

ORACLE®



Java

Kompendium programisty

Wydanie IX

Herbert Schildt



Helion 

Oracle
Press™

Tytuł oryginału: Java™ The Complete Reference, Ninth Edition

Tłumaczenie: Piotr Rajca
na podstawie „Java. Kompendium programisty. Wydanie VIII”
w tłumaczeniu Mikołaja Szczepaniaka

ISBN: 978-83-283-0812-1

Original edition copyright © 2014 by McGraw-Hill Education (Publisher).
All rights reserved.

Polish edition copyright © 2015 by HELION S.A.
All rights reserved.

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/javkp9>
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
<ftp://ftp.helion.pl/przyklady/javkp9.zip>

Printed in Poland.

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

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

Spis treści

O autorze	23
Przedmowa	25

CZĘŚĆ I Język Java

1 Historia i ewolucja języka Java	31
Rodowód Javy	31
Narodziny nowoczesnego języka — C	31
Język C++ — następny krok	33
Podwaliny języka Java	33
Powstanie języka Java	33
Powiązanie z językiem C#	35
Jak język Java zmienił internet	35
Aplety Javy	35
Bezpieczeństwo	36
Przenośność	36
Magia języka Java — kod bajtowy	36
Serwlety — Java po stronie serwera	37
Hasła języka Java	38
Prostota	38
Obiektowość	38
Niezawodność	38
Wielowątkowość	39
Neutralność architektury	39
Interpretowalność i wysoka wydajność	39
Rozproszenie	39
Dynamika	40
Ewolucja Javy	40
Java SE 8	42
Kultura innowacji	43

2	Podstawy języka Java	45
	Programowanie obiektowe	45
	Dwa paradygmaty	45
	Abstrakcja	46
	Trzy zasady programowania obiektowego	46
	Pierwszy przykładowy program	50
	Wpisanie kodu programu	50
	Kompilacja programów	51
	Bliższe spojrzenie na pierwszy przykładowy program	51
	Drugi prosty program	53
	Dwie instrukcje sterujące	54
	Instrukcja if	54
	Pętla for	55
	Bloki kodu	56
	Kwestie składniowe	58
	Znaki białe	58
	Identyfikatory	58
	Stałe	58
	Komentarze	58
	Separatory	59
	Słowa kluczowe języka Java	59
	Biblioteki klas Javy	60
3	Typy danych, zmienne i tablice	61
	Java to język ze ścisłą kontrolą typów	61
	Typy proste	61
	Typy całkowitoliczbowe	62
	Typ byte	62
	Typ short	63
	Typ int	63
	Typ long	63
	Typy zmiennoprzecinkowe	63
	Typ float	64
	Typ double	64
	Typ znakowy	64
	Typ logiczny	66
	Bliższe spojrzenie na stałe	66
	Stałe całkowitoliczbowe	66
	Stałe zmiennoprzecinkowe	67
	Stałe logiczne	68
	Stałe znakowe	68
	Stałe łańcuchowe	69
	Zmienne	69
	Deklaracja zmiennej	69
	Inicjalizacja dynamiczna	70
	Zasięg i czas życia zmiennych	70
	Konwersja typów i rzutowanie	72
	Automatyczna konwersja typów	72
	Rzutowanie niezgodnych typów	73
	Automatyczne rozszerzanie typów w wyrażeniach	74
	Zasady rozszerzania typu	74

Tablice	75
Tablice jednowymiarowe	75
Tablice wielowymiarowe	77
Alternatywna składnia deklaracji tablicy	80
Kilka słów o łańcuchach	80
Uwaga dla programistów języka C lub C++ na temat wskaźników	81
4 Operatory	83
Operatory arytmetyczne	83
Podstawowe operatory arytmetyczne	84
Operator reszty z dzielenia	84
Operatory arytmetyczne z przypisaniem	85
Inkrementacja i dekrementacja	86
Operatory bitowe	87
Logiczne operatory bitowe	88
Przesunięcie w lewo	90
Przesunięcie w prawo	91
Przesunięcie w prawo bez znaku	92
Operatory bitowe z przypisaniem	93
Operatory relacji	94
Operatory logiczne	95
Operatory logiczne ze skracaniem	96
Operator przypisania	96
Operator ?	97
Kolejność wykonywania operatorów	97
Stosowanie nawiasów okrągłych	98
5 Instrukcje sterujące	99
Instrukcje wyboru	99
Instrukcja if	99
Instrukcja switch	102
Instrukcje iteracyjne	105
Pętla while	106
Pętla do-while	107
Pętla for	109
Wersja for-each pętli for	112
Pętle zagnieżdżone	116
Instrukcje skoku	116
Instrukcja break	117
Instrukcja continue	120
Instrukcja return	121
6 Wprowadzenie do klas	123
Klasy	123
Ogólna postać klasy	123
Prosta klasa	124
Deklarowanie obiektów	126
Bliższe spojrzenie na operator new	127
Przypisywanie zmiennych referencyjnych do obiektów	127
Wprowadzenie do metod	128
Dodanie metody do klasy Box	129
Zwracanie wartości	130
Dodanie metody przyjmującej parametry	131

Konstruktor	133
Konstruktor sparametryzowany	134
Słowo kluczowe this	135
Ukrywanie zmiennych składowych	135
Mechanizm odzyskiwania pamięci	136
Metoda finalize()	136
Klasa stosu	137
7 Dokładniejsze omówienie metod i klas	139
Przeciążanie metod	139
Przeciążanie konstruktorów	141
Obiekty jako parametry	143
Dokładniejsze omówienie przekazywania argumentów	145
Zwracanie obiektów	146
Rekurencja	147
Wprowadzenie do kontroli dostępu	149
Składowe statyczne	152
Słowo kluczowe final	153
Powtórka z tablic	154
Klasy zagnieżdżone i klasy wewnętrzne	155
Omówienie klasy String	157
Wykorzystanie argumentów wiersza poleceń	159
Zmienna liczba argumentów	160
Przeciążanie metod o zmiennej liczbie argumentów	162
Zmienna liczba argumentów i niejednoznaczności	163
8 Dziedziczenie	165
Podstawy dziedziczenia	165
Dostęp do składowych a dziedziczenie	166
Bardziej praktyczny przykład	167
Zmienna klasy bazowej może zawierać referencję do obiektu podklasy	169
Słowo kluczowe super	170
Wykorzystanie słowa kluczowego super do wywołania konstruktora klasy bazowej	170
Drugie zastosowanie słowa kluczowego super	173
Tworzenie hierarchii wielopoziomowej	173
Kiedy są wykonywane konstruktory?	176
Przesłanie metod	177
Dynamiczne przydzielanie metod	178
Dlaczego warto przesłaniać metody?	180
Zastosowanie przesłaniania metod	180
Klasy abstrakcyjne	181
Słowo kluczowe final i dziedziczenie	184
Słowo kluczowe final zapobiega przesłanianiu	184
Słowo kluczowe final zapobiega dziedziczeniu	184
Klasa Object	185
9 Pakiety i interfejsy	187
Pakiety	187
Definiowanie pakietu	187
Znajdowanie pakietów i ścieżka CLASSPATH	188
Prosty przykład pakietu	189
Ochrona dostępu	189
Przykład dostępu	190

Import pakietów	192
Interfejsy	194
Definiowanie interfejsu	194
Implementacja interfejsu	195
Interfejsy zagnieżdżone	197
Stosowanie interfejsów	198
Zmienne w interfejsach	201
Interfejsy można rozszerzać	202
Metody domyślne	203
Podstawy metod domyślnych	204
Bardziej praktyczny przykład	205
Problemy wielokrotnego dziedziczenia	206
Metody statyczne w interfejsach	207
Ostatnie uwagi dotyczące pakietów i interfejsów	207
10 Obsługa wyjątków	209
Podstawy obsługi wyjątków	209
Typy wyjątków	210
Nieprzechwycone wyjątki	210
Stosowanie instrukcji try i catch	211
Wyświetlenie opisu wyjątku	212
Wiele klauzul catch	213
Zagnieżdżone instrukcje try	214
Instrukcja throw	216
Klauzula throws	217
Słowo kluczowe finally	217
Wyjątki wbudowane w język Java	219
Tworzenie własnej podklasy wyjątków	219
Łańcuch wyjątków	222
Trzy nowe cechy wyjątków	223
Wykorzystanie wyjątków	224
11 Programowanie wielowątkowe	225
Model wątków języka Java	226
Priorytety wątków	227
Synchronizacja	227
Przekazywanie komunikatów	228
Klasa Thread i interfejs Runnable	228
Wątek główny	228
Tworzenie wątku	230
Implementacja interfejsu Runnable	230
Rozszerzanie klasy Thread	232
Wybór odpowiedniego podejścia	232
Tworzenie wielu wątków	233
Stosowanie metod isAlive() i join()	234
Priorytety wątków	236
Synchronizacja	237
Synchronizacja metod	237
Instrukcja synchronized	239
Komunikacja międzywątkowa	240
Zakleszczenie	244
Zawieszanie, wznawianie i zatrzymywanie wątków	245
Uzyskiwanie stanu wątku	247
Korzystanie z wielowątkowości	249

12	Wyliczenia, automatyczne opakowywanie typów prostych i adnotacje (metadane) ...	251
	Typy wyliczeniowe	251
	Podstawy wyliczeń	251
	Metody values() i valueOf()	253
	Wyliczenia Javy jako typy klasowe	254
	Wyliczenia dziedziczą po klasie Enum	256
	Inny przykład wyliczenia	257
	Opakowania typów	258
	Klasa Character	259
	Klasa Boolean	259
	Opakowania typów numerycznych	259
	Automatyczne opakowywanie typów prostych	260
	Automatyczne opakowywanie i metody	261
	Automatyczne opakowywanie i rozpakowywanie w wyrażeniach	262
	Automatyczne opakowywanie typów znakowych i logicznych	263
	Automatyczne opakowywanie pomaga zapobiegać błędom	264
	Słowo ostrzeżenia	264
	Adnotacje (metadane)	265
	Podstawy tworzenia adnotacji	265
	Określanie strategii zachowywania adnotacji	266
	Odczytywanie adnotacji w trakcie działania programu za pomocą refleksji	266
	Interfejs AnnotatedElement	270
	Wartości domyślne	271
	Adnotacje znacznikowe	272
	Adnotacje jednoelementowe	272
	Wbudowane adnotacje	274
	Adnotacje typów	275
	Adnotacje powtarzalne	279
	Ograniczenia	281
13	Wejście-wyjście, applety i inne tematy	283
	Podstawowa obsługa wejścia i wyjścia	283
	Strumienie	284
	Strumienie znakowe i bajtowe	284
	Predefiniowane strumienie	286
	Odczyt danych z konsoli	286
	Odczyt znaków	286
	Odczyt łańcuchów	287
	Wyświetlanie informacji na konsoli	289
	Klasa PrintWriter	289
	Odczyt i zapis plików	290
	Automatyczne zamykanie pliku	295
	Podstawy appletów	298
	Modyfikatory transient i volatile	301
	Operator instanceof	301
	Modyfikator strictfp	303
	Metody napisane w kodzie rdzennym	303
	Problemy z metodami rdzennymi	306
	Stosowanie asercji	306
	Opcje włączania i wyłączania asercji	308
	Import statyczny	309
	Wywoływanie przeciążonych konstruktorów za pomocą this()	311
	Kompaktowe profile API	313

14	Typy sparametryzowane	315
	Czym są typy sparametryzowane?	315
	Prosty przykład zastosowania typów sparametryzowanych	316
	Typy sparametryzowane działają tylko dla typów referencyjnych	319
	Typy sparametryzowane różnią się, jeśli mają inny argument typu	319
	W jaki sposób typy sparametryzowane zwiększają bezpieczeństwo?	319
	Klasa sparametryzowana z dwoma parametrami typu	321
	Ogólna postać klasy sparametryzowanej	322
	Typy ograniczone	322
	Zastosowanie argumentów wieloznacznych	324
	Ograniczony argument wieloznaczny	327
	Tworzenie metody sparametryzowanej	331
	Konstruktory sparametryzowane	332
	Interfejsy sparametryzowane	333
	Typy surowe i starszy kod	335
	Hierarchia klas sparametryzowanych	337
	Zastosowanie sparametryzowanej klasy bazowej	337
	Podklasa sparametryzowana	339
	Porównywanie typów w hierarchii klas sparametryzowanych	340
	w czasie wykonywania	342
	Rzutowanie	342
	Przykrywanie metod w klasach sparametryzowanych	342
	Wnioskowanie typów a typy sparametryzowane	343
	Znoszenie	344
	Metody mostu	344
	Błędy niejednoznaczności	346
	Pewne ograniczenia typów sparametryzowanych	347
	Nie można tworzyć egzemplarza parametru typu	347
	Ograniczenia dla składowych statycznych	347
	Ograniczenia tablic typów sparametryzowanych	347
	Ograniczenia wyjątków typów sparametryzowanych	348
15	Wyrażenia lambda	349
	Wprowadzenie do wyrażen lambda	349
	Podstawowe informacje o wyrażeniach lambda	350
	Interfejsy funkcyjne	351
	Kilka przykładów wyrażen lambda	352
	Blokowe wyrażenia lambda	354
	Sparametryzowane interfejsy funkcyjne	356
	Przekazywanie wyrażen lambda jako argumentów	357
	Wyrażenia lambda i wyjątki	360
	Wyrażenia lambda i przechwytywanie zmiennych	361
	Referencje do metod	362
	Referencje do metod statycznych	362
	Referencje do metod instancyjnych	363
	Referencje do metod a typy sparametryzowane	366
	Referencje do konstruktorów	368
	Predefiniowane interfejsy funkcyjne	372

