące!).

Listing 9.1. `/functional/SimpleStreamDemo.java`

```
static Hero[] heroes = {
    new Hero("Grelber", 21),
    new Hero("Roderick", 12),
    new Hero("Franciszek", 35),
    new Hero("Superman", 65),
    new Hero("Jumbletron", 22),
    new Hero("Maverick", 1),
    new Hero("Palladyn", 50),
    new Hero("Atena", 50) };

public static void main(String[] args) {

    long adultYearsExperience = Arrays.stream(heroes)
        .filter(b -> b.age >= 18)
        .mapToInt(b -> b.age).sum();
    System.out.println("Jesteśmy w dobrych rękach! Dorośli " +
        "superbohaterowie mają w sumie " + adultYearsExperience +
        " lata doświadczeń.");

    List<Object> sorted = Arrays.stream(heroes)
        .sorted((h1, h2) -> h1.name.compareTo(h2.name))
        .map(h -> h.name)
        .collect(Collectors.toList());
    System.out.println("Superbohaterowie posortowani według imion: " +
        sorted);
}
```

Spróbujmy teraz wykonać powyższy program, aby przekonać się, czy działa prawidłowo:

Jesteśmy w dobrych rękach! Dorośli superbohaterowie mają w sumie 243 lata doświadczeń.
Superbohaterowie posortowani według imion: [Athena, Franciszek, Grelber, Jumbletron, Maverick, Palladyn, Roderick, Superman]

Kompletną listę dostępnych operacji można znaleźć w dokumentacji interfejsu `java.util.stream.Stream`.

9.4. Poprawianie przepustowości dzięki wykorzystaniu strumieni i kolekcji równoległych

Problem 8

Chcemy połączyć możliwości interfejsu `Stream` z przetwarzaniem współbieżnym, a przy tym wciąż móc korzystać z interfejsu programistycznego do obsługi kolekcji, który nie jest bezpieczny pod względem wielowątkowym.

Rozwiązanie

Należy użyć strumieni równoległych.

Analiza

Standardowe typy kolekcji, takie jak większość implementacji interfejsów `List`, `Set` oraz `Map`, nie zapewniają możliwości bezpiecznej wielowątkowej aktualizacji zawartości; jeśli w jednym wątku spróbujemy dodać jakiś obiekt do kolekcji lub go z niej usunąć, a jednocześnie inny wątek będzie się odwoływał do obiektów przechowywanych w tej samej kolekcji, to może to doprowadzić do awarii programu. Natomiast nic nie stoi na przeszkodzie, by w większej liczbie wątków jednocześnie odczytywać zawartość tej samej kolekcji. Zagadnienia związane z wielowątkowością opisałem w rozdziale 22.

Szkielet kolekcji udostępnia „klasy synchronizowane”, które zapewniają możliwość automatycznej synchronizacji wątków zyskiwaną kosztem wprowadzenia rywalizacji pomiędzy wątkami ograniczającej możliwości działania współbieżnego. Aby zapewnić możliwość efektywnego wykonywania operacji, należy skorzystać ze *strumieni współbieżnych*, które pozwalają na bezpieczne stosowanie standardowych typów kolekcji, o ile tylko *podczas przetwarzania kolekcji ich zawartość nie jest modyfikowana*.

Aby użyć takiego równoległego strumienia, wystarczy o niego poprosić. W tym celu zamiast metody `stream()`, z której skorzystaliśmy w recepturze 9.3, należy wywołać metodę `parallelStream()`.

W ramach przykładu założmy, że nasz interes z aparatami cyfrowymi świetnie się rozwija i musimy *naprawdę szybko* wyszukiwać aparaty na podstawie typu i zakresu cen (przy okazji używając krótszego i prostszego kodu niż wcześniej):

/functional/CameraSearchParallelStream.java

```
public static void main(String[] args) {  
    for (Object camera : privateListOfCameras.parallelStream()) {  
        filter(c -> c.isIlc() && c.getPrice() < 500).  
            toArray() {  
                System.out.println(camera);  
            }  
    }  
}
```

- ❶ Tworzymy strumień równoległy na podstawie listy (List) obiektów Camera. Zawartość kolekcji zwróconej przez strumień zostanie następnie pobrana w pętli „foreach”.
- ❷ Filtrujemy aparaty na podstawie ceny, używając przy tym tego samego wyrażenia lambda typu Predicate, który stosowaliśmy już w recepturze 9.1.
- ❸ Kończymy działanie strumienia, konwertując go na tablicę.
- ❹ Wewnątrz pętli „foreach” wyświetlamy kolejno poszczególne aparaty zwrócone przez strumień.



Powyższy kod będzie działał niezawodnie wyłącznie w przypadku, jeśli żaden wątek nie spróbuje zmodyfikować zawartości danych w trakcie ich przeszukiwania. Informacje o tym, jak to zapewnić, korzystając z mechanizmu blokowania wątków, można znaleźć w rozdziale 22.

9.5. Tworzenie własnych interfejsów funkcyjnych

Problem ⑧

Chcemy napisać interfejs funkcyjny, którego moglibyśmy używać do tworzenia wyrażeń lambda.

Rozwiązanie

Należy stworzyć interfejs definiujący jedną metodę abstrakcyjną i opcjonalnie dodać do niego adnotację @FunctionalInterface.

Analiza

Jak już wspominałem wcześniej, interfejs funkcyjny to interfejs deklarujący jedną metodę abstrakcyjną. Do niektórych powszechnie znanych interfejsów funkcyjnych należą: `java.lang.Runnable`, `java.util.Observer` oraz `java.awt.event.ActionListener`. Z kolei przykładami interfejsów „niefunkcyjnych” są: `java.util.Observable` oraz `java.awt.event.WindowListener`, gdyż każdy z nich definiuje więcej niż jedną metodę.

Tworzenie własnych interfejsów funkcyjnych nie jest trudne. Zanim jednak się za to zabierzemy, należy pamiętać, że bardzo wiele takich interfejsów istnieje w JDK! Zgodnie z informacjami podanymi w recepturze 9.2 warto sprawdzić dokumentację interfejsu `java.util.function`, który udostępnia wiele predefiniowanych interfejsów funkcyjnych ogólnego przeznaczenia, w tym zastosowany już wcześniej interfejs `Predicate`.

My jednak mimo wszystko chcemy zdefiniować własny interfejs funkcyjny. Poniżej przedstawiłem prosty przykład takiego interfejsu:

/functional/ProcessIntsUsingFunctional.java

```
interface MyFunctionalInterface {
    int compute(int x);
}
```

Można by go użyć w poniższym programie do przetworzenia tablicy liczb całkowitych:

/functional/ProcessIntsUsingFunctional.java

```
static int[] integers = {1, 2, 3};

public static void main(String[] args) {
    int total = 0;
    for (int i : integers)
        total += process(i, x -> x * x + 1);
    System.out.println("Suma wynosi " + total);
}

private static int process(int i, MyFunctionalInterface o) {
    return o.compute(i);
}
```

Gdyby interfejs zawierający metodę `compute()` nie był interfejsem funkcyjnym — gdyby deklarował więcej niż jedną metodę — to nie można by go użyć w taki sposób.

Aby mieć możliwość zagwarantowania, że dany interfejs jest interfejsem funkcyjnym i takim pozostanie, została stworzona adnotacja `@FunctionalInterface`, która jest stosowana tak samo jak adnotacja `@Override` (obie zostały zdefiniowane w pakiecie `java.lang`). Jest to adnotacja opcjonalna, która jest używana w celu zapewnienia, że dany interfejs będzie spełniał wymogi narzucane interfejsom funkcyjnym. Można by ją dodać do naszego interfejsu z poprzedniego przykładu w następujący sposób:

```
@FunctionalInterface
interface MyFunctionalInterface {
    int compute(int x);
}
```

Gdyby później ktoś pracujący nad kodem dodał do niego kolejną metodę, na przykład:

```
int recompute(int x);
```

to interfejs przestałby spełniać warunki interfejsu funkcyjnego, jednak kompilator lub zintegrowane środowisko programistyczne wykryłoby to bezpośrednio po zapisaniu pliku lub jego kompilacji, pozwalając programiście zaoszczędzić czas konieczny na określenie, dlaczego wyrażenia lambda przestały działać. Poniżej przedstawiłem komunikat, który w takim przypadku wygenerowałby kompilator `javac`:

```
C:\javasrc>javac -d build src\lang\MyFunctionalInterface.java
src\lang\MyFunctionalInterface.java:3: error: Unexpected @FunctionalInterface
annotation
@FunctionalInterface
^
MyFunctionalInterface is not a functional interface
multiple non-overriding abstract methods found in
interface MyFunctionalInterface

1 error

C:\javasrc>
```

Oczywiście, czasem może się zdarzyć, że nasz interfejs naprawdę będzie musiał mieć więcej niż jedną metodę. W takich przypadkach złudzenie (lub efekt) funkcyjności interfejsu można zachować poprzez wskazanie metody „domyślnej” — poprzedzenie jej słowem kluczowym `default`. drugiej metody takiego interfejsu wciąż będzie można używać w wyrażeniach lambda.

```
public interface ThisIsStillFunctional {  
    default int compute(int ix) { return ix * ix + 1 };  
    int anotherMethod(int y);  
}
```

W interfejsach funkcyjnych tylko domyślne metody mogą zawierać instrukcje, a w każdym takim interfejsie może istnieć jedna metoda, w której definicji nie ma słowa kluczowego `default`.

Poza tym przedstawiony wcześniej interfejs `MyFunctionalInterface` można z powodzeniem zastąpić domyślnym interfejsem `java.util.IntUnaryOperator`, zmieniając także nazwę metody z `compute()` na `applyAsInt()`. W przykładach dołączonych do książki w katalogu */functional* dostępna jest wersja programu korzystająca z tego interfejsu — `ProcessIntsIntUnaryOperator`.

Patrz także

Domyślnych metod definiowanych w interfejsach można także używać do wstawiania kodu do innych klas, co opisałem w recepturze 9.7.

9.6. Używanie istniejącego kodu w sposób funkcyjny dzięki wykorzystaniu odwołań do metod

Problem ⑧

Dysponujemy już istniejącym kodem spełniającym warunki interfejsów funkcyjnych i chcielibyśmy używać go bez konieczności dopasowywania nazw metod do tych zdefiniowanych w interfejsie.

Rozwiązanie

Należy zastosować odwołania do funkcji, takie jak `MyClass::myFunc` lub `someObj::someFunc`.

Analiza

Słowo „odwołanie” w języku Java ma równie wiele znaczeń co słowo „sesja”. Zastanówmy się:

- Ze zwyczajnych obiektów korzystamy zazwyczaj, używając odwołań.
- Typy referencyjne, takie jak `WeakReference`, mają ściśle określone znaczenie dla mechanizmu odzyskiwania pamięci.
- W języku Java 8 pojawiła się zupełnie nowa możliwość odwoływania się do konkretnych metod.
- Można się nawet odwołać do „metody instancyjnej dowolnego obiektu określonego typu”.

Nowa składnia pozwalająca na tworzenie takich odwołań składa się z nazwy obiektu lub klasy, dwóch znaków dwukropka oraz nazwy metody, do której chcemy się odwołać w kontekście obiektu lub klasy (zgodnie ze standardowymi regułami języka Java, stosując nazwę klasy, można się odwoływać do jej metod statycznych, natomiast stosując nazwę zmiennej obiektowej — do jej metod instancyjnych). Aby odwołać się do konstruktora, należy użyć słowa kluczowego `new`, na przykład `MyClass::new`. Takie odwołanie tworzy wyrażenie lambda, które można wywołać, zapisać w zmiennej, której typem będzie interfejs funkcyjny, i tak dalej.

W przykładzie przedstawionym na listingu 9.2 tworzymy odwołanie typu `Runnable`, które zamiast standardowej metody `run` zawiera metodę `walk` mającą ten sam typ wartości wynikowej i argumentów. Warto zwrócić uwagę na zastosowanie `this` jako określenia obiektu podczas tworzenia odwołania. Następnie obiekt `Runnable` przekazujemy w wywołaniu konstruktora `Thread` i uruchamiamy wątek — w efekcie zamiast metody `run` zostanie wywołana metoda `walk`.

Listing 9.2. */functional/ReferencesDemo.java*

```
/** "Chodź, nie biegaj" */
public class ReferencesDemo {

    // Zakładamy, że to jest istniejąca metoda, której nazwy nie
    // chcemy zmieniać.
    public void walk() {
        System.out.println("ReferencesDemo.walk(): zastępuje wywołanie metody
                               run.");
    }

    // To jest nasza główna metoda, która wykonuje metodę walk w nowym
    // wątku.
    public void doIt() {
        Runnable r = this::walk;
        new Thread(r).start();
    }

    // Zwyczajna, bardzo prosta metoda main, która wszystko uruchomi.
    public static void main(String[] args) {
        new ReferencesDemo().doIt();
    }
}
```

Oto wyniki wykonania tego programu:

```
ReferencesDemo.walk(): zastępuje wywołanie metody run.
```

Przykład przedstawiony na listingu 9.3 tworzy obiekt `AutoCloseable` przeznaczony do użycia w instrukcji `try` zarządzającej zasobami (patrz punkt „Instrukcja `try` zarządzająca zasobami” w podrozdziale „Nowości wprowadzone w wersji Java 7” dodatku A). Interfejs `AutoCloseable` zawiera metodę `close()`, jednak w naszym przykładzie metoda ta ma nazwę `cloz()`. Używana w programie zmienna referencyjna typu `AutoCloseable` ma nazwę `autoCloseable` i jest tworzona wewnątrz instrukcji `try`, co oznacza, że jej metoda udająca metodę `close()` zostanie wywołana po zakończeniu realizacji bloku `try`. W tym przypadku znajdujemy się w statycznej metodzie `main()` i dysponujemy zmienną referencyjną `rnd2` zawierającą odwołanie do obiektu naszej klasy, zastosujemy zatem tę zmienną do utworzenia odwołania do metody zgodnej z interfejsem `AutoCloseable`.

Listing 9.3. */functional/ReferencesDemo2.java*

```
public class ReferencesDemo2 {
    void cloz() {
        System.out.println("Zamiast wywołania metody close().");
    }

    public static void main(String[] args) throws Exception {
        ReferencesDemo2 rd2 = new ReferencesDemo2();

        // Używamy odwołania do metody w celu przypisania do zmiennej
        // typu AutoCloseable "autoCloseable" odwołania do metody
        // o zgodnej sygnaturze "c" (oczywiście chodzi o metodę close,
        // lecz chcę pokazać, że nazwa metody nie ma w tym
        // przypadku znaczenia).
        try (AutoCloseable autoCloseable = rd2::cloz) {
            System.out.println("Wykonujemy jakieś działania.");
        }
    }
}
```

Oto wyniki wykonania tego programu:

```
Wykonujemy jakieś działania.
Zamiast wywołania metody close().
```

Oczywiście istnieje także możliwość stosowania takich rozwiązań, które wykorzystują własne interfejsy funkcyjne, takie jak ten przedstawiony w recepturze 9.5. Czytelnik na pewno też jest świadomy, a przynajmniej domyśla się, że dowolne odwołanie do obiektu w języku Java można przekazać w wywołaniu metody `System.out.println()`, a w efekcie zostanie wyświetlony jakiś opis obiektu. Oba te zagadnienia zostały przedstawione w przykładzie z listingu 9.4. Zdefiniowaliśmy w nim interfejs funkcyjny o nazwie `FunInterface`, którego metoda wymaga przekazania kilku argumentów (w zasadzie tylko po to, by nie można go pomylić z już istniejącymi interfejsami funkcyjnymi). Metoda nosi nazwę `process`, jednak — jak już wiemy — nazwa ta nie ma większego znaczenia — jej implementacja w programie nosi nazwę `work`. Jest to metoda statyczna, przez co nie możemy stwierdzić, że nasza klasa `ReferencesDemo3` implementuje interfejs `FunInterface` (mimo że nazwy metod są takie same — metoda statyczna nie może bowiem przesłonić odziedziczonej metody instancyjnej). Okazuje się jednak, że możemy utworzyć odwołanie lambda do metody `work`. Następnie przekazujemy to odwołanie w wywołaniu metody `println()`, pokazując tym samym, że jego struktura odpowiada obiektowi języka Java.

Listing 9.4. */functional/ReferencesDemo3.java*

```
public class ReferencesDemo3 {

    interface FunInterface {
        void process(int i, String j, char c, double d);
    }

    public static void work(int i, String j, char c, double d){
        System.out.println("Muuu");
    }

    public static void main(String[] args) {
        FunInterface sample = ReferencesDemo3::work;
        System.out.println("Główna metoda obliczeniowa: " + sample);
    }
}
```

Poniżej przedstawiłem wyniki generowane przez ten program:

```
Główna metoda obliczeniowa: functional.ReferencesDemo3$$Lambda$1/918221580@4517d9a3
```

Fragment `Lambda$1` w wyświetlonej nazwie odpowiada identyfikatorom `$1` stosowanym w anonimowych klasach wewnętrznych.

Ostatni rodzaj odwołań do funkcji opisany w dokumentacji Javy, określane jako odwołanie do „metody instancyjnej dowolnego obiektu konkretnego typu”, jest chyba najbardziej zawiłą nowością wprowadzoną w języku Java 8. Pozwala ona na zadeklarowanie odwołania do metody instancyjnej, jednak bez określania, o który obiekt chodzi. Oznacza to, że można jej używać z dowolnym obiektem danej klasy! W przykładzie przedstawionym na listingu 9.5 mamy tablicę łańcuchów znaków, którą chcemy posortować. Ponieważ nazwiska podane w tablicy mogą się zaczynać zarówno od małych, jak i od wielkich liter, chcemy je posortować, używając metody `compareToIgnoreCase()` klasy `String`, która nie uwzględnia wielkości liter.

Ponieważ chciałem pokazać kilka różnych sposobów sortowania, utworzyłem dwa odwołania do tablicy: pierwsze — do oryginalnej, nieposortowanej tablicy, a drugie — do kopii roboczej, którą będziemy odtwarzać, sortować i wyświetlać, używając metody pomocniczej (nie przedstawiałem tu jej kodu, gdyż jest to zwyczajna pętla `for` wyświetlająca łańcuchy znaków z przekazanej tablicy).

Listing 9.5. /functional/ReferecesDemo4.java

```
import java.util.Arrays;
import java.util.Comparator;

public class ReferencesDemo4 {

    static final String[] unsortedNames = {
        "Gosling", "de Raadt", "Torvalds", "Ritchie", "Hopper"
    };

    public static void main(String[] args) {
        String[] names;

        // Sortowanie z wykorzystaniem
        // "metody instancyjnej dowolnego obiektu konkretnego typu"
        names = unsortedNames.clone();
        Arrays.sort(names, String::compareToIgnoreCase);           ❶
        dump(names);

        // Analogiczne sortowanie z użyciem wyrażenia lambda:
        names = unsortedNames.clone();
        Arrays.sort(names, (str1, str2) -> str1.compareToIgnoreCase(str2)); ❷
        dump(names);

        // Analogiczne sortowanie wykonane w standardowy sposób:
        names = unsortedNames.clone();
        Arrays.sort(names, new Comparator<String>() {               ❸
            @Override
            public int compare(String str1, String str2) {
                return str1.compareToIgnoreCase(str2);
            }
        });
        dump(names);
    }
}
```

```
// Najprostszy sposób sortowania, z użyciem istniejącego komparatora.
names = unsortedNames.clone();
Arrays.sort(names, String.CASE_INSENSITIVE_ORDER);
dump(names);
}
```

- ❶ Używając „metody instancyjnej dowolnego obiektu konkretnego typu”, deklarujemy odwołanie do metody `compareToIgnoreCase` dowolnego obiektu `String` użytego w wywołaniu.
- ❷ Przedstawia analogiczne sortowanie wykonane przy użyciu wyrażenia lambda.
- ❸ Pokazuje sposób, „którego używali nasi dziadkowie, pisząc programy w Javie”.
- ❹ To przykład bezpośredniego użycia wyeksportowanego komparatora, który pokazuje, że wszystko można zrobić na kilka sposobów.

Na wszelki wypadek wykonałem powyższy przykład i uzyskałem następujące wyniki:

```
de Raadt Gosling Hopper Ritchie Torvalds
de Raadt Gosling Hopper Ritchie Torvalds
de Raadt Gosling Hopper Ritchie Torvalds
de Raadt Gosling Hopper Ritchie Torvalds
```

9.7. Wstawianie istniejącego kodu metod

Problem ⑧

Słyszeliśmy o możliwości tworzenia „wstawek”, czyli wykorzystywania istniejących metod w innych klasach, i chcielibyśmy z niej skorzystać.

Rozwiązanie

Należy skorzystać z importu statycznego lub zadeklarować jeden bądź więcej interfejsów funkcyjnych z metodami „domyślnymi” zawierającymi kod, którego chcemy użyć, a następnie zaimplementować te metody.

Analiza

Programiści używający innych języków programowania często sztydzi z Javy, wyśmiewając brak możliwości tworzenia tak zwanych „wstawek” (ang. *mixin*), czyli wykorzystywania fragmentów kodu pochodzącego z innych typów.

Jedną z możliwości uzyskania takiego efektu jest skorzystanie z „importu statycznego”, która jest dostępna w Javie już od dekady. Jest ona powszechnie stosowana w tekstach jednostkowych (patrz receptura 1.13). Rozwiązanie to ma jednak tę wadę, że pozwala na wykorzystywanie wyłącznie metod statycznych, a nie instancyjnych.

Nowszy mechanizm korzysta z interesującego efektu ubocznego, będącego konsekwencją zmian wprowadzonych w języku Java 8, związanych z obsługą wyrażeń lambda: pozwala on na dołączanie do typu kodu zaimplementowanego w zupełnie odrębnych, niezwiązanych z nim typach danych. Czy twórcy Javy w końcu zrezygnowali ze swojego nieustępliwego sprzeciwu wobec wielokrotnego dziedziczenia? Na pierwszy rzut oka mogłoby się tak wydawać, ale spokojnie: nasze możliwości ograniczają się do wykorzystywania metod z wielu

interfejsów, a nie z wielu klas. Gdyby Czytelnik jeszcze nie zorientował się, że w interfejsach można definiować metody (a nie jedynie deklarować je), to powinien zajrzeć do receptury 9.5. Przeanalizujemy następujący przykład:

/lang/MixinsDemo.java

```
interface Bar {
    default String filter(String s) {
        return "Przefiltrowane " + s;
    }
}

interface Foo {
    default String convolve(String s) {
        return "zwinięte " + s;
    }
}

public class MixinsDemo implements Foo, Bar{

    public static void main(String[] args) {
        String input = args.length > 0 ? args[0] : "Witam";
        String output = new MixinsDemo().process(input);
        System.out.println(output);
    }

    private String process(String s) {
        return filter(convolve(s)); // Wywołanie wstawionych metod!
    }
}
```

Poniżej przedstawiłem wyniki wykonania tego programu:

```
C:\javasrc>javac -d build lang/MixinsDemo.java
C:\javasrc>java -cp build lang.MixinsDemo
Przefiltrowane zwinęte Witam
```

```
C:\javasrc>
```

No i proszę — obecnie Java już obsługuje wstawki!