CZĘŚĆ II

Biblioteka języka Java

16	Obsługa łańcuchów	377
	Konstruktory klasy String	377
	Długość łańcucha	379
	Specjalne operacje na łańcuchach	379
	Literały tekstowe	379
	Konkatenacja łańcuchów	380
	Konkatenacja łańcuchów z innymi typami danych	380
	Konwersja łańcuchów i metoda toString()	381
	Wyodrębnianie znaków	382
	Metoda charAt()	382
	Metoda getChars()	382
	Metoda getBytes()	382
	Metoda toCharArray()	383
	Porównywanie łańcuchów	383
	Metody equals() i equalsIgnoreCase()	383
	Metoda regionMatches()	384
	Metody startsWith() i endsWith()	384
	Metoda equals() kontra operator ==	384
	Metoda compareTo()	385
	Przeszukiwanie łańcuchów	386
	Modyfikowanie łańcucha	387
	Metoda substring()	387
	Metoda concat()	388
	Metoda replace()	388
	Metoda trim()	389
	Konwersja danych za pomocą metody valueOf()	389
	Zmiana wielkości liter w łańcuchu	390
	Łączenie łańcuchów	390
	Dodatkowe metody klasy String	391
	Klasa StringBuffer	391
	Konstruktory klasy StringBuffer	391
	Metody length() i capacity()	393
	Metoda ensureCapacity()	393
	Metoda setLength()	393
	Metody charAt() i setCharAt()	393
	Metoda getChars()	394
	Metoda append()	394
	Metoda insert()	395
	Metoda reverse()	395
	Metody delete() i deleteCharAt()	395
	Metoda replace()	396
	Metoda substring()	396
	Dodatkowe metody klasy StringBuffer	397
	Klasa StringBuilder	398
17	Pakiet java.lang	399
	Opakowania typów prostych	399
	Klasa Number	400
	Klasy Double i Float	400
	Klasy Byte, Short, Integer i Long	403

Klasa Character	411
Dodatki wprowadzone w celu obsługi punktów kodowych Unicode	413
Klasa Boolean	414
Klasa Void	415
Klasa Process	415
Klasa Runtime	416
Zarządzanie pamięcią	416
Wykonywanie innych programów	418
Klasa ProcessBuilder	419
Klasa System	421
Wykorzystanie metody currentTimeMillis() do obliczania czasu wykonywania programu ...	422
Użycie metody arraycopy()	423
Właściwości środowiska	424
Klasa Object	424
Wykorzystanie metody clone() i interfejsu Cloneable	425
Klasa Class	426
Klasa ClassLoader	429
Klasa Math	429
Funkcje trygonometryczne	429
Funkcje wykładnicze	430
Funkcje zaokrągleń	430
Inne metody klasy Math	430
Klasa StrictMath	433
Klasa Compiler	433
Klasa Thread i ThreadGroup oraz interfejs Runnable	433
Interfejs Runnable	433
Klasa Thread	433
Klasa ThreadGroup	435
Klasa ThreadLocal i InheritableThreadLocal	439
Klasa Package	439
Klasa RuntimePermission	439
Klasa Throwable	439
Klasa SecurityManager	439
Klasa StackTraceElement	439
Klasa Enum	441
Klasa ClassValue	442
Interfejs CharSequence	442
Interfejs Comparable	442
Interfejs Appendable	442
Interfejs Iterable	443
Interfejs Readable	443
Interfejs AutoCloseable	443
Interfejs Thread.UncaughtExceptionHandler	444
Podpakiety pakietu java.lang	444
Podpakiet java.lang.annotation	444
Podpakiet java.lang.instrument	444
Podpakiet java.lang.invoke	444
Podpakiet java.lang.management	444
Podpakiet java.lang.ref	445
Podpakiet java.lang.reflect	445

18	Pakiet java.util, część 1. — kolekcje	447
	Wprowadzenie do kolekcji	448
	Zmiany w kolekcjach wprowadzone w JDK 5	449
	Typy sparametryzowane w znaczący sposób zmieniają kolekcje	449
	Automatyczne opakowywanie ułatwia korzystanie z typów prostych	450
	Pętla for typu for-each	450
	Interfejsy kolekcji	450
	Interfejs Collection	451
	Interfejs List	453
	Interfejs Set	454
	Interfejs SortedSet	454
	Interfejs NavigableSet	455
	Interfejs Queue	455
	Interfejs Deque	455
	Klasy kolekcji	458
	Klasa ArrayList	458
	Klasa LinkedList	461
	Klasa HashSet	462
	Klasa LinkedHashSet	463
	Klasa TreeSet	464
	Klasa PriorityQueue	465
	Klasa ArrayDeque	465
	Klasa EnumSet	466
	Dostęp do kolekcji za pomocą iteratora	466
	Korzystanie z iteratora Iterator	468
	Pętla typu for-each jako alternatywa dla iteratora	469
	Spliteratory	470
	Przechowywanie w kolekcjach własnych klas	473
	Interfejs RandomAccess	474
	Korzystanie z map	474
	Interfejsy map	474
	Klasy map	479
	Komparatory	483
	Wykorzystanie komparatora	485
	Algorytmy kolekcji	490
	Klasa Arrays	495
	Starsze klasy i interfejsy	499
	Interfejs Enumeration	499
	Klasa Vector	500
	Klasa Stack	503
	Klasa Dictionary	504
	Klasa Hashtable	505
	Klasa Properties	508
	Wykorzystanie metod store() i load()	510
	Ostatnie uwagi na temat kolekcji	512
19	Pakiet java.util, część 2. — pozostałe klasy użytkowe	513
	Klasa StringTokenizer	513
	Klasa BitSet	514
	Klasy Optional, OptionalDouble, OptionalInt oraz OptionalLong	517
	Klasa Date	519
	Klasa Calendar	521
	Klasa GregorianCalendar	524
	Klasa TimeZone	525

Klasa SimpleTimeZone	525
Klasa Locale	527
Klasa Random	528
Klasa Observable	530
Interfejs Observer	531
Przykład użycia interfejsu Observer	531
Klasy Timer i TimerTask	533
Klasa Currency	535
Klasa Formatter	536
Konstruktory klasy Formatter	536
Metody klasy Formatter	537
Podstawy formatowania	537
Formatowanie łańcuchów i znaków	539
Formatowanie liczb	540
Formatowanie daty i godziny	540
Specyfikatory %n i %%	542
Określanie minimalnej szerokości pola	543
Określanie precyzji	544
Używanie znaczników (flag) formatów	545
Wyrównywanie danych wyjściowych	545
Znaczniki spacji, plusa, zera i nawiasów	546
Znacznik przecinka	547
Znacznik #	547
Opcja wielkich liter	547
Stosowanie indeksu argumentu	548
Zamykanie obiektu klasy Formatter	549
Metoda printf() w Javie	549
Klasa Scanner	549
Konstruktory klasy Scanner	549
Podstawy skanowania	551
Kilka przykładów użycia klasy Scanner	554
Ustawianie separatorów	557
Pozostałe elementy klasy Scanner	558
Klasy ResourceBundle, ListResourceBundle i PropertyResourceBundle	559
Dodatkowe klasy i interfejsy użytkowe	563
Podpakiety pakietu java.util	564
java.util.concurrent, java.util.concurrent.atomic oraz java.util.concurrent.locks	564
java.util.function	564
java.util.jar	567
java.util.logging	567
java.util.prefs	567
java.util.regex	567
java.util.spi	567
java.util.stream	567
java.util.zip	567
20 Operacje wejścia-wyjścia: analiza pakietu java.io	569
Klasy i interfejsy obsługujące operacje wejścia-wyjścia	570
Klasa File	570
Katalogi	573
Stosowanie interfejsu FilenameFilter	574
Alternatywna metoda listFiles()	575
Tworzenie katalogów	575
Interfejsy AutoCloseable, Closeable i Flushable	575
Wyjątki operacji wejścia-wyjścia	576

Dwa sposoby zamykania strumieni	576
Klasy strumieni	577
Strumienie bajtów	578
Klasa <code>InputStream</code>	578
Klasa <code>OutputStream</code>	579
Klasa <code>FileInputStream</code>	579
Klasa <code>FileOutputStream</code>	581
Klasa <code>ByteArrayInputStream</code>	583
Klasa <code>ByteArrayOutputStream</code>	584
Filtrowane strumienie bajtów	585
Buforowane strumienie bajtów	586
Klasa <code>SequenceInputStream</code>	589
Klasa <code>PrintStream</code>	590
Klasy <code>DataOutputStream</code> i <code>DataInputStream</code>	593
Klasa <code>RandomAccessFile</code>	594
Strumienie znaków	595
Klasa <code>Reader</code>	595
Klasa <code>Writer</code>	596
Klasa <code>FileReader</code>	596
Klasa <code>FileWriter</code>	597
Klasa <code>CharArrayReader</code>	598
Klasa <code>CharArrayWriter</code>	599
Klasa <code>BufferedReader</code>	600
Klasa <code>BufferedWriter</code>	601
Klasa <code>PushbackReader</code>	601
Klasa <code>PrintWriter</code>	602
Klasa <code>Console</code>	603
Serializacja	605
Interfejs <code>Serializable</code>	606
Interfejs <code>Externalizable</code>	606
Interfejs <code>ObjectOutput</code>	606
Klasa <code>ObjectOutputStream</code>	607
Interfejs <code>ObjectInput</code>	608
Klasa <code>ObjectInputStream</code>	608
Przykład serializacji	608
Korzyści wynikające ze stosowania strumieni	610
21 System NIO	611
Klasy systemu NIO	611
Podstawy systemu NIO	612
Bufory	612
Kanały	613
Zestawy znaków i selektory	614
Udoskonalenia dodane do systemu NIO w wydaniu JDK 7	615
Interfejs <code>Path</code>	615
Klasa <code>Files</code>	615
Klasa <code>Paths</code>	618
Interfejsy atrybutów plików	619
Klasy <code>FileSystem</code> , <code>FileSystems</code> i <code>FileStore</code>	620
Stosowanie systemu NIO	621
Stosowanie systemu NIO dla operacji wejścia-wyjścia na kanałach	621
Stosowanie systemu NIO dla operacji wejścia-wyjścia na strumieniach	629
Stosowanie systemu NIO dla operacji na ścieżkach i systemie plików	631
Przykłady stosowania kanałów w wersjach sprzed JDK 7	638
Odczytywanie plików (wersje sprzed JDK 7)	638
Zapisywanie plików (wersje sprzed JDK 7)	641

22	Obsługa sieci	645
	Podstawy działania sieci	645
	Klasy i interfejsy obsługujące komunikację siecią	646
	Klasa InetAddress	647
	Metody fabryczne	647
	Metody klasy	648
	Klasy InetAddress oraz Inet6Address	649
	Gniazda klientów TCP/IP	649
	URL	652
	Klasa URLConnection	654
	Klasa HttpURLConnection	656
	Klasa URI	658
	Pliki cookie	658
	Gniazda serwerów TCP/IP	658
	Datagramy	659
	Klasa DatagramSocket	659
	Klasa DatagramPacket	660
	Przykład użycia datagramów	660
23	Klasa Applet	663
	Dwa rodzaje apletów	663
	Podstawy apletów	664
	Klasa Applet	665
	Architektura apletu	667
	Szkielet apletu	667
	Inicjalizacja i przerywanie działania apletu	668
	Przykrycie metody update()	669
	Proste metody wyświetlania składników apletów	670
	Żądanie ponownego wyświetlenia	671
	Prosty aplet z paskiem reklamowym	672
	Wykorzystywanie paska stanu	674
	Znacznik APPLET języka HTML	675
	Przekazywanie parametrów do apletów	676
	Udoskonalenie apletu z paskiem reklamowym	677
	Metody getDocumentBase() i getCodeBase()	679
	Interfejs AppletContext i metoda showDocument()	679
	Interfejs AudioClip	681
	Interfejs AppletStub	681
	Wyświetlanie danych wyjściowych na konsoli	681
24	Obsługa zdarzeń	683
	Dwa mechanizmy obsługi zdarzeń	683
	Model obsługi zdarzeń oparty na ich delegowaniu	684
	Zdarzenia	684
	Źródła zdarzeń	684
	Obiekty nasłuchujące zdarzeń	685
	Klasy zdarzeń	685
	Klasa ActionEvent	686
	Klasa AdjustmentEvent	687
	Klasa ComponentEvent	688
	Klasa ContainerEvent	688
	Klasa FocusEvent	689
	Klasa InputEvent	689
	Klasa ItemEvent	690

Klasa KeyEvent	691
Klasa MouseEvent	691
Klasa MouseEvent	693
Klasa TextEvent	694
Klasa WindowEvent	694
Źródła zdarzeń	695
Interfejsy nasłuchujące zdarzeń	696
Interfejs ActionListener	696
Interfejs AdjustmentListener	696
Interfejs ComponentListener	697
Interfejs ContainerListener	697
Interfejs FocusListener	697
Interfejs ItemListener	697
Interfejs KeyListener	697
Interfejs MouseListener	697
Interfejs MouseMotionListener	698
Interfejs MouseWheelListener	698
Interfejs TextListener	698
Interfejs WindowFocusListener	698
Interfejs WindowListener	698
Stosowanie modelu delegowania zdarzeń	699
Obsługa zdarzeń generowanych przez mysz	699
Obsługa zdarzeń generowanych przez klawiaturę	701
Klasy adapterów	704
Klasy wewnętrzne	705
Anonimowa klasa wewnętrzna	707
25 Wprowadzenie do AWT: praca z oknami, grafiką i tekstem	709
Klasy AWT	710
Podstawy okien	712
Klasa Component	712
Klasa Container	712
Klasa Panel	713
Klasa Window	713
Klasa Frame	713
Klasa Canvas	713
Praca z oknami typu Frame	713
Ustawianie wymiarów okna	714
Ukrywanie i wyświetlanie okna	714
Ustawianie tytułu okna	714
Zamykanie okna typu Frame	714
Tworzenie okna typu Frame w aplecie AWT	714
Obsługa zdarzeń w oknie typu Frame	716
Tworzenie programu wykorzystującego okna	720
Wyświetlanie informacji w oknie	721
Wprowadzenie do stosowania grafiki	722
Rysowanie odcinków	722
Rysowanie prostokątów	722
Rysowanie elips, kół i okręgów	723
Rysowanie łuków	723
Rysowanie wielokątów	723
Prezentacja metod rysujących	723
Dostosowywanie rozmiarów obiektów graficznych	725

Praca z klasą Color	725
Metody klasy Color	726
Ustawianie bieżącego koloru kontekstu graficznego	727
Aplet demonstrujący zastosowanie klasy Color	727
Ustawianie trybu rysowania	728
Praca z czcionkami	729
Określanie dostępnych czcionek	730
Tworzenie i wybieranie czcionek	731
Uzyskiwanie informacji o czcionkach	733
Zarządzanie tekstowymi danymi wyjściowymi z wykorzystaniem klasy FontMetrics	734
Wyświetlanie tekstu w wielu wierszach	735
Wyśrodkowanie tekstu	737
Wyrównywanie wielowierszowych danych tekstowych	737
26 Stosowanie kontrolek AWT, menedżerów układu graficznego oraz menu	741
Podstawy kontrolek AWT	742
Dodawanie i usuwanie kontrolek	742
Odpowiadanie na zdarzenia kontrolek	742
Wyjątek HeadlessException	743
Etykiety	743
Stosowanie przycisków	744
Obsługa zdarzeń przycisków	744
Stosowanie pól wyboru	747
Obsługa zdarzeń pól wyboru	748
Klasa CheckboxGroup	749
Kontrolki list rozwijanych	751
Obsługa zdarzeń list rozwijanych	751
Stosowanie list	753
Obsługa zdarzeń generowanych przez listy	754
Zarządzanie paskami przewijania	755
Obsługa zdarzeń generowanych przez paski przewijania	756
Stosowanie kontrolek typu TextField	758
Obsługa zdarzeń generowanych przez kontrolkę TextField	759
Stosowanie kontrolek typu TextArea	760
Wprowadzenie do menedżerów układu graficznego komponentów	762
FlowLayout	762
BorderLayout	764
Stosowanie obramowań	766
GridLayout	767
Klasa CardLayout	768
Klasa GridBagLayout	771
Menu i paski menu	775
Okna dialogowe	780
FileDialog	784
Przykrywanie metody paint()	786
27 Obrazy	787
Formaty plików	787
Podstawy przetwarzania obrazów: tworzenie, wczytywanie i wyświetlanie	788
Tworzenie obiektu obrazu	788
Ładowanie obrazu	788
Wyświetlanie obrazu	789
Interfejs ImageObserver	790
Podwójne buforowanie	791
Klasa MediaTracker	793

Interfejs ImageProducer	796
Klasa MemoryImageSource	796
Interfejs ImageConsumer	797
Klasa PixelGrabber	797
Klasa ImageFilter	799
Klasa CropImageFilter	800
Klasa RGBImageFilter	801
Dodatkowe klasy obsługujące obrazy	810

28 Narzędzia współbieżności 811

Pakiety interfejsu Concurrent API	812
Pakiet java.util.concurrent	812
Pakiet java.util.concurrent.atomic	813
Pakiet java.util.concurrent.locks	813
Korzystanie z obiektów służących do synchronizacji	813
Klasa Semaphore	813
Klasa CountdownLatch	818
CyclicBarrier	819
Klasa Exchanger	821
Klasa Phaser	823
Korzystanie z egzekutorów	830
Przykład prostego egzekutora	830
Korzystanie z interfejsów Callable i Future	832
Typ wyliczeniowy TimeUnit	834
Kolekcje współbieżne	835
Blokady	835
Operacje atomowe	838
Programowanie równoległe przy użyciu frameworku Fork/Join	839
Najważniejsze klasy frameworku Fork/Join	839
Strategia dziel i zwyciężaj	843
Prosty przykład użycia frameworku Fork/Join	843
Znaczenie poziomu równoległości	845
Przykład użycia klasy RecursiveTask<V>	848
Asynchroniczne wykonywanie zadań	850
Anulowanie zadania	850
Określanie statusu wykonania zadania	851
Ponowne uruchamianie zadania	851
Pozostałe zagadnienia	851
Wskazówki dotyczące stosowania frameworku Fork/Join	853
Pakiet Concurrency Utilities a tradycyjne metody języka Java	853

29 API strumieni 855

Podstawowe informacje o strumieniach	855
Interfejsy strumieni	856
Jak można uzyskać strumień?	858
Prosty przykład stosowania strumieni	859
Operacje redukcji	862
Stosowanie strumieni równoległych	864
Odwzorowywanie	866
Tworzenie kolekcji	870
Iteratory i strumienie	873
Stosowanie typu Iterator i strumieni	873
Stosowanie spliteratorów	874
Inne możliwości API strumieni	877

30 Wyrażenia regularne i inne pakiety	879
Pakiety głównego API języka Java	879
Przetwarzanie wyrażeń regularnych	879
Klasa Pattern	881
Klasa Matcher	882
Składnia wyrażeń regularnych	882
Przykład dopasowywania do wzorca	883
Dwie opcje dopasowywania do wzorca	887
Przegląd wyrażeń regularnych	888
Refleksje	888
Zdalne wywoływanie metod (RMI)	891
Prosta aplikacja typu klient-serwer wykorzystująca RMI	891
Formatowanie dat i czasu przy użyciu pakietu java.text	894
Klasa DateFormat	894
Klasa SimpleDateFormat	896
Interfejs API dat i czasu dodany w JDK 8	897
Podstawowe klasy do obsługi dat i czasu	898
Formatowanie dat i godzin	899
Analiza łańcuchów zawierających daty i godziny	901
Inne możliwości pakietu java.time	902

CZĘŚĆ III

Wprowadzenie do programowania GUI przy użyciu pakietu Swing

31 Wprowadzenie do pakietu Swing	905
Geneza powstania biblioteki Swing	905
Bibliotekę Swing zbudowano na bazie zestawu narzędzi AWT	906
Podstawowe cechy biblioteki Swing	906
Komponenty biblioteki Swing są lekkie	906
Biblioteka Swing obsługuje dołączany wygląd i sposób obsługi	907
Podobieństwo do architektury MVC	907
Komponenty i kontenery	908
Komponenty	908
Kontenery	909
Panele kontenerów najwyższego poziomu	909
Pakiety biblioteki Swing	910
Prosta aplikacja na bazie biblioteki Swing	910
Obsługa zdarzeń	914
Tworzenie apletu na bazie biblioteki Swing	917
Rysowanie w bibliotece Swing	919
Podstawy rysowania	919
Wyznaczanie obszaru rysowania	920
Przykład rysowania	920
32 Przewodnik po pakiecie Swing	923
Klasy JLabel i ImageIcon	923
Klasa JTextField	925
Przyciski biblioteki Swing	926
Klasa JButton	927
Klasa JToggleButton	929
Poła wyboru	931
Przyciski opcji	932
Klasa JTabbedPane	934
Klasa JScrollPane	936

Klasa JList	938
Klasa JComboBox	941
Drzewa	943
Klasa JTable	945

33 Wprowadzenie do systemu menu pakietu Swing 949

Podstawy systemu menu	949
Przegląd klas JMenuBar, JMenu oraz JMenuItem	951
Klasa JMenuBar	951
Klasa JMenu	952
Klasa JMenuItem	953
Tworzenie menu głównego	953
Dodawanie mnemonik i kombinacji klawiszy do opcji menu	957
Dodawanie obrazów i etykiet ekranowych do menu	959
Stosowanie klas JRadioButtonMenuItem oraz JCheckBoxMenuItem	960
Tworzenie menu podręcznych	962
Tworzenie paska narzędzi	964
Stosowanie akcji	967
Finalna postać programu MenuDemo	971
Dalsze poznawanie pakietu Swing	977

CZĘŚĆ IV**Wprowadzenie do programowania GUI przy użyciu platformy JavaFX****34 Wprowadzenie do tworzenia interfejsów graficznych z użyciem JavaFX 981**

Podstawowe pojęcia z zakresu JavaFX	982
Pakiety JavaFX	982
Klasy Stage oraz Scene	982
Węzły i graf sceny	983
Układy	983
Klasa Application i metody cyklu życia	983
Uruchamianie aplikacji JavaFX	984
Szkielet aplikacji JavaFX	984
Kompilacja i uruchamianie programów JavaFX	987
Wątek aplikacji	988
Prosta kontrolka JavaFX: Label	988
Stosowanie przycisków i zdarzeń	990
Podstawowe informacje o zdarzeniach	990
Prezentacja kontrolki Button	991
Przedstawienie obsługi zdarzeń i kontrolki Button	992
Bezpośrednie rysowanie na płótnie	994

35 Prezentacja kontrolek JavaFX 999

Stosowanie klas Image i ImageView	999
Dodawanie obrazów do etykiet	1001
Stosowanie obrazów w przyciskach	1003
Kontrolka ToggleButton	1005
Kontrolka RadioButton	1008
Obsługa zdarzeń w grupie	1010
Alternatywne sposoby obsługi przycisków opcji	1011
Kontrolka CheckBox	1014
Kontrolka ListView	1017
Paski przewijania w kontrolkach ListView	1020
Włączanie możliwości wielokrotnego wyboru	1021

Kontrolka ComboBox	1022
Kontrolka TextField	1024
Kontrolka ScrollPane	1026
Kontrolka TreeView	1029
Prezentacja efektów i transformacji	1033
Efekty	1034
Transformacje	1035
Prezentacja zastosowania efektów i transformacji	1036
Dodawanie etykiet ekranowych	1038
Dezaktywacja kontroltek	1039
36 Prezentacja systemu menu platformy JavaFX	1041
Podstawowe informacje o menu	1041
Prezentacja klas MenuBar, Menu oraz MenuItem	1043
Klasa MenuBar	1043
Klasa Menu	1043
Klasa MenuItem	1044
Tworzenie menu głównego	1045
Dodawanie mnemonik i akceleratorów do elementów menu	1049
Dodawanie obrazów do elementów menu	1051
Stosowanie klas RadioMenuItem i CheckMenuItem	1051
Tworzenie menu podręcznego	1053
Tworzenie paska narzędzi	1057
Kompletna nowa wersja programu demonstracyjnego	1059
Dalsze poznawanie platformy JavaFX	1064

CZĘŚĆ V

Stosowanie Javy w praktyce

37 Java Beans	1067
Czym jest komponent typu Java Bean?	1067
Zalety komponentów Java Beans	1068
Introspekcja	1068
Wzorce właściwości	1068
Wzorce projektowe dla zdarzeń	1069
Metody i wzorce projektowe	1070
Korzystanie z interfejsu BeanInfo	1070
Właściwości ograniczone	1070
Trwałość	1071
Interfejs Customizer	1071
Interfejs Java Beans API	1071
Klasa Introspector	1073
Klasa PropertyDescriptor	1073
Klasa EventSetDescriptor	1073
Klasa MethodDescriptor	1073
Przykład komponentu Java Bean	1073
38 Serwlety	1077
Podstawy	1077
Cykl życia serwletu	1078
Alternatywne sposoby tworzenia serwletów	1078
Korzystanie z serwera Tomcat	1079
Przykład prostego serwletu	1080
Tworzenie i kompilacja kodu źródłowego serwletu	1080

Uruchamianie serwera Tomcat	1081
Uruchamianie przeglądarki i generowanie żądania	1081
Interfejs Servlet API	1081
Pakiet javax.servlet	1081
Interfejs Servlet	1082
Interfejs ServletConfig	1082
Interfejs ServletContext	1083
Interfejs ServletRequest	1083
Interfejs ServletResponse	1083
Klasa GenericServlet	1083
Klasa ServletInputStream	1083
Klasa ServletOutputStream	1085
Klasy wyjątków związanych z servletami	1085
Odczytywanie parametrów servletu	1085
Pakiet javax.servlet.http	1086
Interfejs HttpServletRequest	1087
Interfejs HttpServletResponse	1087
Interfejs HttpSession	1087
Klasa Cookie	1089
Klasa HttpServlet	1090
Obsługa żądań i odpowiedzi HTTP	1090
Obsługa żądań GET protokołu HTTP	1090
Obsługa żądań POST protokołu HTTP	1092
Korzystanie ze znaczników kontekstu użytkownika	1093
Śledzenie sesji	1095

DODATKI

A Komentarze dokumentujące	1099
Znaczniki narzędzia javadoc	1099
Znacznik @author	1100
Znacznik {@code}	1100
Znacznik @deprecated	1100
Znacznik {@docRoot}	1101
Znacznik @exception	1101
Znacznik {@inheritDoc}	1101
Znacznik {@link}	1101
Znacznik {@linkplain}	1101
Znacznik {@literal}	1101
Znacznik @param	1101
Znacznik @return	1101
Znacznik @see	1102
Znacznik @serial	1102
Znacznik @serialData	1102
Znacznik @serialField	1102
Znacznik @since	1102
Znacznik @throws	1102
Znacznik {@value}	1102
Znacznik @version	1103
Ogólna postać komentarzy dokumentacyjnych	1103
Wynik działania narzędzia javadoc	1103
Przykład korzystający z komentarzy dokumentacyjnych	1103
Skorowidz	1105

Wyliczenia, automatyczne opakowywanie typów prostych i adnotacje (metadane)

Ten rozdział dotyczy trzech stosunkowo nowych elementów języka programowania Java: typów wyliczeniowych, automatycznego opakowywania typów prostych i adnotacji (nazywanych także metadanymi). Każdy z nich rozszerza użyteczność języka, gdyż ułatwia wykonywanie typowych zadań programistycznych. W tym rozdziale zostaną dodatkowo omówione typy opakowujące (ang. *wrapper type*) i podstawy refleksji (ang. *reflection*).