Czy to oznacza, że mamy jak szaleni tworzyć interfejsy zawierające implementację metod? Nie. Trzeba pamiętać, że rozwiązanie to opracowano z myślą o tworzeniu „interfejsów funkcyjnych” wykorzystywanych w wyrażeniach lambda. Stosowane z umiarem faktycznie pozwala na tworzenie wstawek i konstruowanie aplikacji w nieco inny sposób niż przy wykorzystaniu tradycyjnego dziedziczenia, agregacji oraz technik programowania aspektowego. Jednak nadużywanie tej techniki może prowadzić do powstania nieczytelnego kodu, doprowadzać do szaleństwa programistów przyzwyczajonych do starszych wersji Javy i wprowadzać chaos.

A

- adnotacja, 700, 787
 - @Column, 790
 - @Component, 293
 - @Entity, 790
 - @FunctionalInterface, 310
 - @Id, 625, 790
 - @Override, 788
 - @Resource, 293
 - @Test, 67
 - @WebService, 463
- adnotacje
 - Java 5, 836
 - JAXB, 673
 - JPA, 789
- adres
 - domenowy, 694
 - IP, 443
 - karty sieciowej, 557
 - URL, 459
- aktualizacja stanu tablicy, 736
- algorytm Soundex, 127
- analiza
 - argumentów, 87
 - danych, 156
 - JSON, 664, 665
 - wejściowych, 335, 338
 - dokumentów XML, 681
 - łańcuchów znaków, 212
 - pliku, 331
 - pliku dziennika, 155
 - przepływu, 61
 - składniowa XML, 679
- analyzer syntaktyczny, 339, 679
- Android, 847
- animacja, 728
- animator, 728
- anonimowe klasy wewnętrzne, 303, 486
- AOP, aspect-oriented programming, 265
- Apache, 568
- Apache Subversion, 49
- API, 205, 263, 843
- API SAX, 679
- aplety, 704, 796, 846
- aplikacje
 - serwerowe, 441
 - sieciowe, 439, 710
- archiwa, 702
- archiwum
 - CPAN, 817
 - JAR, 43, 54, 368
 - sieciowe, 710
 - ZIP, 368
- arkusze stylów XSL, 672
- artefakty, 464
- ASCII, 108, 326, 446
- asercja, 63
- atak typu DoS, 574
- audio, 424
- automatyczne oczyszczanie pamięci, 825
- automatyzacja
 - kompilacji, 50, 53, 56
 - testowania, 53, 56
 - wdrażania, 53, 56
 - zależności, 53, 56
- AWT, Abstract Window Toolkit, 475

B

- badanie adnotacji, 790
- baza danych, 621
 - JDBC, 631
 - PostgreSQL, 660
- bezpieczeństwo sieci, 586
- biblioteka
 - Apache Commons StringUtils, 129
 - AWT, 414

- biblioteka
 - HttpClient, 460
 - Java Media Framework, 426
 - JavaFX, 429
 - log4j, 572, 576
 - netlog, 573
 - SLF4J, 574
- biblioteki dodatkowe, 77
- bieżąca data, 208
- BlackBerry OS, 847
- blok
 - eval { }, 819
 - try-catch, 321
- blokada, deadlock, 723, 741, 747
- blokada czytelnika-pisarza, 742
- błąd
 - kompilatora, 284
 - logiczny, 173
 - zaokrąglenia, 166, 175
- błędy sieciowe, 445

C

- CDI, Context and Dependency Injection, 291, 293
- certifikat cyfrowy, 571, 718
- ciągła integracja, 69
- CVS, Concurrent Versions System, 49
- cyfry rzymskie, 183
- cykl życiowy wątku, 727
- cykliczny zapis danych, 756
- czas, 207
 - wykonania programu, 262, 273, 780
- czcionka, 411
 - PostScript, 417
 - TrueType, 417

D

- dane
 - binarne, 448
 - serializowane, 450
 - tekstowe, 446
 - XML, 672, 675
- darwinysys-api, 45
- datagramy UDP, 452
- daty i godziny, 205
- definicja
 - adnotacji, 787, 790
 - typu dokumentu, DTD, 644, 686, 688
 - wzorców, 180
- deklarowanie tablic, 132
- dekorator HTML, 650

- delegowanie, 265
- deskryptory plików, 321
- DNS, Domain Name System, 558
- dodawanie
 - dat, 214
 - pliku do zmiennej, 87
- dokładność liczb zmiennoprzecinkowych, 173
- dokument XML, 671
- dokumentacja, 264, 554
 - języka, 83
 - klas, 696
- dołączanie
 - końca rodzimego, 820
 - plików, 440
- DOM, Document Object Model, 681
- domyślne ustawienia lokalne, 546
- dopasowywanie
 - wzorców, 131
 - znaków, 152
- DoS, Denial of Service, 574
- dostęp do
 - bazy danych, 621, 623, 650
 - danych, 224, 233
 - formularza, 740
 - kolekcji, 230
 - komunikatów, 573
 - prywatnych pól, 774
- drukowanie, 430
- drzewa binarne, 235
- DSL, domain-specific language, 58
- DTD, Document Type Definition, 644, 686, 688
- dynamiczna strona WWW, 72
- dynamiczne ładowanie klas, 769, 775
- dyrektywa #ifdef, 61
- działanie równoległe, 753
- dźwięk, 407

E

- Eclipse, 36, 45, 69
- ECMAScript, 812
- EDT, event dispatching thread, 479
- edycja programów, 33
- edytor kwrite, 804
- efektywność działania programu, 779
- EJB, Enterprise JavaBeans, 67
- EOF, End Of File, 331
- epoka, 211
- etykieta ekranowa, 511
- ewolucja zastosowań Javy, 846

F

- fabryka, 461
- flaga
 - CANON_EQ, 152
 - MULTILINE, 153
 - UNIX_LINES, 153
- flagi metody `Pattern.compile()`, 151
- fonetyczne porównywanie nazwisk, 127
- FOP, Formatting Objects Processor, 672
- format
 - CSV, 120
 - DB, 629
 - DB/DBM, 622
 - DBM, 621
 - HTML, 676
 - JSON, 382, 441, 661–666
 - JWS, 716
 - MIME, 599
 - PostScript, 383
 - XML, 382, 441
 - ZIP, 368
- formatowanie, 328, 329
 - daty i czasu, 209
 - komponentów, 511
 - komunikatów, 547
 - liczb, 178
 - obiektów, 266
 - tekstów, 125
 - wyjątków, 496
- formaty pakietów TFTP, 455
- formularz HTML, 591
- FP, functional programming, 299
- funkcja, *Patrz* metoda
- funkcje
 - czyste, 300
 - pierwszej klasy, 300
 - trygonometryczne, 192
 - zwrotne, 303

G

- generator
 - analizatorów syntaktycznych, 338
 - kompilatorów, 339
- generowanie
 - artefaktów klienta, 464
 - dokumentacji, 698
 - grafiki dwuwymiarowej, 407
 - kodu XML, 689
 - liczb pseudolosowych, 167, 189
 - liczbowych palindromów, 201
 - odstępów, 117
 - testów jednostkowych, 67

- geolokalizacja adresu IP, 460
- Git, 49
- GML, General Markup Language, 669
- gniazda, 439, 555
 - asynchroniczne, 439
 - spotkania, 563
- Google Guice, 291
- graficzny interfejs użytkownika, GUI, 68, 242, 302, 475–534
 - formatowanie komponentów, 511
 - karty, 483
 - klasa
 - JColorChooser, 509
 - JFileChooser, 506
 - JOptionPane, 494
 - JSpinner, 505
 - JTextArea, 499
 - metoda przeciągnij i upuść, 522
 - obsługa czynności, 486, 488
 - przycisk Zamknij, 489
 - przyciski, 484
 - sposób prezentacji programu, 515
 - układ okna, 480
 - uruchamianie, 478
 - wyjątki, 496
 - wyświetlanie komponentów, 477
- grafika, 407
- grafika dwuwymiarowa, 415
- gramatyka, 338

H

- Hibernate, 624, 625
- hierarchia klas, 276
- HotJava, 831

I

- IDE, Integrated Development Environment, 31, 42, 310, 476
 - Eclipse, 36, 45, 69
 - IntelliJ IDEA, 36, 38
 - NetBeans, 36
- identyfikator URI, 459
- ikony, 716
- informacje
 - o adresach sieciowych, 443
 - o dacie, 211
 - o danych, 647
 - o katalogach głównych, 399
 - o klasie, 784
 - o obiektach, 562
 - o pliku, 388

- informacje
 - o strukturze plików, 368
 - o zmianach pliku, 402
 - z właściwości systemowych, 81
- inkrementacja, 281
- instalowanie
 - klas, 775
 - oprogramowania, 710, 711
- instrukcja
 - if, 62
 - switch, 843
 - try, 312, 842
- IntelliJ IDEA, 36, 38
- interfejs, 276, 277
 - ActionListener, 300, 304, 485
 - AutoCloseable, 312, 842, 843
 - Callable, 765
 - CameraAcceptor, 305
 - Closeable, 843
 - Collection, 252
 - Comparable, 247
 - Comparator, 247
 - DocumentHandler, 679
 - Enumeration, 232
 - Executor, 765
 - FunInterface, 313
 - Iterable, 255
 - Iterator, 99, 230, 233, 253
 - LayoutManager, 481, 529
 - List, 232
 - ListIterator, 235
 - Lock, 745
 - Map, 839
 - MDI, 618
 - MyFunctionalInterface, 311
 - Predicate<T>, 305
 - Queue, 750
 - RowSet, 642
 - Runnable, 723, 765
 - ScriptEngine, 813
 - ScriptEngineFactory, 813, 814
 - Serializable, 363
 - Set, 232
 - Stream, 306, 308
 - WindowListener, 489, 490
- interfejsy
 - DOM, 681
 - funkcyjne, 299–304, 309
 - modelu, 291
 - rozszerzające RowSet, 643
 - sieciowe, 557
 - widoku, 292
 - zdalne, 276

- programistyczne, 205
 - JavaHelp, 699
 - JDOM, 671
 - SAX, 671, 681
 - XPath, 684
- introspekcja, 769, 774
- IP, Internet Protocol, 452
- iterator, 99, 230, 233, 253
- izolacja transakcji, 648

J

- J2EE, 832
- Java 1.0, 831
- Java 1.1, 832
- Java 1.3, 833
- Java 1.4, 833
- Java 2, 832
- Java 5, 834
- Java 6, 840
- Java 7, 841, 843
- Java 8, 844
- Java Communications API, 374, 375
- Java Compiler API, 840
- Java EE, 555
- Java Logging API, 572
- Java Media Framework, 407, 426
- Java Messaging Services, 471
- Java Micro Edition, 847
- Java Print Service API, 430
- Java SDK, 32
- Java SE, 302
- Java Web, 302
- Java Web Start, 711, 713, 716
- Javadoc, 264, 696
- JavaFX, 429, 846
- JavaHelp, 699
- JavaMail Extension, 587
- javasrc, 45, 47
- JAXB, Java Architecture for XML Bindings, 672, 841
- JAX-WS, 461, 841
- JCL, Jakarta Commons Logging, 573
- JCP, Java Community Process, 693
- JDBC, Java DataBase Connectivity, 621, 629, 841
- JDEE, Java Development Environment for Emacs, 34
- JDK, Java Development Kit, 32, 75, 82
- JDOM, 671
- JDOM API, 681
- Jenkins, 70–72
- język
 - BeanShell, 803
 - C, 360, 826
 - Clojure, 804

- CSP, 767
- DSL, 58
- Erlang, 767
- GML, 669
- Go, 767
- Groovy, 803
- HTML, 669
- JavaScript, 811
- IRuby, 803
- Jython, 803
- MIF, 691
- Perl, 817
- Perl 5, 813
- PostScript, 407
- R, 812
- Renjin, 804
- Scala, 804
- SGML, 670
- SQL, 629
- WSDL, 462
- XML, 669–692
- XSL, 672
- języki
 - dynamiczne, 803
 - programowania funkcyjnego, 299
- JILT, Java Internationalization and Localization Toolkit, 544
- JIT, Just In Time, 833
- JMS, Java Messaging Service, 555
- JNDI, Java Naming and Directory Interface, 291
- JNI, Java Native Interfejs, 820, 826
- JNLP, Java Net Launch Protocol, 716, 718
- JPA, Java Persistence API, 623, 625, 789
- JRE, Java Runtime Environment, 31
- JSF, JavaServer Faces, 77, 555
- JSON, JavaScript Object Notation, 661
- JSP, JavaServer Pages, 555
- JSP JavaBeans, 705
- JSR, Java Specification Request, 722
- JSSE, Java Secure Socket Extension, 570
- JVM, 711, 769, 845
- JWS, Java Web Start, 714, 716

K

- kalkulator, 336, 465
- karty, 483
- katalog
 - numbers, 45
 - regexp, 45
 - strings, 45

- klasa
- AffineTransform, 417
 - Applet, 419
 - ArrayList, 223
 - AudioClip, 407
 - BigDecimal, 198
 - BigInteger, 198
 - BitSet, 218
 - BufferedInputStream, 320
 - Button, 523
 - Calendar, 205, 215
 - CallableStatement, 641
 - ChessMoveException, 290
 - ChoiceFormat, 188
 - Class, 769
 - ClassesInPackage, 786
 - ClassLoader, 371, 716, 777
 - Collection, 218
 - Collections, 289
 - CommPort, 375
 - Complex, 196
 - Component, 407, 409
 - Configuration, 673
 - ConnectException, 445
 - Console, 323
 - Constructor, 771
 - CrossRef, 796
 - CSVImport, 120
 - DatagramPacket, 452
 - DatagramSocket, 452
 - DataInputStream, 359
 - DataOutputStream, 358
 - Date, 60, 215
 - DateTimeFormatter, 209
 - DaytimeText, 446
 - Debug, 112
 - DecimalFormat, 180
 - EnTab, 112
 - EntryLayout, 529
 - Enum, 285
 - EscContLineReader, 355
 - EventQueue, 479
 - Exception, 290
 - ExecAndPrint, 809
 - ExtensionFileFilter, 508
 - FancyClassJustToShowAnnotation, 792
 - Field, 771
 - File, 84, 85, 387
 - FileInputStream, 340
 - FileIO, 341
 - FileOutputStream, 340, 391
 - FileProperties, 244, 602
 - FileReader, 340

klasa

- FileWriter, 340, 391
- Fmt, 127
- Font, 423
- FontChooser, 524
- Formatter, 317, 327, 328
- GetOpt, 88–94
- Graphics, 407, 408
- Graphics2D, 415
- GZIPInputStream, 373
- GZIPOutputStream, 373
- Handler, 564, 760
- HashMap, 238
- Hashtable, 224, 238
- Image, 419
- ImageIcon, 423
- ImageIO, 423
- IndentContLineReader, 355
- InetAddress, 443
- InputStreamReader, 351
- JColorChooser, 509, 510
- JDialog, 495
- JDOM, 690
- JFileChooser, 506
- JFrame, 414, 477
- JOptionPane, 494
- JSpinner, 505
- JTabbedPane, 483
- JTextArea, 499, 502
- JTextAreaWriter, 502
- LabelText, 706
- List, 221
- Locale, 178, 535
- LocalTime, 213
- Ls, 398
- Mailer, 594
- MailReaderBean, 614
- Map, 221
- Matcher, 141
- Math, 167
- Matrix, 194
- Message, 587
- MessageFormat, 547
- Method, 771
- MethodHandle, 844
- MutableInteger, 281, 282
- NetworkInterface, 581
- NoRouteToHostException, 445
- NumberFormat, 166, 178
- Object, 223, 745
- ObjectInputStream, 363
- ObjectOutputStream, 359, 363
- OutputStream, 500
- OutputStreamWriter, 351
- Part, 599
- Path, 387, 401
- Pattern, 141
- PersonTest, 67
- Pipe, 466
- Plotter, 295
- PlotterAWT, 434
- PluginsViaAnnotations, 793
- Preferences, 240, 382
- PrintStream, 325, 348
- Process, 811
- ProcessBuilder, 807
- Properties, 242, 602
- RandomAccessFile, 359
- RecursiveAction, 753
- RecursiveTask, 753
- ResourceBundle, 543
- ResultSetMetaData, 647
- RomanNumberFormat, 184
- RuntimeException, 290
- ScaledNumberFormat, 187
- Scanner, 317, 321, 335
- SecurityManager, 802
- ServerSocket, 555–557
- Session, 587
- Set, 221
- Socket, 441, 444
- Soundex, 127, 128
- SpinnerEditor, 505
- Sprite, 729
- SSLServerSocketFactory, 570
- Stack, 110, 199, 256
- Store, 587, 603
- Stream, 301
- StreamTokenizer, 332
- String, 95, 109, 116
- StringAlign, 106
- StringBuffer, 95, 102
- StringBuilder, 96, 102
- StringTokenizer, 98–101, 331
- SwingUtilities, 479
- System.Properties, 84
- TeePrintStream, 349
- TextAreaOutputStream, 500
- TextAreaWriter, 500
- Thread, 496, 564, 724, 727
- Throwable, 290
- TimeUnit, 742
- Toolkit, 513
- Transformer, 689
- Transport, 587
- TreeSet, 233, 249

- URI, 458
- URLClassLoader, 778, 844
- URLConnection, 460
- UseLocales, 546
- UserDBJDBC, 638
- Vector, 224, 233, 262
- WindowAdapter, 490
- ZipEntry, 368
- ZipFile, 368
- klasy
 - abstrakcyjne, 218, 277
 - anonimowe, 486
 - bazowe, 278
 - pakietu java.io, 318
 - potomne, 265, 277
 - synchronizowane, 308
 - wartościowe, 268
 - wewnętrzne, 274, 486
 - wytwórcze, 461
- klauzula catch, 842
- klient
 - pogawędek, 468
 - UDP, 453
 - usługi internetowej REST, 459
 - usługi internetowej SOAP, 461
- kodowanie
 - UTF-8, 548
 - UUENCODE, 599
- kody
 - formatujące, 327
 - formatujące dat i godzin, 330
 - rodzime, 820
 - zależne od systemu, 84
 - źródłowe, 75
- kolejka, 751
- kolekcje, 221, 230
 - ogólne, 225
 - równoległe, 299, 308
- kolorowanie syntaktyczne, 33
- kolory predefiniowane, 509
- komentarz, 50, 243, 264
- kompilacja, 32
 - przyrostowa, 35
 - skryptu, 816
 - warunkowa, 61
- kompilator, 700, 820
- kompilator javac, 32
- komponenty
 - EJB3, 555
 - graficzne, 409
 - JavaBeans, 704
 - spakowane, 708
 - Swing, 511
- kompresja
 - plików, 368, 373
 - znaków tabulacji, 111
- komunikacja między wątkami, 745
- komunikat class not found, 83
- komunikaty, 547
 - o błędach, 572
 - o odrzuconych metodach, 59, 60
 - o wyjątkach, 74, 633
 - testowe, 64
- koniec
 - pliku, 331
 - wiersza, 352
- konkatenacja łańcuchów, 102
- konsola, 320
- konstruktor
 - Date, 60
 - kopiujący, 264
 - wyjątku, 290
- konstruktory
 - klasy Exception, 290
 - klasy Thread, 727
- kontrola wielkości
 - liter, 116
 - znaków, 150
- konwersja
 - daty i czasu, 209, 211, 216
 - liczb, 171
 - obiektów, 672, 675
 - między systemami liczbowymi, 181
 - znaków Unicode, 108
- kończenie programu, 489
- kopiowanie plików, 341
- kreator tworzenia aplikacji, 36

L

- lambda, 301, 302
- leksem, 100
- liczba wątków, 563
- liczby, 165
 - bardzo duże, 197
 - całkowite, 172
 - losowe, 167, 189
 - w łańcuchu, 101
 - zespółone, 195
 - zmiennoprzecinkowe, 173, 175, 177
- LIFO, Last In, First Out, 110, 256
- limit czasu wykonania, 734
- lista
 - argumentów, 837
 - zawartości archiwum, 368

- listy połączone, 217, 234
- log4j, 576
- logarytm, 192
- lokalizacja, 535
- loopback, 581

Ł

- łańcuchy, 95
 - dane rozdzielone przecinkami, 120
 - daty, 212
 - dzielenie, 98
 - konwertowanie znaków Unicode, 108
 - łączenie, 102
 - odczytywanie fragmentów, 97
 - odwracanie kolejności słów, 110
 - porównywanie znaków, 119
 - przetwarzane po jednej literze, 104
 - sprawdzanie liczb, 168
 - usuwanie odstępów, 117
 - usuwanie odstępów z końca, 119
 - usuwanie znaków, 544
 - wielkość liter, 116
 - wyrównywanie, 105
 - zapisywanie, 240
- łączenie języków, 817

M

- Mac OS X, 703
 - pakiet Swing, 519
- macierz, 193
- magazyn, 604
- magazyn asocjacyjny, 221
- maszyna wirtualna, 301
- MDI, Multiple Document Interface, 618
- mechanizm
 - AOP, 265
 - asercji, 63
 - blokad, 741
 - CDI, 293
 - dołączania plików, 440
 - enum, 284
 - EventLog, 573
 - kompilacji warunkowej, 62
 - ładowania klasy, 778
 - odzyskiwania pamięci, 782
 - rejestracji komunikatów, 578
 - skryptowy, 812
 - sprawdzania typów, 219
 - synchronizacji, 723
 - testowy TestRunner, 68
 - znaczący, 407