Typy wyliczeniowe

Wersje Javy sprzed wydania JDK 5 nie zawierały pewnego elementu uważanego przez wielu programistów za bardzo przydatny: **typów wyliczeniowych**, tzw. **wyliczeń** (ang. *enumeration*). **Typ wyliczeniowy** w najprostszej postaci to lista nazwanych stałych. Choć Java oferuje inne elementy zapewniające podobne działanie, na przykład zmienne `final`, programiści oczekiwali raczej obsługi typów wyliczeniowych w tradycyjnej formie (w postaci, którą doskonale znali z wielu innych języków programowania). Począwszy od wydania JDK 5, typy wyliczeniowe są dostępne również dla programistów języka Java i stały się jego integralnym i często stosowanym elementem.

Wyliczenia w Javie w swojej podstawowej postaci przypominają wyliczenia znane z innych języków. Podobieństwo jest jednak tylko powierzchowne, gdyż w Javie wyliczenie definiuje typ klasy. Przekształcenie typów wyliczeniowych w pełnoprawne klasy pozwoliło znacznie wzbogacić ich możliwości. Na przykład wyliczenia w Javie mogą mieć konstruktory, metody i zmienne składowe. Choć trzeba było długo czekać na ich wprowadzenie, na pewno nikt nie czuje się zawiedziony ich możliwościami.

Podstawy wyliczeń

Wyliczenia tworzy się za pomocą nowego słowa kluczowego `enum`. Oto proste wyliczenie zawierające różne odmiany jabłek.

```
// Wyliczenie odmian jabłek.  
enum Apple {  
    Jonathan, GoldenDel, RedDel, Winesap, Cortland  
}
```

Identyfikatory Jonathan, GoldenDel itd. nazywamy **stałymi wyliczeniowymi**. Każda taka stała jest niejawnie zadeklarowana jako publiczna i statyczna składowa wyliczenia Apple. Co więcej, typ stałych jest taki sam jak typ wyliczenia, w którym się znajdują (w tym przypadku Apple). Z tego powodu w języku Java stałe te nazywamy stałymi **własnego typu**, ponieważ są tego samego typu co wyliczenie, do którego należą.

Po zdefiniowaniu wyliczenia można utworzyć zmienną typu wyliczeniowego. Choć wyliczenie definiuje typ klasy, do utworzenia jego egzemplarza nie stosuje się operatora new. Deklaracja zmiennej typu wyliczeniowego przypomina więc deklaracje dowolnego typu prostego. Poniższy wiersz dokonuje deklaracji zmiennej ap typu wyliczeniowego Apple.

```
Apple ap;
```

Ponieważ zmienna ap jest egzemplarzem typu Apple, jedynymi wartościami, które można jej przypisać (a tym samym: które może przechowywać), są stałe wyliczeniowe. Oto sposób przypisania do ap wartości RedDel.

```
ap = Apple.RedDel;
```

Zauważ, że nazwa stałej została poprzedzona nazwą wyliczenia.

Można sprawdzić równość dwóch stałych wyliczeniowych, używając operatora relacji ==. Poniższy kod porównuje wartość zmiennej ap ze stałą GoldenDel.

```
if(ap == Apple.GoldenDel) //...
```

Wartości wyliczenia mogą być używane do sterowania instrukcją switch. Oczywiście wszystkie elementy case muszą używać stałych z tego samego wyliczenia przekazanego w instrukcji switch. Poniższy kod instrukcji switch jest w zupełności poprawny.

// Wykorzystanie wyliczenia do sterowania instrukcją switch.

```
switch(ap) {
    case Jonathan:
        //...
    case Winesap:
        //...
```

Łatwo zauważyć, że stałe w klauzulach case nie są poprzedzane nazwą typu wyliczeniowego — zamiast zapisu Apple.Winesap zastosowano sam identyfikator Winesap. Wynika to z faktu, iż wyboru odpowiedniego typu wyliczeniowego dla klauzul case dokonano już w instrukcji switch. Próba użycia pełnej nazwy stałej wyliczeniowej spowoduje zgłoszenie błędu kompilacji.

Gdy wyświetla się stała wyliczeniową, na przykład za pomocą instrukcji println(), zostanie przekazana jej nazwa. Wykonanie poniższej instrukcji:

```
System.out.println(Apple.Winesap);
```

spowoduje wyświetlenie na ekranie identyfikatora Winesap.

Następujący program łączy w sobie wszystkie wspomniane cechy wyliczeń i przedstawia zastosowanie wyliczenia Apple.

// Wyliczenie odmian jabłek.

```
enum Apple {
    Jonathan, GoldenDel, RedDel, Winesap, Cortland
}
```

```
class EnumDemo {
    public static void main(String args[])
    {
        Apple ap;

        ap = Apple.RedDel;

        // Wyświetlenie wartości wyliczenia.
        System.out.println("Wartość ap: " + ap);
        System.out.println();

        ap = Apple.GoldenDel;
```



```
// Porównanie wartości dwóch wyliczeń.
if (ap == Apple.GoldenDel)
    System.out.println("ap zawiera GoldenDel.\n");

// Wykorzystanie wyliczenia do sterowania instrukcją switch.
switch (ap) {
    case Jonathan:
        System.out.println("Jonathan jest czerwone.");
        break;
    case GoldenDel:
        System.out.println("Golden Delicious jest żółte.");
        break;
    case RedDel:
        System.out.println("Red Delicious jest czerwone.");
        break;
    case Winesap:
        System.out.println("Winesap jest czerwone.");
        break;
    case Cortland:
        System.out.println("Cortland jest czerwone.");
        break;
}
}
```

Wynik działania programu jest następujący.

Wartość ap: RedDel

ap zawiera GoldenDel

Golden Delicious jest żółte.

Metody values() i valueOf()

Wszystkie wyliczenia automatycznie zawierają dwie predefiniowane metody: values() i valueOf(). Ich ogólna postać jest następująca.

```
public static typ-wyliczeniowy[] values()
public static typ-wyliczeniowy valueOf(String tekst)
```

Metoda values() zwraca tablicę zawierającą listę stałych wyliczeniowych. Metoda valueOf() zwraca stałą wyliczeniową, której odpowiada tekst (obiekt String) przekazany jako argument *tekst*. Dla wcześniejszego przykładu z wyliczeniem Apple wykonanie operacji Apple.valueOf("Winesap") spowodowałoby zwrócenie stałej wyliczeniowej Winesap.

Poniższy program demonstruje praktyczne zastosowanie metod values() i valueOf():

```
// Wykorzystanie wbudowanych metod typów wyliczeniowych.

// Wyliczenie odmian jabłek.
enum Apple {
    Jonathan, GoldenDel, RedDel, Winesap, Cortland
}

class EnumDemo2 {
    public static void main(String args[])
    {
        Apple ap;

        System.out.println("Oto wszystkie stałe wyliczenia Apple:");

        // użycie values()
        Apple allapples[] = Apple.values();
        for (Apple a : allapples)
            System.out.println(a);
    }
}
```

```

System.out.println();

// użycie valueOf()
ap = Apple.valueOf("Winesap");
System.out.println("ap zawiera " + ap);

}
}

```

Wynik działania programu jest następujący.

```

Oto wszystkie stałe wyliczenia Apple:
Jonathan
GoldenDel
RedDel
Winesap
Cortland

```

```
ap zawiera Winesap
```

Zauważ, że program używa pętli for typu for-each do przejścia przez wszystkie elementy tablicy uzyskanej za pośrednictwem metody `values()`. W celu lepszej ilustracji działania kodu utworzyłem zmienną `allApples` i przypisałem jej referencję do tablicy stałych wyliczeniowych. Ten krok nie jest jednak potrzebny, ponieważ pętlę można napisać w sposób przedstawiony poniżej, eliminując potrzebę stosowania zmiennej `allApples`.

```

for(Apple a : Apple.values())
    System.out.println(a);

```

Zwróć uwagę na to, iż wartość odpowiadająca nazwie `Winesap` została pobrana za pomocą metody `valueOf()`.

```
ap = Apple.valueOf("Winesap");
```

Jak już wspomniano, metoda `valueOf()` zwraca stałą wyliczeniową skojarzoną z przekazaną (w formie łańcucha) nazwą.

Wyliczenia Javy jako typy klasowe

Jak już wspomniano, typy wyliczeniowe w Javie mają postać typów klasowych. Choć nie tworzy się egzemplarzy wyliczenia za pomocą operatora `new`, we wszystkich innych sytuacjach wyliczenie przypomina tradycyjną klasę. Zaimplementowanie wyliczeń w Javie jako pełnoprawnych klas, daje im ogromne możliwości. Na przykład można utworzyć konstruktor, dodać zmienne lub metody składowe, a nawet zaimplementować interfejsy.

Trzeba zrozumieć, iż każda stała wyliczeniowa jest obiektem typu wyliczeniowego. Oznacza to, że konstruktor zdefiniowany w ramach typu wyliczeniowego będzie wywoływany przy okazji konstruowania każdej ze stałych wyliczeniowych. Co więcej, każda stała wyliczeniowa posiada własną kopię zmiennych składowych zdefiniowanych w wyliczeniu. Rozważmy następującą wersję wyliczenia `Apple`.

```

// Użycie konstruktora, zmiennej składowej i metody dla wyliczenia.
enum Apple {
    Jonathan(10), GoldenDel(9), RedDel(12), Winesap(15), Cortland(8);

    private int price; // cena każdego z jablek

    // konstruktor
    Apple(int p) { price = p; }

    int getPrice() { return price; }
}

class EnumDemo3 {
    public static void main(String args[])

```

```

{
    Apple ap;

    // Wyświetl cenę odmiany Winesap.
    System.out.println("Winesap kosztuje " +
        Apple.Winesap.getPrice() +
        " złotych.\n");

    // Wyświetl wszystkie jabłka i ich ceny.
    System.out.println("Ceny wszystkich jabłek:");
    for(Apple a : Apple.values())
        System.out.println(a + " kosztuje " + a.getPrice() +
            " złotych.");
}
}

```

Wynik działania programu jest następujący.

Winesap kosztuje 15 złotych.

Ceny wszystkich jabłek:
 Jonathan kosztuje 10 złotych.
 GoldenDel kosztuje 9 złotych.
 RedDel kosztuje 12 złotych.
 Winesap kosztuje 15 złotych.
 Cortland kosztuje 8 złotych.

Przedstawiona wersja programu zawiera trzy nowe elementy. Pierwszym jest zmienna składowa `price` przechowująca cenę danej odmiany jabłek. Po drugie, istnieje konstruktor `Apple()` przyjmujący jako parametr cenę jabłka. Trzeci element to metoda `getPrice()` zwracająca cenę jabłka.

W momencie deklarowania zmiennej `ap` w metodzie `main()` Java wywołuje konstruktor `Apple()` dla każdej stałej. Zauważ, że argumenty dla konstruktora przekazuje się, umieszczając je w nawiasie po nazwie stałej.

```
Jonathan(10), GoldenDel(9), RedDel(12), Winesap(15), Cortland(8);
```

Wartości te znajdują się w parametrze `p` konstruktora, który przypisze je do zmiennej składowej `price`. Pamiętaj, iż konstruktor zostanie wykonany dla każdej stałej.

Ponieważ każda stała wyliczeniowa posiada własną kopię zmiennej `price`, cenę danej odmiany jabłka pobiera się, wywołując metodę `getPrice()`. Cenę odmiany `Winesap` w metodzie `main()` pobieramy za pomocą poniższego wywołania.

```
Apple.Winesap.getPrice()
```

Ceny wszystkich odmian uzyskujemy, przetwarzając wyliczenie za pomocą `for`. Ponieważ każda stała wyliczeniowa dysponuje własną kopią zmiennej `price`, wartość skojarzona z jedną stałą jest przypisana tylko do tej stałej. Takie rozwiązanie jest niezwykle użyteczne, ale może być wprowadzone tylko wtedy, gdy wyliczenia zaimplementuje się jako klasy (jak w `Javie`).

Choć poprzedni przykład zawiera tylko jeden konstruktor, podobnie jak w tradycyjnych klasach można utworzyć kilka przeciążonych wersji konstruktora. Poniższa wersja wyliczenia `Apple` zawiera konstruktor domyślny ustalający cenę na wartość `-1`, czyli brak ceny.

```

// Użycie domyślnego konstruktora wyliczenia.
enum Apple {
    Jonathan(10), GoldenDel(9), RedDel, Winesap(15), Cortland(8);
    private int price; // cena każdego z jablek

    // konstruktor
    Apple(int p) { price = p; }

    // przeciążony konstruktor
    Apple() { price = -1; }

    int getPrice() { return price; }
}

```

W tej wersji odmiana `RedDel` nie przekazuje żadnego argumentu. Oznacza to, iż zostanie dla niej wywołany konstruktor domyślny powodujący ustawienie zmiennej `price` na wartość `-1`.

Ze stosowaniem typów wyliczeniowych wiążą się dwa ograniczenia. Po pierwsze, typ wyliczeniowy nie może dziedziczyć po innej klasie. Po drugie, wyliczenie nie może być klasą bazową. Innymi słowy, wyliczenia nie uczestniczą w hierarchii klas. Poza tymi ograniczeniami wyliczenia zachowują się podobnie do klas. Trzeba jednak pamiętać o tym, iż każda stała wyliczeniowa jest obiektem typu wyliczeniowego.

Wyliczenia dziedziczą po klasie Enum

Choć nie można korzystać z klasy bazowej przy definiowaniu wyliczenia, wszystkie wyliczenia automatycznie dziedziczą po klasie `java.lang.Enum`. Klasa ta definiuje kilka metod, z których można korzystać we wszystkich wyliczeniach. Klasa szczegółowo zostanie opisana w części II niniejszej książki, ale trzy jej metody warto opisać już teraz.

Można pobrać wartość liczbową przypisaną do stałej wyliczenia i wskazującą jej miejsce na liście. Wartość tę nazywa się **wartością kolejności** i pobiera się za pomocą metody `ordinal()`.

```
final int ordinal()
```

Metoda zwraca wartość kolejności stałej wyliczeniowej, dla której została wywołana. Wartości kolejności to liczby całkowite zaczynające się od 0. Z tego powodu w wyliczeniu `Apple` odmiana `Jonathan` posiada wartość kolejności 0, odmiana `GoldenDel` wartość 1, `RedDel` wartość 2 itd.

Do porównania wartości kolejności dwóch stałych wyliczeniowych służy metoda `compareTo()`. Jej ogólna postać jest następująca.

```
final int compareTo(typ-wyliczeniowy e)
```

Element *typ-wyliczeniowy* to typ wyliczenia; *e* to stała porównywana ze stałą, dla której została wywołana metoda. Pamiętaj, że stała, dla której jest wywoływana metoda, oraz stała przekazywana jako argument muszą być tego samego typu. Jeżeli wywoływana stała ma wartość kolejności mniejszą od *e*, wtedy metoda `compareTo()` zwraca wartość ujemną. Gdy wartości kolejności obu stałych są takie same, metoda zwraca 0. Jeżeli wywoływana stała ma wartość kolejności większą od *e*, wtedy metoda `compareTo()` zwraca wartość dodatnią.

Sprawdzanie równości stałej wyliczeniowej z dowolnym innym obiektem można wykonać za pomocą metody `equals()`, która przysłała domyślną metodę `equals()` klasy `Object`. Mimo że porównanie może dotyczyć stałej wyliczeniowej i dowolnego innego obiektu, metoda zwróci wartość `true` tylko wtedy, gdy przekazany obiekt będzie reprezentował tę samą stałą w ramach tego samego typu wyliczeniowego. Innymi słowy, metoda zwróci wartość `false` nawet wtedy, gdy przekaże się stałą innego wyliczenia, ale o takiej samej wartości kolejności.

Warto pamiętać, że dwie referencje do egzemplarzy typów wyliczeniowych zawsze można porównać za pomocą zwykłego operatora `==`.

Poniższy program ilustruje użycie metod `ordinal()`, `compareTo()` i `equals()`.

```
// Przykład użycia metod ordinal(), compareTo() i equals().
```

```
// Wyliczenie odmian jabłek.
```

```
enum Apple {
    Jonathan, GoldenDel, RedDel, Winesap, Cortland
}
```

```
class EnumDemo4 {
    public static void main(String args[])
    {
        Apple ap, ap2, ap3;

        // Pobranie wartości przypisanych do stałych - metoda ordinal().
        System.out.println("Oto wszystkie stałe wyliczenia" +
            " oraz ich wartości liczbowe: ");
        for(Apple a : Apple.values())
            System.out.println(a + " " + a.ordinal());
    }
}
```

```
ap = Apple.RedDel;
ap2 = Apple.GoldenDel;
ap3 = Apple.RedDel;

System.out.println();

// Przykład zastosowania metod compareTo() i equals().
if(ap.compareTo(ap2) < 0)
    System.out.println(ap + " znajduje się przed " + ap2);

if(ap.compareTo(ap2) > 0)
    System.out.println(ap2 + " znajduje się przed " + ap);

if(ap.compareTo(ap3) == 0)
    System.out.println(ap + " jest równe " + ap3);

System.out.println();

if(ap.equals(ap2))
    System.out.println("Błąd!");

if(ap.equals(ap3))
    System.out.println(ap + " jest równe " + ap3);

if(ap == ap3)
    System.out.println(ap + " == " + ap3);

}
}
```

Program po uruchomieniu wyświetla następujące wyniki.

Oto wszystkie stałe wyliczenia oraz ich wartości liczbowe:

```
Jonathan 0
GoldenDel 1
RedDel 2
Winesap 3
Cortland 4
```

```
GoldenDel znajduje się przed RedDel
```

```
RedDel jest równe RedDel
```

```
RedDel jest równe RedDel
```

```
RedDel == RedDel
```

Inny przykład wyliczenia

Zanim przejdziemy do omawiania dalszych zagadnień, warto przyjrzeć się jeszcze jednemu przykładowi użycia typu wyliczeniowego. W rozdziale 9. pojawił się program automatycznego podejmowania decyzji. W tamtej wersji zmienne reprezentujące możliwe odpowiedzi NO, YES, MAYBE, LATER, SOON i NEVER zostały zadeklarowane w interfejsie. Choć tamto podejście jest całkowicie poprawne, zastosowanie wyliczenia wydaje się lepszym wyborem. Oto ulepszona wersja tamtego programu, używająca wyliczenia o nazwie Answers do udzielania odpowiedzi.

```
// Ulepszona wersja programu podejmowania decyzji
// z rozdziału 9. Ta wersja używa wyliczenia
// zamiast zmiennych interfejsu do reprezentacji
// możliwych odpowiedzi.
```

```
import java.util.Random;
```

```
// Wyliczenie ze wszystkimi możliwymi odpowiedziami.
```

```
enum Answers {
    NO, YES, MAYBE, LATER, SOON, NEVER
```

```

}

class Question {
    Random rand = new Random();
    Answers ask() {
        int prob = (int) (100 * rand.nextDouble());

        if (prob < 15)
            return Answers.MAYBE; // 15%
        else if (prob < 30)
            return Answers.NO; // 15%
        else if (prob < 60)
            return Answers.YES; // 30%
        else if (prob < 75)
            return Answers.LATER; // 15%
        else if (prob < 98)
            return Answers.SOON; // 13%
        else
            return Answers.NEVER; // 2%
    }
}

class AskMe {
    static void answer(Answers result) {
        switch(result) {
            case NO:
                System.out.println("Nie");
                break;
            case YES:
                System.out.println("Tak");
                break;
            case MAYBE:
                System.out.println("Może");
                break;
            case LATER:
                System.out.println("Później");
                break;
            case SOON:
                System.out.println("Wkrótce");
                break;
            case NEVER:
                System.out.println("Nigdy");
                break;
        }
    }

    public static void main(String args[]) {
        Question q = new Question();
        answer(q.ask());
        answer(q.ask());
        answer(q.ask());
        answer(q.ask());
    }
}

```

Opakowania typów

Java używa typów prostych (nazywanych również typami podstawowymi), takich jak `int` lub `double`, do przechowywania podstawowych typów danych. Takie rozwiązanie podyktowane jest chęcią zwiększenia wydajności. Zastosowanie obiektów dla tych typów spowodowałoby ogromny narzut nawet w bardzo prostych obliczeniach. Oznacza to jednak, iż typy proste nie stanowią części hierarchii klasy i nie dziedziczą po klasie `Object`.

Mimo wszystkich zalet typów prostych (w wymiarze wydajności) w pewnych sytuacjach lepszym rozwiązaniem jest operowanie na obiektowej reprezentacji tych typów. Na przykład nie można przekazać typu prostego jako referencji, której wymaga metoda. Wiele standardowych struktur danych Javy opiera się na obiektach, co oznacza, iż do tych struktur nie można przekazać typów prostych. W celu zniwelowania tego rodzaju problemów Java zawiera **opakowania typów** — klasy hermetyzujące pojedynczą wartość typu prostego. Wszystkie te klasy zostaną szczegółowo omówione w części II książki; ponieważ jednak są bezpośrednio powiązane z funkcją automatycznego opakowywania i rozpakowywania typów prostych, warto wprowadzić je już teraz.

W Javie istnieją następujące opakowania typów prostych: Double, Float, Long, Integer, Short, Byte, Character i Boolean. Klasy te oferują wiele metod ułatwiających pełną integrację typów prostych z hierarchią klas Javy. Oto krótkie omówienie poszczególnych typów.

Klasa Character

Klasa Character to opakowanie dla typu char. Konstruktor klasy ma następującą postać.

```
Character(char zn)
```

Element *zn* to znak, który zostanie opakowany przez obiekt klasy Character.

Metoda `charValue()` służy do pobrania wartości char przechowywanej w obiekcie.

Aby uzyskać wartość typu char zawartą w obiekcie klasy Character, należy wywołać metodę `charValue()`:

```
char charValue()
```

Metoda zwraca znak reprezentowany przez obiekt opakowania.

Klasa Boolean

Klasa Boolean to opakowanie dla typu boolean. Konstruktorzy klasy mają następującą postać.

```
Boolean(boolean wartośćLogiczna)
```

```
Boolean(String wartośćŁańcuchowa)
```

W pierwszej wersji parametr *wartośćLogiczna* musi mieć albo wartość `true`, albo wartość `false`. W drugiej wersji, jeśli przekazany parametr *wartośćŁańcuchowa* zawiera łańcuch "true" (pisany albo małymi, albo wielkimi literami), nowy obiekt klasy Boolean będzie reprezentował wartość `true`. W przeciwnym razie będzie reprezentował wartość `false`.

Metoda `booleanValue()` służy do pobrania wartości boolean przechowywanej w obiekcie.

```
boolean booleanValue()
```

Metoda zwraca wartość `true` lub `false` reprezentowaną przez obiekt.

Opakowania typów numerycznych

Typy opakowujące najczęściej stosuje się dla wartości numerycznych. Oto typy związane z liczbami: Byte, Short, Integer, Long, Float i Double. Wszystkie te typy dziedziczą po klasie abstrakcyjnej Number. Klasa ta deklaruje metody zwracające wartość obiektu w kilku różnych formatach liczbowych. Metody te zostały wymienione poniżej.

```
byte byteValue()
```

```
double doubleValue()
```

```
float floatValue()
```

```
int intValue()
```

```
long longValue()
```

```
short shortValue()
```

Metoda `doubleValue()` zwraca wartość obiektu jako typ `double`, metoda `floatValue()` jako typ `float` itd. Wymienione metody są implementowane przez wszystkie liczbowe typy opakowujące.

Wszystkie opakowania dla liczb definiują dwa konstruktory, które umożliwiają utworzenie obiektu na podstawie podanej wartości lub jej reprezentacji łańcuchowej. Na przykład klasa `Integer` ma następujące konstruktory.

```
Integer(int wartość)
Integer(String łańcuch)
```

Jeżeli *łańcuch* nie zawiera poprawnej reprezentacji tekstowej liczby, konstruktor zgłasza wyjątek `NumberFormatException`.

Wszystkie typy opakowujące przykrywają metodę `toString()`, która zwraca czytelną, tekstową (łańcuchową) reprezentację wartości przechowywanej w danym obiekcie. Umożliwia to między innymi bezpośrednio przekazywanie obiektu jako argumentu metody `println()`. Nie trzeba dokonywać wcześniejszej konwersji do typu prostego.

Poniższy przykładowy program ilustruje wykorzystanie typu opakowującego liczby całkowitej do hermetyzacji wartości, a następnie jej wydobywania.