- menedżer układu, 481, 528
 - BorderLayout, 481
 - BoxLayout, 481
 - CardLayout, 481
 - FlowLayout, 481
 - GridBagLayout, 481
 - GridLayout, 481
- menu wielojęzyczne, 539
- metaadnotacje, 792
- metadane, 700
- metadane JDBC, 646, 648
- metaznaki, 133, 153
- metoda, 300
 - accept(), 398, 556, 558, 565
 - actionPerformed(), 300
 - add(), 477, 736
 - addActionListener(), 488
 - addChoosableFileFilter(), 506
 - addLayoutComponent(), 529
 - addShutdownHook(), 273
 - advPlain(), 122
 - advQuoted(), 122
 - append(), 109
 - appendTail(), 146
 - assertThat, 68
 - binarySearch(), 250
 - boolean accept(), 508
 - boolean matches(), 140
 - characters(), 679
 - charAt(), 95, 104, 109
 - Class.forName(), 628
 - clone(), 264, 272
 - compareTo(), 249
 - compareToIgnoreCase(), 314
 - compute(), 310, 753
 - computeArea(), 279, 280
 - connect(), 453
 - ConstructDialog(), 509
 - consume(), 747
 - contains(), 250
 - containsKey(), 250
 - containsValue(), 250
 - converse(), 447
 - copyFile(), 341
 - createFont(), 417
 - createNewFile(), 390
 - createStatement(), 629, 634
 - createTempFile(), 394, 395
 - currentTimeMillis(), 779, 782
 - DateTimeFormatter.ofPattern(), 209
 - defineClass(), 779
 - delete(), 392
 - deleteOnExit(), 379, 394

deriveFont(), 417
 doGet(), 739
 Double.isNaN():, 174
 drawImage(), 419
 drawLine(), 408
 drawString(), 410
 DriverManager.getConnection(), 628–631
 end(), 144
 endClass(), 796
 endElement(), 679
 entrySet(), 239
 equals(), 116, 176, 246, 267
 equalsIgnoreCase(), 116
 eval(), 816
 exec(), 804, 805
 executeQuery(), 629, 637
 executeUpdate(), 644
 fetch(), 62
 finalize(), 264, 273
 find(), 143
 Float.isNaN(), 174
 Font.createFont(), 417
 foo, 172
 forEach(), 845
 fork(), 753
 format(), 106, 127
 forName(), 775
 get(), 226
 getAvailableLocales(), 538
 getClass(), 371, 770
 getClassLoader(), 371
 getColor(), 408
 getConstructors(), 771
 getContentPane(), 414, 477
 getenv(), 80
 getFields(), 784
 getFont(), 408
 getGlassPane(), 414
 getGraphics(), 409, 423
 getHeight(), 420
 getHostAddress(), 443
 getHostName(), 443
 getImage(), 419, 421
 getImageLoadStatus(), 423
 getInetAddress(), 444
 getInputStream(), 368, 376, 446, 558, 808
 getInstance(), 288
 getLineNumber(), 356
 getLocalHost(), 444
 getMediaPlayer(), 429
 getMessages(), 604
 getMetaData(), 647
 getMethod(), 773
 getMethods(), 771, 784
 getName(), 368
 getopt(), 88
 getOutputStream(), 446, 450, 558
 getPortIdentifiers(), 376
 getResource(), 371, 716
 getResourceAsStream(), 371
 getResultSetMetaData(), 635
 getServerSocket(), 570
 getSize(), 420
 getStringBound(), 423
 getWarnings(), 633
 getWidth(), 420
 group(), 144
 groupCount(), 144
 hashCode(), 266, 267, 271
 hasMoreTokens(), 98
 indexOf(), 250
 InetAddress.getByName(), 558
 int getPriority(), 728
 Integer.parseInt(), 181
 invoke(), 773
 invokeAndWait(), 479
 invokeLater(), 479
 isNan(), 173
 Iterable.forEach(), 230, 255
 join(), 727, 734
 keySet(), 226
 launch(), 523
 layoutContainer(), 529
 list(), 397
 listFiles(), 397
 listFolder(), 607
 listRoots(), 399, 400
 loadClass(), 778
 Locale.getInstance(), 545
 Locale.setDefault(), 546, 547
 log(), 192
 Logger.getLogger(), 576, 579
 long getFilePointer(), 360
 lookingAt(), 143
 main(), 88, 826
 matches(), 144
 matcher(), 140
 matcher.matches(), 156
 matches(), 143
 minimumLayoutSize(), 529
 mkButton(), 540
 mkDialog(), 540
 mkdir(), 400
 mkdirs(), 400
 mkMenu(), 540
 myString.toCharArray(), 104

metoda
 newInstance(), 775
 nextDouble(), 191
 nextGaussian(), 191
 nextInt(), 335
 nextToken(), 98
 notify(), 728, 745
 metoda notifyAll(), 728, 745
 metoda open(), 374, 376
 paint(), 407, 409, 421
 paintComponent(), 409
 parse(), 121, 212
 parseArguments, 88
 parseMedia(), 429
 parseObject(), 187
 Pattern.compile(), 142, 151
 pattern.matcher(), 142
 Period.between(), 213
 play(), 429
 poll(), 563
 pop(), 227
 preferredLayoutSize(), 529
 prepareImage(), 422
 prepareMedia(), 429
 prepareStatement(), 629
 print(), 63
 printf, 327, 839
 println(), 188, 353
 printStackTrace(), 74
 push(), 227
 put(), 226
 random(), 189
 read(), 321
 readerToString(), 347
 readLine(), 97, 153, 353
 readLock(), 742
 readPassword(), 324
 readValue(), 664
 remove(), 254
 removeLayoutComponent(), 529
 renameTo(), 391
 replaceAll(), 146
 ResourceBundle.getBundle(), 537
 reverse(), 111
 run(), 725, 726
 runserver(), 758
 Runtime.exec(), 803
 scanf(), 335
 search(), 250, 306
 select(), 563
 setAccessible(), 774
 setBounds(), 529
 setColor(), 408, 415
 setCommand(), 643
 setDefaultCloseOperation(), 489–492
 setDefaultUncaughtExceptionHandler(), 496
 setErr(), 347
 setFont(), 408
 setIn(), 347
 setLogStream(), 633
 setMinimumIntegerDigits(), 179
 setOut(), 347
 setPaint(), 415
 setSerialPortParams(), 376
 setToolTipText(), 511
 showDialog(), 509
 showMessageDialog(), 494, 540
 singletonList(), 289
 singletonMap(), 289
 singletonSet(), 289
 sort(), 244
 start(), 144
 startClass(), 796
 startElement(), 679
 statusAll(), 423
 stop(), 727, 732
 String getDescription(), 508
 String.substring(), 145
 StringBuilder.length(), 104
 substring(), 97, 105, 117
 System.console(), 323
 System.getenv(), 80
 System.getProperties(), 81
 System.setOut(), 348
 toArray(), 252
 toCharArray(), 104
 toInstant().getEpochSecond(), 396
 toLowerCase(), 116
 toString(), 96, 102, 266
 toUpperCase(), 116
 Transport.send(), 592
 trim(), 119
 UIManager.setLookAndFeel(), 515
 updateRow(), 643
 valueOf(), 172
 void setPriority(int), 728
 wait(), 728, 745
 windowClosing(), 490
 writeLock(), 742
 writeValue(), 664
 XMLReaderFactory.createXMLReader(), 680
 metodologia
 Extreme Programming, 67
 JUnit, 67
 metody, 771
 abstrakcyjne, 279, 309
 cyklu życiowego wątku, 727
 do obsługi dat i godzin, 206

- fabryczne, 179
- generujące liczby losowe, 191
- instancyjne, 315
- interfejsu `LayoutManager`, 529
- klasy
 - `ArrayList`, 224
 - `ClassLoader`, 371
 - `File`, 388
 - `Lock`, 742
 - `Scanner`, 336
- kończące działanie strumieni, 307
- polimorficzne, 325
- pomocnicze, 539, 540, 579
- prywatne, 774
- przeciążone, 500
- przekazujące, 307
- rejestrujące `SLF4J`, 575
- wytwarzające strumienie, 307
- zwrotne, 276
- metryki czcionki, 411
- MIF, Maker Interchange Format, 691
- MIME, 599
- miniserwer JAX-WS, 841
- mnożenie macierzy, 193
- moduł
 - `CPAN`, 817
 - `Inline`, 819
- modyfikator `volatile`, 821
- modyfikowanie
 - danych, 641, 644
 - strumieni standardowych, 347
 - systemu plików, 388
 - wielkości tablic, 220
- MVC, 291

N

- nagłówki `Since`, 83
- narzędzia
 - JDK, 33
 - JILT, 544
- narzędzie, *Patrz* program
- nazwa
 - pakietu, 693
 - pliku, 390
- nazwy kanoniczne, 390
- NetBeans, 36
- NFS, 555
- niezmienność łańcuchów znaków, 96
- NIO, New IO, 319
- nowe API, 206, 207
- numer
 - portu, 454, 556
 - wersji `UUID`, 366

O

- obiekt
 - `ActionListener`, 484
 - `AudioClip`, 424
 - `Canvas`, 428
 - `Class`, 777
 - `ClassLoader`, 717, 769, 778
 - `Condition`, 745
 - `Connection`, 629, 634
 - `DatagramPacket`, 452, 453
 - `DatagramSocket`, 453
 - `Date`, 451
 - `Dimension`, 513
 - `Document`, 689
 - `DriverManager`, 631
 - `EntityManager`, 623
 - `EntityManagerFactory`, 623
 - `Handler`, 579
 - `HttpServletRequest`, 760
 - `InetAddress`, 444
 - `InputStream`, 446
 - `Iterator`, 743
 - `JSONArray`, 666
 - `Listener`, 484
 - `Logger`, 574, 579
 - `MediaTracker`, 422
 - `OutputStream`, 446
 - `PreparedStatement`, 637
 - `Process`, 808
 - `Properties`, 85
 - `Reader`, 322
 - `ResourceBundle`, 536, 541
 - `ResultSet`, 635, 641
 - `RowSet`, 642
 - `SerialPort`, 374
 - `ServerSocket`, 568
 - `Stream`, 322
 - `String`, 322
 - `System.err`, 325
 - `System.out`, 325
 - `System.Properties`, 81
 - `TexturedPaint`, 415
 - `URL`, 372
 - `WatchService`, 403
- obiektowy model dokumentu, DOM, 681
- obiekty opakowujące, 165
- obliczanie
 - funkcji trygonometrycznych, 192
 - logarytmów, 192
 - różnic pomiędzy datami, 213
- obraz dysku, 712
- obrazy, 419, 423

- obrazy ruchome, 426
- obsługa
 - animacji, 728
 - błędów sieciowych, 445
 - czcionek, 417
 - dat i godzin, 206, 207, 208
 - drukowania, 430
 - dużych liczb, 197
 - formatu JSON, 663, 667
 - interfejsu użytkownika, 302
 - kolekcji, 224, 308
 - komunikacji, 375
 - obrazów, 423
 - operacji wejścia-wyjścia, 319
 - plików dźwiękowych, 424
 - połączeń sieciowych, 443
 - portów, 374
 - protokołu HTTP, 567
 - protokołu UDP, 453
 - typów liczbowych, 167
 - wielu klientów, 563
 - wstawek, 316
 - wyjątków, 563
 - wyrażeń lambda, 315
 - wyrażeń regularnych, 131, 140
 - żądania, 760
- ODBC, Open DataBase Connectivity, 622
- odbiorca zdarzeń, 484
- odczytywanie
 - danych, 351
 - binarnych, 358, 448
 - serializowanych, 450
 - tekstowych, 446
 - informacji, 320
 - obiektów, 363
 - obrazów, 423
 - poczty elektronicznej, 603
 - skompresowanych plików, 373
 - z konsoli, 323
 - z okna terminala, 323
 - zawartości pliku, 347
 - znaków, 322
- odejmowanie dat, 214
- odmowa usługi, 574
- odnajdywanie
 - elementów XML, 684
 - metadanych JDBC, 646
 - obiektu, 250
 - plików, 371
 - tekstu, 143
 - usług, 444
 - znaków, 153
- odrzuć separatorów, 100
- odtworzenie pliku dźwiękowego, 424
- odwołania do funkcji, 311
- odwołanie typu Runnable, 312
- odwracanie kolejności słów, 110
- odwzorowywanie, 238
- odzyskiwanie pamięci, 782
- ograniczenia
 - czasowe, 734
 - DTD, 686
- okna dialogowe, 494, 541
- okno
 - Eclipse Application Bundle Export, 713
 - New Project, 40
 - Run Configuration, 81
 - wyboru czcionki, 527
- określanie
 - bieżącej daty, 208
 - systemu operacyjnego, 85
- opakowywanie adnotacji, 845
- opcja
 - jar, 703
 - Język i region, 537
 - Ustawienia regionalne, 536
- opcje
 - do refaktoryzacji, 39
 - programu grep, 159
- OpenJDK, 830
- operacje wejścia-wyjścia, 317
- operator
 - diamantowy, 228
 - konkatenacji, 102
 - równości, 267
 - trójargumentowy, 188
- optymalizacja działania równoległego, 753
- ostrzeżenie, 60
- otwarte oprogramowanie, 76
- otwieranie pliku, 340

P

- pakiet, 693
 - ADT, 36
 - aplikacji, 712
 - AWT, 477
 - com.darwinsys, 46
 - com.darwinsys.swingui, 540
 - CygWin, 132
 - Developer Tools, 41
 - Hamcrest Matchers, 68
 - Jackson, 664
 - java.io, 97, 318
 - java.lang.reflect, 771
 - java.sql, 621

- java.text, 178
- java.time, 207
- java.time.chrono, 207
- java.time.format, 207
- java.time.temporal, 207
- java.time.zone, 207
- java.util, 218
- java.util.concurrent, 751, 765
- java.util.concurrent.locks, 741
- java.util.logging, 579
- java.util.regex, 140, 141
- JavaFX, 522
- javax.imageio, 423
- javax.mail, 587
- javax.script, 811, 813
- JILT, 544, 545
- Joda-Time, 205
- org.json, 665
- Swing, 475, 519
- SWT, 475
- UDP, 452
- palindrom, 202
- panel GlassPane, 477
- parametr T, 227
- parametry komunikacji, 376
- pętla
 - for, 230
 - foreach, 230, 835
 - while, 230, 232
 - zwrotna, 581
- plik
 - Address.java, 624
 - AllClasses.java, 275
 - AnnotationOverrideDemo.java, 701
 - AppletAdapter.java, 800
 - AppletViewer.java, 798
 - Appt.java, 246
 - Array.java, 261
 - Array1.java, 219
 - ArrayHunt.java, 251
 - ArrayIterator.java, 253
 - ArrayIteratorDemo.java, 255
 - ArrayTwoDObjects.java, 260
 - AudioPlay.java, 425
 - AutoSave.java, 756
 - BadNewLine.java, 354
 - BigNumCalc.java, 198
 - BigNums.java, 198
 - BookRank.java, 157
 - Buggy.java, 65
 - build.gradle, 58
 - BuildingManagement.java, 278
 - BusCard.java, 552
 - ButtonDemo.java, 484
 - BuzzInServlet.java, 738
 - CachedRowSetDemo.java, 643
 - Calc.java, 463
 - CalcScriptEngine.java, 813
 - CalcScriptEngineFactory.java, 814
 - CallTrack.java, 248
 - CameraSearchPredicate.java, 306
 - Case.java, 116
 - CastNeeded.java, 170
 - CatStdin.java, 322
 - CDIMain.java, 293
 - ChatClient.java, 469
 - ChatServer.java, 582
 - CheckForSwing.java, 83
 - CheckOpenMailRelayGui.java, 503
 - CheckSum.java, 105
 - ChessMoveException.java, 290
 - ClassesInPackage.java, 786
 - comm.jar, 375
 - CommPortSimple.java, 376
 - Complex.java, 196
 - Configuration.java, 673
 - Connect.java, 633
 - ConnectFriendly.java, 445
 - ConnectSimple.java, 442
 - ConsoleRead.java, 324
 - ConsoleViewer.java, 293, 294
 - ContLineReader.java, 356
 - ControllerTightlyCoupled.java, 292
 - Cookies.java, 777
 - Cooklet.java, 776
 - Creat.java, 391
 - CrossRef.java, 794
 - CrossRefXML.java, 796
 - CSVImport.java, 122
 - CSVRE.java, 124
 - CSVSimple.java, 121
 - CurrentDateTime.java, 208
 - DatabaseMetaDemo.java, 648
 - DateAdd.java, 215
 - DateConversions.java, 212
 - DateDiff.java, 214
 - DateFormatter.java, 211
 - DateParse.java, 212, 213
 - DaytimeObject.java, 451
 - DaytimeObjectServer.java, 562
 - DaytimeServer.java, 561
 - DaytimeText.java, 447
 - DaytimeUDP.java, 453
 - DefeatPrivacy.java, 775
 - Delete.java, 392
 - DeTab.java, 114

plik

DirRoots.java, 399
DocWriteDOM.java, 689, 690
DrawStringDemo.java, 411
DropShadow.java, 413
EchoClientOneLine.java, 447
EchoServer.java, 559
EchoServerThreaded.java, 564
EnTab.java, 112
EntryLayout.java, 531
EntryLayoutTest.java, 530
EnumList.java, 287
EqualsDemo.java, 269
EqualsDemoTest.java, 270
err.log, 349
ErrorUtil.java, 497
ExecAndPrint.java, 809
ExecDemoLs.java, 808
ExecDemoNS.java, 805
ExecDemoSimple.java, 804
ExecDemoWait.java, 810
FileIO.java, 341
FileSaver.java, 380
FileStatus.java, 388
FileWatchServiceDemo.java, 403
Find.java, 404
FindField.java, 772
FloatCmp.java, 176
Fmt.java, 125
FNFilter.java, 398
FNFilterL.java, 398
FontChooser.java, 524
FormatPluralsChoice.java, 189
ForEachChar.java, 104
FormatPlurals.java, 188
FormatterDates.java, 330
FractMult.java, 173
GetAndInvokeMethod.java, 773
GetImage.java, 420
GetMark.java, 119
GetNumber.java, 168
GetOpt.java, 90
GetOptSimple.java, 89
GoodNewLine.java, 354
Grapher.java, 436
Grep0.java, 150
GrepNIO.java, 148
Handler.java, 760
HashMapDemo.java, 239
HashMapWithRemoves.java, 240
HelloApplet.java, 797
HelloJni.c, 823
HelloJni.java, 821

Heron.java, 174
hibernate.cfg.xml, 627
HibernateSimple.java, 627
Httpd.java, 758
HttpdConcurrent.java, 765
I18N.java, 540
IndentContLineReader.java, 357
IndentContLineReaderTest.java, 355
InetAddrDemo.java, 443
InfNan.java, 174
IntegerBinOctHexEtc.java, 181
Intr.java, 734
IterableForEach.java, 231
IteratorDemo.java, 234
JabaDex.jnlp, 716
JavadocDemo.java, 697
JAXPTTransform.java, 678
JColorChooserDemo.java, 510
JFileChooserDemo.java, 507
JFileFilter.java, 508
JFrameDemo.java, 477
JFrameDemoSafe.java, 479
JFrameFlowLayout.java, 482
JfxVideo.java, 430
JGrep.java, 160
JLabelHTMLDemo.java, 511
JMFPlayer.java, 426
Join.java, 735
JOptionDemo.java, 494, 542
JPASimple.java, 625
JSSEWebServer0.java, 570
KwikLinkChecker.java, 472
LabelText.java, 706
LegacyDates.java, 216
LinkList.java, 236
Listen.java, 556
ListenInside.java, 557
ListMethods.java, 771
ListsOldAndNew.java, 836
LNFSwitcher.java, 517
LoadDriver.java, 630
LocalDateToJsonManually.java, 663
Log4JDemo.java, 577
LogRegExp.java, 155
MacOsUiHints.java, 520
MailClient.java, 612
MailComposeBean.java, 614
MailComposeFrame.java, 618
MailConstants.java, 603
Mailer.java, 595
MailLister.java, 605
MailReaderBean.java, 608
MailtoButton.java, 589

Matrix.java, 194
 Media.java, 285
 MediaFancy.java, 287
 MenuIntl.java, 549
 MessageFormatDemoIntl.java, 549
 MessageNode.java, 607
 Model.java, 294
 MutableInteger.java, 282
 MyJavaP.java, 784
 MyStack.java, 227
 MyStackDemo.java, 229
 MyToyAnnotation.class, 790
 NetworkInterfaceDemo.java, 581
 NLMatch.java, 154
 NumFormat2.java, 179
 NumSeries.java, 182
 PaintDemo.java, 408
 Palindrome.java, 202
 PathsFilesDemo.java, 402
 people.dtd, 688
 people.xsd, 686
 people.xml, 677
 persistence.xml, 623, 626
 PlotDriver.java, 296
 Plotter.java, 295
 PlotterAWT.java, 434
 pom.xml, 53
 PrefsDemo.java, 241
 PrintPostScript.java, 433
 ProcessBuilderDemo.java, 807
 ProdCons1.java, 746
 ProdCons15.java, 751
 ProdCons2.java, 748
 PropsCompanies.java, 243
 PSFormatter.java, 383
 random.dat, 361
 RandomInt.java, 190
 RDateClient.java, 449
 ReaderIter.java, 147, 148
 ReadersWriterDemo.java, 743
 ReadGZIP.java, 374
 ReadOnly.java, 396
 ReadPassword.java, 324
 ReadRandom.java, 360
 ReadStdinInt.java, 322
 ReadWriteImage.java, 423
 ReadWriteJackson.java, 664
 RecursiveActionDemo.java, 754
 REmatch.java, 144
 REmatchTwoFields.java, 145
 RemCat.java, 455
 Rename.java, 392
 RenjinScripting.java, 812
 RESimple.java, 140
 RestClientFreeGeoIp.java, 460
 REsubstr.java, 145
 ResultsDecorator.java, 651
 ResultsDecoratorHTML.java, 647
 ResultSetUpdate.java, 642
 RomanNumberFormat.java, 185
 RomanYear.java, 184
 Round.java, 177
 RunOnEdt.java, 480
 SAXLister.java, 679
 ScanStringTok.java, 332
 ScriptEnginesDemo.java, 812
 Sender.java, 593
 SendMime.java, 600
 SerialDemoAbstractBase.java, 364
 SerialDemoXML.java, 675
 SerializableUser.java, 366
 SetLocale.java, 547
 ShapeDriver.java, 280
 SimpleStack.java, 257
 SimpleStreamDemo.java, 307
 Singleton.java, 288
 Slf4jDemo.java, 574
 SoftwareParseJackson.java, 665
 SoftwareParseOrgJson.java, 666
 SortArray.java, 245
 Soundex.java, 128
 SoundexSimple.java, 127
 SpinnerDemo.java, 505
 spring/Controller.java, 292
 Sprite.java, 729
 SQLRunner.java, 653
 StopBoolean.java, 732
 StopClose.java, 733
 StringAlign.java, 106
 StringAlignSimple.java, 106
 StringConvenience.java, 142
 StringParse.java, 282
 StringPrintA.java, 781
 StringPrintB.java, 781
 StringReverse.java, 111
 StringToDouble.java, 168
 StrTokDemo.java, 99
 SubstringComparator.java, 245
 Swinging.pl, 817
 SysDep.java, 86
 TabPaneDemo.java, 483
 Tabs.java, 115
 TeePrintStream.java, 350
 Telnet.java, 467
 TempConverter.java, 200
 TempFiles.java, 394