// Przykład użycia opakowania.

```
class Wrap {
    public static void main(String args[]) {

        Integer i0b = new Integer(100);

        int i = i0b.intValue();

        System.out.println(i + " " + i0b); // wyświetla 100 100
    }
}
```

Kod opakowuje wartość całkowitą 100 obiektem klasy `Integer` nazwanym `i0b`. Następnie pobiera tę wartość z obiektu `i` i przypisuje ją do zmiennej `i`.

Proces umieszczania wartości wewnątrz nazywa się **opakowywaniem**. Następujący wiersz opakowuje więc wartość 100 w obiekcie typu `Integer`.

```
Integer i0b = new Integer(100);
```

Proces wydobywania wartości z typu opakowującego nazywa się **rozpakowywaniem** lub **wydobywaniem**. Następujący wiersz wydobywa wartość z obiektu `i0b`.

```
int i = i0b.intValue();
```

Powyższy sposób opakowywania i wydobywania wartości był stosowany od samego początku w języku Java. Wraz z wydaniem JDK 5 wprowadzono jednak rozwiązanie, które znacznie uprościło stosowanie opakowań — automatyczne opakowywanie i wydobywanie typów prostych.

Automatyczne opakowywanie typów prostych

Począwszy od wydania JDK 5, język Java oferuje dwa ważne mechanizmy: **automatyczne opakowywanie** i **automatyczne wydobywanie typów prostych**. Automatyczne opakowywanie oznacza, iż typ prosty jest automatycznie umieszczany w odpowiadającym mu typie opakowującym, jeśli sytuacja wymaga użycia obiektu. Nie trzeba jawnie tworzyć obiektu. Automatyczne wydobywanie oznacza, iż typ opakowujący jest automatycznie rozpakowywany w razie konieczności użycia typu prostego. Nie trzeba jawnie wywoływać metody `intValue()` lub `doubleValue()`.

Nowa funkcja języka Java ułatwia pisanie niektórych algorytmów, ponieważ eliminuje konieczność żmudnego, ręcznego opakowywania i wydobywania wartości. Co więcej, takie rozwiązanie zapobiega popełnianiu błędów i stanowi jeden z ważnych elementów typów sparametryzowanych, które działają jedynie dla obiektów. Automatyczne opakowywanie ułatwia też korzystanie z kolekcji frameworku `Collections`, które zostaną omówione w części II książki.

Automatyczne opakowywanie eliminuje konieczność ręcznego tworzenia obiektów opakowujących wartości typów prostych — wystarczy przypisać typ prosty zmiennej referencyjnej typu opakowującego. Java automatycznie utworzy odpowiedni obiekt. Oto nowoczesny sposób tworzenia obiektu `Integer` przechowującego wartość 100.


```
Integer i0b = 100; // automatyczne opakowanie typu int
```

Zauważ, że nie pojawia się jawne tworzenie obiektu za pomocą operatora `new`. Java wykonuje to zadanie za programistę.

W celu rozpakowania obiektu wystarczy przypisać zmiennej odpowiedniego typu prostego referencję do obiektu. Oto przykład rozpakowania obiektu `i0b`.

```
int i = i0b; // automatyczne rozpakowanie
```

Java sama zajmuje się wszystkimi szczegółami.

Poniżej znajduje się wcześniejszy program napisany z wykorzystaniem automatycznego opakowywania i rozpakowywania.

// Przykład automatycznego opakowywania.

```
class AutoBox {
    public static void main(String args[]) {

        Integer i0b = 100; // automatyczne opakowanie typu int

        int i = i0b; // automatyczne rozpakowanie

        System.out.println(i + " " + i0b); // wyświetla 100 100
    }
}
```

Automatyczne opakowywanie i metody

Działanie tego mechanizmu nie ogranicza się tylko do operacji przypisania. Automatyczne opakowywanie występuje zawsze wtedy, gdy typ prosty wymaga konwersji na obiekt. Automatyczne rozpakowanie występuje zawsze wtedy, gdy obiekt musi zostać zamieniony na typ prosty. Oznacza to, iż automatyczne opakowanie może mieć miejsce na przykład w momencie przekazania argumentu do metody lub zwrócenia wartości przez metodę. Oto przykład wart przeanalizowania.

*// Automatyczne opakowanie i rozpakowanie dotyczy
// także argumentów metod i zwracanych wartości.*

```
class AutoBox2 {
    // Pobierz parametr Integer
    // i zwróć wartość typu int.
    static int m(Integer v) {
        return v ; // automatyczne rozpakowanie do int
    }

    public static void main(String args[]) {
        // Przekazanie wartości typu int do m() i przypisanie zwróconej wartości
        // obiektowi typu Integer. Argument 100 jest automatycznie opakowany typem
        // Integer. Zwracana wartość jest ponownie automatycznie opakowywana
        // typem Integer.
        Integer i0b = m(100);

        System.out.println(i0b);
    }
}
```

Program wyświetli na ekranie następującą wartość.

```
100
```

Zauważ, że metoda `m()` wymaga argumentu typu `Integer` i zwraca jako wynik typ `int`. Wewnątrz metody `main()` metodzie `m()` jest przekazywana wartość `100`. Ponieważ metoda `m()` wymaga typu `Integer`, przekazana wartość jest automatycznie opakowywana. Następnie metoda zwraca typ `int` odpowiadający wartości z obiektu `Integer`. Powoduje to automatyczne rozpakowanie `v`. Ponieważ uzyskana wartość jest przypisywana do zmiennej `i0b`, następuje ponowne automatyczne opakowanie wartości typu `int`.

Automatyczne opakowywanie i rozpakowywanie w wyrażeniach

Ogólnie automatyczne opakowywanie lub rozpakowywanie ma miejsce zawsze wtedy, gdy wymagana jest konwersja na obiekt lub z obiektu. Dotyczy to również wyrażień. W wyrażeniu obiekt numeryczny jest automatycznie rozpakowywany. W razie konieczności wynik wyrażenia jest opakowywany obiektem. Oto przykład takiego zachowania.

// Automatyczne opakowywanie w wyrażeniach.

```
class AutoBox3 {
    public static void main(String args[]) {
        Integer iOb, iOb2;
        int i;

        iOb = 100;
        System.out.println("Oryginalna wartość iOb: " + iOb);

        // Poniższy kod powoduje automatyczne rozpakowanie iOb,
        // inkrementowanie wartości i ponownie opakowanie
        // wyniku w obiekcie iOb.
        ++iOb;
        System.out.println("Po ++iOb: " + iOb);

        // Tutaj iOb jest rozpakowywane, obliczana jest wartość
        // wyrażenia i wynik jest ponownie pakowany do iOb2.
        iOb2 = iOb + (iOb / 3);
        System.out.println("iOb2 po wyrażeniu: " + iOb2);

        // Obliczenie tego samego wyrażenia,
        // ale bez automatycznego opakowywania.
        i = iOb + (iOb / 3);
        System.out.println("i po wyrażeniu: " + i);
    }
}
```

Program generuje następujące wyniki.

```
Oryginalna wartość iOb: 100
Po ++iOb: 101
iOb2 po wyrażeniu: 134
i po wyrażeniu: 134
```

Warto zwrócić szczególną uwagę na poniższy wiersz.

```
++iOb;
```

Wyrażenie w tej formie oznacza inkrementację wartości przechowywanej w obiekcie iOb: najpierw dochodzi do rozpakowania wartości, następnie jest ona inkrementowana, a na końcu ponownie opakowywana obiektem.

Automatyczne rozpakowywanie umożliwia mieszanie w jednym wyrażeniu różnych typów obiektów numerycznych. Po rozpakowaniu zachodzi standardowa promocja i konwersja typów. Poniższy program jest całkowicie poprawny.

```
class AutoBox4 {
    public static void main(String args[]) {

        Integer iOb = 100;
        Double dOb = 98.6;

        dOb = dOb + iOb;
        System.out.println("dOb po wyrażeniu: " + dOb);
    }
}
```

Wynik działania programu jest następujący.

```
d0b po wyrażeniu: 198.6
```

Zarówno obiekt d0b typu Double, jak i obiekt i0b typu Integer uczestniczył w dodawaniu. Wynik został opakowany i umieszczony w obiekcie d0b.

Dzięki automatycznemu rozpakowywaniu można używać obiektów liczb całkowitych do sterowania instrukcją switch. Oto przykład.

```
Integer i0b = 2;
```

```
switch(i0b) {
    case 1: System.out.println("jeden");
        break;
    case 2: System.out.println("dwa");
        break;
    default: System.out.println("błąd");
}
```

W momencie obliczania wyrażenia z instrukcji switch obiekt i0b jest automatycznie rozpakowywany.

Jak łatwo zauważyć, automatyczne opakowywanie i rozpakowywanie ułatwia pisanie wyrażen korzystających z obiektów numerycznych. Dawniej, aby osiągnąć ten sam efekt, trzeba było dokonywać rzutowania i wywoływania metod takich jak intValue().

Automatyczne opakowywanie typów znakowych i logicznych

Jak już wspomniano, Java udostępnia także klasy opakowań dla typów boolean i char: odpowiednio Boolean i Character. Automatyczne opakowywanie i rozpakowywanie dotyczy również tych typów. Oto przykład.

// Automatyczne opakowywanie i rozpakowywanie obiektów klas Boolean i Character.

```
class AutoBox5 {
    public static void main(String args[]) {

        // automatyczne opakowanie/rozpakowanie wartości typu boolean
        Boolean b = true;

        // Zastosowanie b w wyrażeniu warunkowym powoduje
        // jego automatyczne rozpakowanie.
        if(b) System.out.println("b wynosi true");

        // automatyczne opakowanie/rozpakowanie wartości typu char
        Character ch = 'x'; // opakowanie typu char
        char ch2 = ch; // rozpakowanie char
        System.out.println("ch2 wynosi " + ch2);
    }
}
```

Program generuje następujące wyniki.

```
b wynosi true
ch2 wynosi x
```

W tym programie warto przede wszystkim zauważyć, iż zachodzi automatyczne rozpakowanie b w wyrażeniu warunkowym instrukcji if. Każde wyrażenie instrukcji if musi dać w wyniku wartość typu boolean. Ponieważ b zawiera wartość logiczną, dochodzi do automatycznego rozpakowania i użycia tej wartości jako wyniku wyrażenia. Z tego względu wspomniana instrukcja jest dla kompilatora w pełni poprawna.

Dzięki automatycznemu rozpakowywaniu można stosować obiekt Boolean w strukturach sterujących. Wewnątrz pętli while, for i do-while dochodzi do automatycznego rozpakowania obiektu do typu boolean. Poniższy kod jest w pełni poprawny.

```
Boolean b;
//...
while(b) { //...
```

Automatyczne opakowywanie pomaga zapobiegać błędom

Automatyczne opakowywanie, poza samą wygodą, pomaga zapobiegać błędom programistów. Rozważmy następujący program.

// Błąd przy ręcznym rozpakowywaniu.

```
class UnboxingError {
    public static void main(String args[]) {

        Integer i0b = 1000; // automatyczne opakowanie wartości 1000

        int i = i0b.byteValue(); // ręczne wydobywanie wartości!!!

        System.out.println(i); // nie spowoduje wyświetlenia wartości 1000!
    }
}
```

Program nie wyświetli oczekiwanej wartości 1000, ale -24! Wynika to z faktu, iż rozpakowanie zostało wykonane metodą `byteValue()`, która spowodowała obcięcie wartości do zakresu dostępnego dla typu `byte`. Automatyczne rozpakowywanie zabezpiecza przed tego rodzaju błędami, ponieważ Java automatycznie zastosuje metodę zgodną z typem `int`.

Ponieważ automatyczne opakowywanie zawsze tworzy odpowiedni obiekt, a automatyczne wydobywanie zawsze pobiera odpowiednią wartość, nie można doprowadzić do błędnego doboru typu i wartości. W tych rzadkich przypadkach, w których programista świadomie chce dokonać zmiany typu, zawsze może zastosować ręczne opakowywanie lub rozpakowywanie. Oczywiście traci się wtedy zalety automatyzacji. Ogólnie: nowo tworzony kod języka Java powinien stosować automatyczne opakowywanie.

Słowo ostrzeżenia

Ponieważ język Java dokonuje automatycznego opakowywania i rozpakowywania, niektórzy programiści mogą ulec pokusie całkowitej rezygnacji z tradycyjnych typów prostych na rzecz obiektów klasy `Integer` czy `Double`. Można na przykład napisać następujący kod.

// Złe użycie automatycznego opakowywania!

```
Double a, b, c;

a = 10.0;
b = 4.0;

c = Math.sqrt(a*a + b*b);

System.out.println("Przeciwprostokątna ma wartość " + c);
```

Obiekty typu `Double` przechowują wartości używane do obliczenia przeciwprostokątnej trójkąta prostokątnego. Choć kod jest poprawny od strony technicznej i wykonuje swoje zadanie bez problemów, stanowi bardzo zły przykład użycia automatycznego opakowywania. Dużo wydajniejsze obliczeniowo będzie w tym przypadku zastosowanie typu prostego `double`. Różnica w wydajności obu rozwiązań wynika z tego, że automatyczne opakowywanie wiąże się z kosztami, które nie występują w przypadku typów prostych.

Ogólnie: należy stosować typy opakowujące tylko wtedy, gdy potrzebna jest obiektowa reprezentacja typów prostych. Automatyczne opakowywanie nie zostało dodane do Javy, by po cichu wyeliminować typy proste.

Adnotacje (metadane)

Począwszy od wydania JDK 5, język Javy obsługuje nowy sposób umieszczania dodatkowych informacji w plikach z kodem źródłowym. Nowy mechanizm, który nazwano **adnotacjami**, w żaden sposób nie wpływa na działanie programu. Adnotacje pozostawiają niezmienną semantykę programu. Okazuje się jednak, że informacje zawarte w adnotacjach mogą być używane przez rozmaite narzędzia na etapie wytwarzania i wdrażania oprogramowania. Adnotacja może zostać przetworzona na przykład przez generator kodu źródłowego. Opisywany mechanizm bywa też określany mianem **metadanych**, jednak nazwa **adnotacje** lepiej oddaje istotę tego rozwiązania i cieszy się większą popularnością.

Podstawy tworzenia adnotacji

Adnotacje można tworzyć przy użyciu mechanizmu na bazie interfejsu. Zaczniemy od przykładu, który deklaruje adnotację `MyAnno`.

// Przykład prostego typu adnotacji.

```
@interface MyAnno {
    String str();
    int val();
}
```

Zauważ znak `@` poprzedzający słowo kluczowe `interface`. Za pomocą tego znaku można zasygnalizować kompilatorowi, że ma do czynienia z deklaracją typu adnotacji. Wewnątrz adnotacji zadeklarowano dwie metody składowe: `str()` i `val()`. Wszystkie adnotacje składają się tylko z deklaracji metod. Nie podaje się kodu deklarowanych metod, Java sama je implementuje. Co więcej, metody te zachowują się jak zmienne.

Adnotacja nie może zawierać klauzuli `extends`. Okazuje się jednak, że wszystkie adnotacje automatycznie rozszerzają interfejs `Annotation`. Deklaracja tego głównego interfejsu znajduje się w pakiecie `java.lang.annotation`. Przesłania on metody `hashCode()`, `equals()` i `toString()` zdefiniowane w klasie `Object`. Dodatkowo zawiera metodę `annotationType()`, która zwraca obiekt `Class` reprezentujący wywołaną adnotację.

Po zadeklarowaniu adnotacji można zacząć jej używać do oznaczania innych elementów kodu. Przed wprowadzeniem JDK 8 adnotacje można było dodawać wyłącznie do deklaracji i właśnie ten sposób ich stosowania przedstawimy w pierwszej kolejności. (W JDK 8 wprowadzono także możliwość dodawania adnotacji do typów — przedstawię ją w dalszej części rozdziału. Niemniej jednak pewne podstawowe techniki odnoszą się do obu rodzajów adnotacji). Adnotacje można stosować praktycznie dla dowolnych rodzajów deklaracji, w tym deklaracji klas, metod, zmiennych, parametrów i stałych wyliczeniowych. Istnieje nawet możliwość oznaczenia jedną adnotacją deklaracji innej adnotacji. We wszystkich przypadkach treść adnotacji musi znaleźć się przed właściwą deklaracją.

Podczas stosowania adnotacji należy przypisać wartości jej składowym. Oto przykład zastosowania adnotacji dla metody:

// Adnotacja dotycząca metody.

```
@MyAnno(str = "Przykład adnotacji", val = 100)
public static void myMeth() { //...
```

Adnotacja jest powiązana z metodą `myMeth()`. Warto dobrze przyjrzeć się stosowanej składni. Nazwa rodzaju adnotacji jest poprzedzona znakiem `@`, a w nawiasach znajduje się lista inicjalizująca składowe. Aby nadać składowej wartość, wystarczy podać nazwę składowej, znak równości i wartość. W tym przykładzie łańcuch "Przykład adnotacji" zostaje przypisany do składowej `str`. Łatwo zauważyć, że w przytoczonym wyrażeniu po identyfikatorze `str` nie użyto nawiasów. Podczas przypisywania wartości składowej adnotacji wystarczy użyć nazwy tej składowej. W tym kontekście metoda wygląda, jakby była zmienną składową.

Określanie strategii zachowywania adnotacji

Zanim przystąpimy do szczegółowego omawiania adnotacji, warto wspomnieć o **strategii zachowywania adnotacji**. Strategia określa, w którym momencie adnotacje są pomijane. Java definiuje trzy takie strategie — znajdują się one w wyliczeniu `java.lang.annotation.RetentionPolicy`. Nazwy tych strategii to `SOURCE`, `CLASS` i `RUNTIME`.

Adnotacja ze strategią `SOURCE` jest zachowywana tylko w pliku kodu źródłowego. Innymi słowy, jest pomijana w trakcie kompilacji kodu.

Adnotacja ze strategią `CLASS` jest zachowywana w pliku `.class` w trakcie kompilacji. Nie jest jednak dostępna dla maszyny wirtualnej w trakcie wykonywania programu.

Adnotacja ze strategią `RUNTIME` jest zachowywana w pliku `.class` w trakcie kompilacji i jest dostępna dla maszyny wirtualnej w trakcie wykonywania programu. Oznacza to, że właśnie strategia `RUNTIME` zapewnia największą trwałość adnotacji.



Uwaga

Adnotacja zastosowana dla deklaracji zmiennej lokalnej nigdy nie jest zachowywana w pliku `.class`.

Strategię zachowywania adnotacji dla notatki można wskazać, stosując jedną z wbudowanych adnotacji `@Retention`. Oto ogólna postać tej adnotacji.

```
@Retention(strategia-zachowywania)
```

Element *strategia-zachowywania* musi być jedną z wcześniej omówionych stałych wyliczeniowych. Jeżeli nie określi się jawnie żadnej strategii, domyślnie zostanie użyta strategia `CLASS`.

Poniższa wersja adnotacji `MyAnno` używa adnotacji `@Retention` do określenia strategii zachowywania `RUNTIME`. Oznacza to, że adnotacja `MyAnno` będzie dostępna dla maszyny wirtualnej w trakcie wykonywania programu.

```
@Retention(RetentionPolicy.RUNTIME)
@interface MyAnno {
    String str();
    int val();
}
```

Odczytywanie adnotacji w trakcie działania programu za pomocą refleksji

Mimo że adnotacje zaprojektowano przede wszystkim z myślą o narzędziach programistycznych i wdrożeniowych, adnotacje stosujące strategię `RUNTIME` można odczytywać w czasie wykonywania programu, korzystając z tzw. **refleksji**. Refleksja umożliwia pobranie informacji na temat klasy w trakcie działania programu. Interfejs API refleksji umieszczono w pakiecie `java.lang.reflect`. Istnieje wiele zastosowań tego mechanizmu, więc nie będę ich tu wszystkich wymieniał. Ograniczymy się tylko do analizy kilku przykładów związanych z adnotacjami.

Pierwszy krok wykorzystania refleksji polega na pobraniu obiektu klasy `Class` reprezentującego klasę, której adnotację chcemy odczytać. Klasa `Class` to jedna z klas wbudowanych w język Java i znajdujących się w pakiecie `java.lang`. Jest ona szczegółowo omówiona w części II książki. Istnieje wiele sposobów uzyskania obiektu `Class`. Jednym z nich jest zastosowanie metody `getClass()` zdefiniowanej w klasie `Object`. Oto ogólna postać metody.

```
final Class<?> getClass()
```

Metoda zwraca obiekt klasy `Class` reprezentujący klasę obiektu wywołującego tę metodę.



Uwaga

Warto zwrócić uwagę na konstrukcję `<?>` użytą po słowie `Class` w powyższej deklaracji metody `getClass()`. Wspomniana konstrukcja ma związek z dostępnym w Javie mechanizmem typów sparametryzowanych. Metoda `getClass()`, jak wiele innych prezentowanych tutaj metod związanych z refleksjami, korzysta z typów sparametryzowanych. Same typy sparametryzowane zostaną omówione w rozdziale 14. Dobre rozumienie typów sparametryzowanych nie jest jednak niezbędne do zrozumienia podstawowych zasad funkcjonowania refleksji.

Po uzyskaniu obiektu klasy `Class` można użyć jego metod do odczytania informacji na temat rozmaitych elementów zadeklarowanych w tej klasie, w tym adnotacji. Aby uzyskać adnotację związaną z konkretnym elementem zadeklarowanym w klasie, należy najpierw pobrać obiekt reprezentujący ten element. Klasa `Class` zawiera między innymi metody `getMethod()`, `getField()` i `getConstructor()`, które pobierają informacje odpowiednio o metodzie, polu i konstruktorze. Metody zwracają obiekty typu `Method`, `Field` i `Constructor`.

Aby lepiej prześledzić cały proces, najlepiej przeanalizować przykład uzyskujący adnotacje związane z metodą. W tym celu trzeba pobrać obiekt `Class` reprezentujący klasę, a następnie wywołać dla niego metodę `getMethod()`, przekazując nazwę metody jako parametr. Ogólna postać metody `getMethod()` jest następująca.

```
Method getMethod(String nazwaMetody, Class<?> ... typyParametrów)
```

Nazwę metody przekazuje się w elemencie *nazwaMetody*. Jeśli metoda przyjmuje argumenty, trzeba przekazać obiekty `Class` reprezentujące typy parametrów (element *typyParametrów*). Zauważ, że *typyParametrów* to parametr o zmiennej liczbie argumentów. Oznacza to, iż można przekazać dowolną liczbę typów parametrów, również zero. Metoda `getMethod()` zwraca obiekt `Method` reprezentujący metodę. Jeśli nie uda się odnaleźć metody, zostanie zgłoszony wyjątek `NoSuchMethodException`.

Adnotacje skojarzone z obiektami klas `Class`, `Method`, `Field` i `Constructor` można uzyskać za pośrednictwem metody `getAnnotation()`. Ogólna postać tej metody jest następująca.

```
Annotation getAnnotation(Class<A> typAdnotacji)
```

Element *typAdnotacji* to obiekt `Class` reprezentujący interesującą nas adnotację. Metoda zwraca referencję do adnotacji. Za pośrednictwem tej referencji można uzyskać wartości reprezentowane przez składowe tej adnotacji. Jeśli nie uda się znaleźć żądanej adnotacji (jeśli na przykład zastosowano dla adnotacji inną strategię zachowywania niż `RUNTIME`), metoda zwróci wartość `null`.

Poniżej pokazano program łączący w sobie wszystkie opisane powyżej elementy i używający refleksji do wyświetlenia adnotacji związanej z metodą.

```
import java.lang.annotation.*;
import java.lang.reflect.*;

// Deklaracja typu adnotacji.
@Retention(RetentionPolicy.RUNTIME)
@interface MyAnno {
    String str();
    int val();
}

class Meta {

    // Adnotacja metody.
    @MyAnno(str = "Przykładowa adnotacja", val = 100)
    public static void myMeth() {
        Meta ob = new Meta();

        // Pobranie adnotacji dla tej metody i
        // wyświetlenie wartości jej składowych.
        try {
            // Najpierw pobierz obiekt Class reprezentujący
            // aktualną klasę.
            Class c = ob.getClass();

            // Pobierz obiekt Method reprezentujący
            // aktualną metodę.
            Method m = c.getMethod("myMeth");

            // Następnie pobierz adnotację dla tej metody.
            MyAnno anno = m.getAnnotation(MyAnno.class);

            // Wyświetl wartości adnotacji.
            System.out.println(anno.str() + " " + anno.val());
        }
    }
}
```

```

    } catch (NoSuchMethodException exc) {
        System.out.println("Nie znaleziono metody.");
    }
}

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

```

Wynik działania programu jest następujący.

Przykładowa adnotacja 100

Program używa refleksji do pobrania i wyświetlenia wartości elementów `str` i `val` adnotacji `MyAnno` przypisanej do metody `myMeth()` klasy `Meta`. Warto zwrócić szczególną uwagę na dwa wiersze. Oto pierwszy z nich.

```
MyAnno anno = m.getAnnotation(MyAnno.class);
```

Szczególnie interesujące jest wyrażenie `MyAnno.class`, które powoduje zwrócenie obiektu klasy `Class` dla typu `MyAnno`, czyli adnotacji. Taką konstrukcję nazywamy **stałą klasową**. Podobne konstrukcje można stosować wszędzie tam, gdzie jest potrzebny obiekt typu `Class` dla znanej klasy. Na przykład poniższego wyrażenia można by użyć do uzyskania obiektu typu `Class` dla klasy `Meta`:

```
Class<?> c = Meta.class;
```

Oczywiście takie rozwiązanie działa tylko wtedy, gdy z wyprzedzeniem zna się nazwę klasy. Nie zawsze występuje taka sytuacja. Ogólnie stałe klasowe można uzyskiwać dla klas, interfejsów, typów prostych i tablic. (Warto pamiętać, że zapis `<?>` ma związek z typami sparametryzowanymi, które zostaną omówione w rozdziale 14.).

Drugi wiersz, którym warto się zainteresować, dotyczy sposobu pobierania wartości elementów `str` i `val` w momencie ich wyświetlania.

```
System.out.println(anno.str() + " " + anno.val());
```

Zauważ, że jest to zwyczajne wywołanie metody o takiej samej nazwie jak element. Podobne konstrukcje można stosować wszędzie tam, gdzie jest potrzebna wartość składowej adnotacji.