- plik
 - TextAreaOutputStream.java, 500
 - TextWriter.java, 499
 - TextToJDBC.java, 645
 - TexturedText.java, 416
 - ThreadBasedCatcher.java, 496
 - ThreadsDemo1.java, 724
 - TiledImageComponent.java, 420
 - Time.java, 782
 - ToArray.java, 252
 - ToStringWith.java, 267
 - ToStringWithout.java, 266
 - ToyStack.java, 256
 - Trig.java, 192
 - UnicodeChars.java, 109
 - UnZip.java, 369
 - URIDemo.java, 459
 - UseLocales.java, 546
 - UserDBJDBC.java, 638
 - UserQuery.java, 636
 - UtilGUI.java, 513
 - VlcjVideo.java, 429
 - WebServer0.java, 568
 - WindowCloser.java, 492
 - WindowDemo, 491
 - WriteBinary.java, 359
 - WriteOrgJson.java, 666
 - WriteRandom.c, 361
 - XmlForm.java, 691
 - XParse.java, 682
 - XPathDemo.java, 685
 - XTW.java, 683
- pliki
 - .bak, 380
 - .class, 816
 - .gz, 373
 - .h, 822
 - .pyc, 816
 - .stub, 709
 - .tar, 373
 - .tgz, 373
 - .xsd, 674
 - budowy, 824
 - dmg, 712
 - DTD, 688
 - dziennika, 155, 348
 - dźwiękowe, 424
 - filtrowanie, 398
 - JAR, 43, 87, 701, 709, 719
 - JNLP, 716
 - konfiguracyjne, 56
 - kopii bezpieczeństwa, 380
 - makefile, 51
 - multimedialne, 428
 - nagłówkowe, 821
 - POM, 55
 - skompresowane, 368, 373
 - SVG, 670
 - tekstowe, 382
 - tworzenie, 390
 - tymczasowe, 379, 394
 - usuwanie, 392
 - WAR, 710
 - właściwości, 548, 577
 - wsadowe, 44
 - WSDL, 461
 - XML, 670
 - zmiana atrybutów, 395
 - zmiana nazwy, 391
- pobieranie
 - deskryptora klasy, 770
 - catalogów głównych, 399
 - przykładów, 44
 - ułamek z liczby, 172
 - wyników, 634
- poczta elektroniczna, 587
- podpis cyfrowy, 718
- podpisywanie plików JAR, 718, 719
- podstawa logarytmu, 193
- POJO, Plain Old Java Objects, 664, 705
- pola, 771
- pola prywatne, 774
- polecenia
 - JDBC, 637
 - SQL, 644
- polecenie
 - dir, 397
 - find, 406
 - javac, 32
 - md, 401
- polimorfizm, 279
- połączenie
 - JDBC, 634
 - szyfrowane, 572
 - UDP, 453
 - z bazą danych, 628
 - z relacyjną bazą danych, 631
 - z serwerem, 441
 - z usługą, 461
- pomiar czasu, 780
- porównywanie
 - deskryptorów klas, 270
 - liczb zmiennoprzecinkowych, 175
 - łańcuchów znaków, 119
- port
 - równoległy, 374
 - szeregowy, 374
- potok, 306

powielanie strumienia, 348
powtórzenia, 232
poziom izolacji transakcji, 648
prezentacja
 adnotacji, 791
 programów, 515
priorytet wątku, 728
procedura osadzona, 641
procesor obiektów formatujących, 672
program, *Patrz także* plik
 analiza danych, 156
 analiza dziennika serwera, 155
 Apache Ant, 50–52, 68
 Apache Builder, 51
 Apache Maven, 45, 51, 57, 372
 AppletViewer, 796
 Buggy, 65
 BusCard, 551
 CompRunner, 410
 cpio, 373
 CrossRef, 794
 CSVSimple, 121
 DaytimeBinary, 560
 DeTab, 114
 Endpoint, 463
 ExecDemoNS, 807, 810
 Find, 404
 fonetyczne porównywanie nazwisk, 127
 ftp, 463
 generowanie liczbowych palindromów, 201
 Gradle, 51, 56
 Grapher, 435
 grep, 132, 159
 groff, 127
 gunzip, 373
 gzip, 373
 HttpdConcurrent, 766
 InstallShield, 711
 JabaDex, 714, 715
 jar, 702
 jarsigner, 719
 javac, 700
 Javadoc, 696
 javap, 784, 790
 Jenkins, 70
 jdb, 64
 JMFPlayer, 426
 JVisualVM, 780
 klient pogawędek internetowych, 468
 klient TFTP, 454
 klient usługi Telnet, 466
 MailClient, 611
 MailReaderBean, 608
 MenuIntl, 549
 narzędzie do formatowania, 125
 nroff, 127
 pełna wersja grep, 159
 Plotter, 294
 PlotterAWT, 434
 porównanie szybkości działania, 261
 Mac OS Jar Bundler, 704
 make, 51, 53
 producent-konsument, 750
 REDemo, 137, 139
 roff, 127
 runoff, 127
 serialver, 366
 serwer pogawędek, 582
 sort, 88
 sprawdzanie odnośników HTTP, 472
 SQLRunner, 650
 syslog, 573
 tar, 373
 Telnet, 564
 TempConverter, 200
 troff, 127
 VLC, 428
 wget, 463
 who, 349
 własny menedżer układu, 528
 wsimport, 461–464
 wybieranie czcionek, 524
 XDoclet, 700
 xml2mif, 691
 XParse, 683
 yacc, 338
 zmiana formatu tekstu, 382
programowanie
 aspektowe, AOP, 265, 316
 funkcyjne, 299, 301
 rozproszone, 301
programy
 instalacyjne, 710, 711
 JILT, 545
 komunikacyjne, 378
 Oracle, 829
 poligraficzne, 794
 uruchomieniowe, 64, 66
 wielojęzyczne, 535, 536, 554
 zewnętrzne, 804, 810
projektowanie układu okna, 480
protokół
 HTTP, 441, 567
 HTTPS, 572
 IMAP, 604
 JNLP, 716

- protokół
 - POP3, 604
 - SMTP, 592
 - SOAP, 461
 - TCP, 452
 - TCP/IP, 439, 441
 - TFTP, 441, 455
 - UDP, 441, 452, 454
- przechwytywanie
 - wyjątków, 496, 580, 819, 842
 - wyników, 808
- przekazywanie wartości, 281
- przekierowywanie, 347
- przekształcanie danych XML, 676
- przesłanianie metod, 267, 501
- przesyłanie zapytań JDBC, 634
- przetwarzanie
 - leniwe, 300
 - liczb całkowitych, 182
 - tekstów, 95
 - współbieżne, 308
- przycisk Zamknij, 489
- przyczyna zasadnicza, root cause, 290
- przypadki zastosowania, 264
- pseudourządzenie, 85

Q

- QNX, 847

R

- RDBMS, 621
- refaktoryzacja, 35
- rejestracja
 - komunikatów, 63, 578
 - operacji sieciowych, 572
 - przez sieć, 574, 576, 579
 - wyjątków, 580
- rekurencja, 202, 363
- relacyjne bazy danych, 621
- repozytorium
 - CVS, 71
 - darwinsys-api, 45
 - Git, 71
 - javasrc, 45, 47
 - Subversion, 71
- REST, 459
- RMI, 471
- rodzaje komponentów, 705
- router, 557
- rozkład prawdopodobieństwa, 191

- rozszerzające interfejsy programistyczne, 87
- rozszerzalny język znaczników, 669
- równość kanoniczna, 152
- RPC, Remote Procedure Call, 471, 555
- rysowanie cienia, 413
- rzutowanie typów, 170, 219, 227, 415

S

- SAX, Simple API for XML, 671, 681
- schematy XML, 686
- separator nazw plików, 85
- serializowanie obiektów, 166, 363, 675
- serwer, 556
 - ciągłej integracji, 49, 70
 - Daytime, 467
 - działający wielowątkowo, 765
 - HTTP, 568
 - pocztowy, 602
 - pogawędek, 582
 - protokołu TFTP, 458
 - SMTP, 466, 592
 - wielowątkowy, 758
- serwis
 - GitHub, 45
 - Maven Central, 46
- serwlet, 366, 555, 737
- serwlet BuzzInServlet, 741
- serwlety spakowane, 709
- sieciowa kolejność zapisu bajtów, 361
- składnia wyrażeń regularnych, 132–136
- skrypt, 44, 811
- skrypt CGI, 592
- SLF4J, 573, 574
- słowa kluczowe w Javadoc, 697
- słownik, 221
- słowo kluczowe
 - ADDRESS, 339
 - default, 300
 - we enum, 285, 790
 - inline, 63
 - native, 821
 - synchronized, 735
 - transient, 363
- SOAP, 461
- sortowanie, 244, 248
- sprawdzanie
 - dostępności klasy, 83
 - odnośników HTTP, 472
 - występowania wzorca, 140
- Spring Framework, 77, 291, 705
- Spring MVC, 77

- stała
 - DO_NOTHING_ON_CLOSE, 489
 - Double.MAX_VALUE, 197
 - INFINITY, 173
 - Long.MAX_VALUE, 197
 - POSITIVE_INFINITY, 174
 - String.CASE_INSENSITIVE_ORDER, 245
- stałe typów wyliczeniowych, 285
- standard
 - IEEE-754, 176
 - ISO 8601, 205
 - JDBC, 841
 - MIME, 599
- standardowy strumień błędów, 633
- status obiektu MediaTracker, 422
- sterownik JDBC, 630, 632
- stos, 110, 256
- stosowanie
 - adnotacji, 787, 792
 - cudzysłówów w XML-u, 670
 - czcionek aplikacji, 418
 - domknięć, 301
 - JPA, 623
 - kolekcji, 225
 - metod i pól, 771
 - procedur osadzonych, 641
 - programu jar, 701
 - przygotowanych poleceń, 637
 - ustawień lokalnych, 545
 - wątków, 721
 - wyrażeń lambda, 301
 - zapytań JDBC, 629
- strażnik, 575
- struktura
 - katalogów, 710
 - pakietów, 694
 - plików, 712
- strukturalizacja danych
 - listy połączone, 234
 - tablice, 218
- struktury
 - danych, 221, 234
 - gramatyczne, 338
 - wielowymiarowe, 259
- strumień, Streams, 299, 307, 319
 - błędów, 325
 - obiektywne, 363
 - równoległe, 308, 309
 - w wejściowe, 320
 - w wyjściowe, 325
- strumień ObjectOutputStream, 562
- suma kontrolna, 105
- SWT, Standard Widget Toolkit, 475
- synchronizacja
 - metody, 736
 - wątków, 735, 741, 747
- system
 - Javadoc, 696
 - kontroli wersji, 49
 - liczbowy, 181
 - operacyjny, 84, 825
 - Android, 847
 - BlackBerry OS, 847
 - Mac OS X, 519, 703
 - QNX, 847
 - plików, 317
 - NFS, 555
- szkielet
 - Fork/Join, 753
 - kolekcji, 221
 - wstrzykiwania zależności, 291
- szyfrowanie SSL, 570

Ś

ścieżki dostępu, 85

T

- tablica, 218, 252
 - args, 88
 - mieszająca, 221
 - wielowymiarowa, 259
- TDD, Test Driven Development, 67
- technologia
 - CORBA, 803
 - EJB2, 706
 - Java Web Start, 714
 - JavaServer Faces, 737
 - JDBC, 621, 622, 629
 - JNI, 821, 826
 - ODBC, 622
 - RMI, 366, 803
 - RPC, 555
 - XSLT, 672
- tekst, 95, *Patrz także* łańcuchy
 - formatowanie, 125
 - odnajdywanie, 143
 - wcięcia, 117
 - wyświetlanie, 410, 411, 415, 417
 - z cieniem, 413
 - zastępowanie, 146
 - znaki niedrukowalne, 118
- Telnet, 466, 564

- testowanie
 - jednostkowe, 58, 66, 69, 500
 - komponentów graficznych, 409
 - programów, 35, 458
 - warunkowe, 61
- transmisja datagramów, 452
- tryb
 - CMYK, 510
 - HSL, 509
 - HSV, 509
 - RGB, 509
 - Swatches, 509
- tworzenie
 - animatorów, 728
 - archiwów, 702
 - dokumentacji, 264
 - dokumentacji klas, 696
 - gniazda, 440
 - grafiki dwuwymiarowej, 415
 - GUI, 522
 - interfejsów funkcyjnych, 309
 - interfejsu, 302
 - iteratora, 253
 - kart, 483
 - katalogów, 400
 - klas, 40, 704, 778
 - klas potomnych, 265
 - listy ustawień lokalnych, 538
 - listy zawartości katalogu, 397
 - mechanizmu skryptowego, 813
 - menu, 539
 - metod pomocniczych, 539
 - metod zwrotnych, 276
 - obiektów kolekcji, 225
 - okien dialogowych, 541
 - pakietów, 693
 - plików tymczasowych, 394
 - pliku, 390
 - pliku archiwum, 709
 - potoków, 347
 - procedur osadzonych, 641
 - programów wielojęzycznych, 535, 554
 - przycisków, 484
 - serwera, 556
 - usługi, 462
 - ustawień poczty elektronicznej, 602
 - wątków, 724, 725, 726
 - wcięć, 117
 - wiązki zasobów, 543
 - wstawek, 315
- typ
 - MIME, 812
 - T, 753
 - wyliczeniowy EnumList, 287

- typy
 - danych JDBC, 636
 - danych SQL, 636
 - Javy, 822
 - JNI, 822
 - liczbowe, 165
 - ogólne, generics, 223, 227, 837
 - referencyjne, 311
 - sterowników JDBC, 632
 - wyliczeniowe, 285, 836, 843, 284

U

- układ okna, 480
- Unicode, 108, 319, 326
- unikanie powtórzeń, 232
- upraszczanie
 - programu, 750
 - serwerów, 765
 - synchronizacji, 741
- URI, Uniform Resource Identifier, 458
- URL, Uniform Resource Locator, 458
- URN, Uniform Resource Name, 458
- uruchamianie
 - GUI, 478
 - programu, 32, 723
 - programu zewnętrznego, 804
- urządzenia peryferyjne, 375
- usługa
 - FreeGeoIP, 460
 - freegeoip.net, 460
 - Telnet, 466
 - WatchService, 402
- usługi internetowe, 441
 - REST, 441, 459
 - SOAP, 441, 461
- ustawienia
 - lokalne, 537, 545
 - pocztowe, 602
- usuwanie
 - łańcuchów, 544
 - plików, 392
 - wcięć, 117
- UTF-16, 108
- uzupełnianie kodu, 35

W

- wartości wyliczeniowe, 284
- wartość
 - mieszająca, 272
 - NaN, 174, 177

- ul>
- wątek przekazywania zdarzeń, EDT, 479
- wątki, 721, 838
- wątki niezależne, 563
- wcięcia, 117
- wersja JDK, 82, 83
- weryfikacja poprawności struktury, 686
- węzeł, node, 681
- video, 407, 426
- wielkość liter, 150
- wielokrotne wykorzystywanie poleceń, 637
- wieloprocesowość, 722
- wielowątkowość, 308
- wielowątkowy serwer sieciowy, 758
- wielozadaniowość, 722
- wirtualna maszyna Javy, JVM, 711, 769, 845
- własne
 - interfejsy funkcyjne, 309
 - wyjątki, 290
- własny iterator, 253
- właściwości klasy File, 85
- właściwość, 81
- wnioskowanie typów, 842
- WSDL, Web Service Description Language, 462
- współbieżność, 838
- wstawianie istniejącego kodu, 315
- wstrzykiwanie zależności, 77, 291, 292
- wtyczka, 73, 792
- wtyczka M2Eclipse, 55
- wybieranie
 - czcionki, 524
 - koloru, 509
 - plików, 506
 - ustawień lokalnych, 537
 - wartości, 505
- wydajność oprogramowania, 782
- wyjątek, 74
 - ClassCastException, 229, 366
 - IOException, 321, 387, 445
 - NumberFormatException, 168
 - SecurityException, 387, 393
 - UnsupportedOperationException, 234
- wykonywanie programu, 808
- wyliczanie wartości, 284
- wyliczenia, 230, 233
- wymiana informacji, 691
- wyrażenia lambda, 301, 304, 311, 488, 726
- wyrażenia regularne, 101, 131, 133
 - dopasowywanie znaków, 152
 - odnajdywanie tekstu, 143
 - odnajdywanie znaków, 153
 - składnia, 132–136
 - sprawdzanie dopasowania, 140
 - wyświetlanie dopasowań, 147
 - wyświetlanie wierszy, 149
 - zastępowanie tekstu, 146
- wyrównanie łańcucha znaków, 106
- wysyłanie
 - poczty elektronicznej, 588, 592
 - wiadomości MIME, 599
 - wiadomości przez serwer, 594
- wyszukiwanie, *Patrz także* odnajdywanie
 - interfejsów sieciowych, 581
 - projektu, 57
- wyświetlanie
 - daty i czasu, 209
 - informacji o klasie, 784
 - klas pakietu, 785
 - nazwy liczby, 187
 - obiektów, 266
 - obrazu, 419
 - okna głównego, 512
 - pasujących wierszy, 149
 - tekstu, 327, 410, 415
 - tekstu wyśrodkowanego, 411
 - wszystkich wystąpień wzorca, 147
 - wyników w oknie, 499
 - zawartości katalogu, 84
- wywoływanie
 - kodu, 825
 - kodu rodzimego, 821
 - zdalnych procedur, 555
- wzorzec, pattern, 209
- wzorzec projektowy, 263, 265, 705
 - MVC, 291
 - Singleton, 288
- wzór Herona, 174
- ## X
- XML, Extensible Markup Language, 669–92
 - XP, Extreme Programming, 67
 - XPath, 684, 685
 - XSL, Extensible Style Language, 672
 - XSLT, Extensible Style Language for Transformations, 676
- ## Z
- zabezpieczanie serwera WWW, 570
 - zablokowanie wątku, 733
 - zakończenie wątku, 734
 - zamiana kolekcji, 252
 - zamykanie aplikacji, 489
 - zaokrąglanie wartości zmiennoprzecinkowych, 177

zapis
 danych, 325, 351
 binarnych, 358, 448
 JSON, 664, 665
 serializowanych, 450
 tekstowych, 446
 użytkownika, 379
 w strumieniu, 360
 w tle, 756
 komunikatów, 579
 łańcuchów znaków, 240
 na dysku, 379
 obiektów, 363
 obrazów, 423
 skompresowanych plików, 373
 wyników, 642
 ze strzałką, 304
 zarządzanie
 kodem, 69
 zasobami, 842
 zasady działania equals(), 268
 zasoby, 77, 543
 zasób JNDI, 626
 zastępowanie tekstu, 146
 zatrzymywanie działania wątku, 732
 zbiory, 232
 zdalne sterowanie, 276
 zdarzenia, 484
 zintegrowane środowisko programistyczne, IDE,
 31, 42, 310, 476
 zmiana
 atrybutów pliku, 395
 nazwy pliku, 391
 prezentacji programu, 515
 zmienna
 CLASSPATH, 31, 87, 372, 630
 LANG, 536
 PATH, 80
 zmienne
 atomowe, 723
 środowiskowe, 79
 znacznik
 <!--, 50
 <class>, 796
 znak
 ", 354
 \$, 153
 @, 266, 700, 788
 ^, 131, 153
 końca wiersza, 352
 minusa, 84
 następnego wiersza, 153
 odwrotnego ukośnika, 84
 separatora akapitów, 153
 separatora ścieżek, 84
 separatora wierszy, 153
 strzałki, 304
 ukośnika, 84
 λ, 302
 znaki
 ASCII, 108, 446
 niedrukowalne, 118
 Unicode, 108, 109
 z akcentem, 153
 złożone, 152
 zwijanie kodu, 304
 zwracanie
 informacji, 562
 odpowiedzi, 558

Ż

źródła, 75
 źródło zdarzeń, 484

Ż

żądania HTTP, 441
 żądanie GET, 739

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



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

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

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

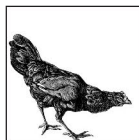
GRUPA WYDAWNICZA



Helion SA

Java. Receptury

Wydanie III



Java to jeden z języków programowania najchętniej wybieranych do tworzenia zaawansowanych systemów informatycznych. Systemy bankowe, aukcyjne oraz inne zaawansowane programy wspomagające codzienną pracę tysięcy ludzi opierają się na Javie i narzędziach z nią związanych. Jeżeli chcesz śmiało wkroczyć w świat tego języka, musisz mieć tę książkę!

Książka należy do cenionej serii „Receptury”, która przedstawia różne zagadnienia w postaci krótkich przepisów. Nie inaczej jest w tym przypadku. Sięgnij po nią i zobacz, jak kompilować, uruchamiać i testować tworzony kod. W kolejnych rozdziałach zaznajomisz się z najlepszymi metodami przetwarzania ciągów znaków oraz nauczysz się korzystać z wyrażeń regularnych i wykonywać operacje na liczbach. Ponadto zdobędziesz dogłębną wiedzę na temat urządzeń wejścia-wyjścia, używania systemu plików, sieci oraz drukarek. To obowiązkowa pozycja na półce każdego programisty Javy!

Dzięki tej książce:

- poznasz język programowania Java
- nauczysz się korzystać z systemu plików oraz sieci
- wykorzystasz wyrażenia regularne
- błyskawicznie rozwiążesz typowe problemy

Najlepsze przepisy dla programistów Javy!

Helion	
30129	numer katalogowy
księgarnia internetowa	
http://helion.pl	
zamówienia telefoniczne	
	0 801 339900
	0 601 339900

Sprawdź najnowsze promocje:
 ① <http://helion.pl/promocje>
 Książki najchętniej czytane:
 ② <http://helion.pl/bestsellery>
 Zamów informacje o nowościach:
 ③ <http://helion.pl/novosci>

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



ISBN 978-83-246-9570-6



9 788324 695706

Informatyka w najlepszym wydaniu

cena 99,00 zł