Drugi przykład refleksji

W poprzednim programie metoda `myMeth()` nie miała żadnych parametrów. W wywołaniu `getMethod()` wystarczyło więc przekazać jedynie nazwę metody. Aby pobrać metodę z parametrami, trzeba dodatkowo przekazać do metody `getMethod()` obiekty klas reprezentujące typy parametrów. Oto odrobinę zmieniona wersja wcześniejszego programu.

```

import java.lang.annotation.*;
import java.lang.reflect.*;

@Retention(RetentionPolicy.RUNTIME)
@interface MyAnno {
    String str();
    int val();
}

class Meta {

    // Metoda myMeth ma teraz dwa parametry.
    @MyAnno(str = "Dwa parametry", val = 19)
    public static void myMeth(String str, int i)
    {
        Meta ob = new Meta();

        try {
            Class c = ob.getClass();

```


// Tutaj pojawia się określenie typów parametrów.

```
Method m = c.getMethod("myMeth", String.class, int.class);

MyAnno anno = m.getAnnotation(MyAnno.class);

System.out.println(anno.str() + " " + anno.val());
} catch (NoSuchMethodException exc) {
    System.out.println("Nie znaleziono metody.");
}
}

public static void main(String args[]) {
    myMeth("test", 10);
}
}
```

Wynik działania tej wersji programu jest następujący.

Dwa parametry 19

Metoda `myMeth()` przyjmuje dwa parametry — jeden typu `String` i drugi typu `int`. W celu pobrania informacji na temat metody konieczne jest wywołanie poniższej wersji metody `getMethod()`.

```
Method m = c.getMethod("myMeth", String.class, int.class);
```

Jako dodatkowe argumenty trzeba przekazać obiekty `Class` reprezentujące klasę `String` i typ `int`.

Odczytywanie wszystkich adnotacji

Aby odczytać wszystkie adnotacje skojarzone z danym elementem i stosujące strategie zachowywania `RUNTIME`, należy użyć metody `getAnnotations()` dla tego elementu. Oto ogólna postać metody.

```
Annotation[] getAnnotations()
```

Metoda zwraca tablicę adnotacji. Metodę można wywołać dla obiektów typu `Class`, `Method`, `Constructor` i `Field` oraz kilku innych.

Poniżej pokazano kolejny przykład ilustrujący sposób wykorzystania refleksji do pobrania wszystkich adnotacji związanych z klasą i metodą. Kod deklaruje dwa typy adnotacji. Są one używane do dołączenia dodatkowych informacji do klasy i metody.

// Wyświetlanie wszystkich adnotacji klasy i metody.

```
import java.lang.annotation.*;
import java.lang.reflect.*;

@Retention(RetentionPolicy.RUNTIME)
@interface MyAnno {
    String str();
    int val();
}

@Retention(RetentionPolicy.RUNTIME)
@interface What {
    String description();
}

@What(description = "Klasa testowa adnotacji")
@MyAnno(str = "Meta2", val = 99)
class Meta2 {

    @What(description = "Metoda testowa adnotacji")
    @MyAnno(str = "Testowanie", val = 100)
    public static void myMeth() {
        Meta2 ob = new Meta2();

        try {
            Annotation annos[] = ob.getClass().getAnnotations();
```

```

// Wyświetlenie wszystkich adnotacji dla klasy Meta2.
System.out.println("Wszystkie adnotacje Meta2:");
for(Annotation a : annos)
    System.out.println(a);

System.out.println();

// Wyświetlenie wszystkich adnotacji dla metody myMeth.
Method m = ob.getClass().getMethod("myMeth");
annos = m.getAnnotations();

System.out.println("Wszystkie adnotacje myMeth:");
for(Annotation a : annos)
    System.out.println(a);

} catch (NoSuchMethodException exc) {
    System.out.println("Nie znaleziono metody.");
}
}

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

```

Wynik działania programu jest następujący.

```

Wszystkie adnotacje Meta2:
@What(description=Klasa testowa adnotacji)
@MyAnno(str=Meta2, val=99)

Wszystkie adnotacje myMeth:
@What(description=Metoda testowa adnotacji)
@MyAnno(str=Testowanie, val=100)

```

Program używa metody `getAnnotations()` do pobrania tablicy notatek związanych z klasą `Meta2` i metodą `myMeth()`. Metoda ta zwraca tablicę obiektów `Annotation`. Przypomnę, iż klasa `Annotation` stanowi interfejs bazowy dla wszystkich notatek i przesłania metodę `toString()` klasy `Object`. Z tego powodu próba wyświetlenia notatki powoduje wywołanie metody `toString()`, która zwraca pełny opis notatki.

Interfejs `AnnotatedElement`

Metody `getAnnotation()` i `getAnnotations()` użyte we wcześniejszych przykładach są zdefiniowane w interfejsie `AnnotatedElement` z pakietu `java.lang.reflect`. Interfejs ten wykorzystuje mechanizm refleksji do pobierania adnotacji i jest zaimplementowany przez klasy `Method`, `Field`, `Constructor`, `Class`, `Package` i kilku innych.

Poza wspomnianymi wcześniej metodami `getAnnotation()` oraz `getAnnotations()` interfejs definiuje jeszcze kilka innych metod. Dwie z nich były dostępne już w JDK 5. Pierwszą z nich jest `getDeclaredAnnotations()` o następującej ogólnej postaci.

```
Annotation[] getDeclaredAnnotations()
```

Metoda zwraca wszystkie niedziedziczone adnotacje występujące w wywołującym obiekcie. Druga metoda to `isAnnotationPresent()` o następującej postaci.

```
boolean isAnnotationPresent(Class<? extends Annotation> typAdnotacji)
```

Metoda zwraca wartość `true`, jeśli z wywoływany obiekt jest skojarzona adnotacja typu `typAdnotacji`. W przeciwnym przypadku zwraca wartość `false`. W JDK 8 zostały do nich dodane trzy kolejne metody: `getDeclaredAnnotation()`, `getAnnotationsByType()` oraz `getDeclaredAnnotationsByType()`. Dwie ostatnie metody automatycznie operują na powtarzających się adnotacjach. (Powtarzające się adnotacje zostały opisane pod koniec tego rozdziału).

Wartości domyślne

Składowym adnotacji można nadawać wartości domyślne, które będą stosowane, jeśli odpowiednie wartości nie zostaną określone podczas tworzenia egzemplarza adnotacji. Wartości domyślne dodaje się poprzez dodanie klauzuli `default` do deklaracji składowej. Oto ogólna postać deklaracji.

typ składowa() default wartość;

Wskazywana *wartość* musi być zgodna z typem *typ*.

Oto przykład adnotacji `@MyAnno` z dodanymi wartościami domyślnymi.

// Deklaracja typu adnotacji zawierającego wartości domyślne.

```
@Retention(RetentionPolicy.RUNTIME)
@interface MyAnno {
    String str() default "Testowanie";
    int val() default 9000;
}
```

Składowym `str` i `val` będą przypisywane odpowiednio domyślne wartości "Testowanie" i 9000. Oznacza to, że w momencie stosowania tej adnotacji dla klasy lub metody nie jest wymagane określenie jakichkolwiek parametrów. W razie potrzeby można jednak przypisać poszczególnym składowym wartości inne niż domyślne. Oto cztery możliwe sposoby stosowania adnotacji `@MyAnno`:

@MyAnno() // obie wartości domyślne

@MyAnno(str = "pewien tekst") // domyślne tylko val

@MyAnno(val = 100) // domyślne tylko str

@MyAnno(str = "Testowanie", val = 100) // brak wartości domyślnych

Poniższy program ilustruje wykorzystanie wartości domyślnych adnotacji.

```
import java.lang.annotation.*;
import java.lang.reflect.*;
```

// Deklaracja typu adnotacji zawierającego wartości domyślne.

```
@Retention(RetentionPolicy.RUNTIME)
@interface MyAnno {
    String str() default "Testowanie";
    int val() default 9000;
}
```

```
class Meta3 {
```

// Adnotacja dla metody używa wartości domyślnych.

@MyAnno()

```
public static void myMeth() {
    Meta3 ob = new Meta3();
```

// Pobranie adnotacji dla metody

// i wyświetlenie wartości jej składowych.

```
try {
```

```
    Class c = ob.getClass();
```

```
    Method m = c.getMethod("myMeth");
```

```
    MyAnno anno = m.getAnnotation(MyAnno.class);
```

```
    System.out.println(anno.str() + " " + anno.val());
```

```
} catch (NoSuchMethodException exc) {
```

```
    System.out.println("Nie znaleziono metody.");
```

```
}
```

```
}
```

```
public static void main(String args[]) {
```

```
    myMeth();
```

```
}
```

```
}
```

Program po uruchomieniu wyświetli następujący wynik.

Testowanie 9000

Adnotacje znacznikowe

Adnotacja znacznikowa to specjalna forma adnotacji, która nie zawiera żadnych składowych. Jej jedynym zadaniem jest znakowanie elementu kodu — wystarczy sama jej obecność przy jakimś elemencie. Najlepszym sposobem sprawdzenia istnienia adnotacji znacznikowej jest użycie metody `isAnnotationPresent()` zdefiniowanej w interfejsie `AnnotatedElement`.

Poniższy przykład ilustruje sposób stosowania adnotacji znacznikowych. Ponieważ interfejs adnotacji nie zawiera składowych, wystarczy sprawdzenie samego istnienia adnotacji (lub jej braku).

```
import java.lang.annotation.*;
import java.lang.reflect.*;

// Adnotacja znacznikowa.
@Retention(RetentionPolicy.RUNTIME)
@interface MyMarker { }

class Marker {

    // Użycie znacznika dla metody.
    // Zauważ, iż nie są wymagane nawiasy.
    @MyMarker
    public static void myMeth() {
        Marker ob = new Marker();

        try {
            Method m = ob.getClass().getMethod("myMeth");

            // Sprawdzenie istnienia adnotacji znacznikowej.
            if (m.isAnnotationPresent(MyMarker.class))
                System.out.println("Adnotacja MyMarker istnieje.");

        } catch (NoSuchMethodException exc) {
            System.out.println("Nie znaleziono metody.");
        }
    }

    public static void main(String args[]) {
        myMeth();
    }
}
```

Wynik działania programu potwierdza istnienie adnotacji:

Adnotacja MyMarker istnieje.

Warto zwrócić uwagę na brak konieczności stosowania nawiasów okrągłych po nazwie adnotacji znacznikowej `@MyMarker`. Ponieważ adnotacja nie zawiera żadnych składowych, wystarczy podać samą jej nazwę.

`@myMarker`

Zastosowanie pustych nawiasów nie jest błędem, ale nie są one wymagane.

Adnotacje jednoelementowe

Adnotacje jednoelementowe zawierają tylko jedną składową. Działają dokładnie tak samo jak inne adnotacje, ale dodatkowo umożliwiają stosowanie skróconej formy przypisania wartości jedynej składowej. W momencie stosowania adnotacji jednoelementowej wystarczy użyć samej wartości składowej — nie trzeba podawać jej nazwy. Aby jednak móc skorzystać z takiego skrótu, składowa musi mieć nazwę `value`.

Poniżej znajduje się przykład wykorzystujący adnotację jednoelementową.

```
import java.lang.annotation.*;
import java.lang.reflect.*;

// Adnotacja jednoelementowa.
@Retention(RetentionPolicy.RUNTIME)
@interface MySingle {
    int value(); // składowa musi mieć nazwę value
}

class Single {

    // Oznaczenie metody adnotacją jednoelementową.
    @MySingle(100)
    public static void myMeth() {
        Single ob = new Single();

        try {
            Method m = ob.getClass().getMethod("myMeth");

            MySingle anno = m.getAnnotation(MySingle.class);

            System.out.println(anno.value()); // wyświetla 100
        } catch (NoSuchMethodException exc) {
            System.out.println("Nie znaleziono metody.");
        }
    }

    public static void main(String args[]) {
        myMeth();
    }
}
```

Zgodnie z oczekiwaniami program po uruchomieniu wyświetla wartość 100. Adnotacja użyta dla metody `myMeth()` ma następującą postać.

```
@MySingle(100)
```

Zauważ brak `value =`.

Skróconą składnię można zastosować również wtedy, gdy adnotacja zawiera inne składowe, pod warunkiem że zdefiniowano dla tych składowych wartości domyślne. Poniższy kod deklaruje adnotację z dodatkową składową `xyz` o wartości domyślnej 0.

```
@interface SomeAnno {
    int value();
    int xyz() default 0;
}
```

Wszędzie tam, gdzie ma być stosowana wartość domyślna składowej `xyz`, można użyć poniższej postaci adnotacji. Wystarczy podać wartość składowej `value` zgodnie ze składnią adnotacji jednoelementowej.

```
@SomeAnno(88)
```

W takiej sytuacji `xyz` zostanie ustawione na wartość domyślną 0, natomiast `value` otrzyma wartość 88. Gdy chce się ustawić wartości obu składowych, trzeba jawnie podać obie nazwy.

```
@SomeAnno(value = 88, xyz = 99)
```

Warto pamiętać, że składnia adnotacji jednoelementowej wymaga użycia nazwy `value` dla jedynej składowej.

Wbudowane adnotacje

Java definiuje wiele wbudowanych adnotacji. W większości są to adnotacje wyspecjalizowane, ale dziewięć ma uniwersalny charakter. Cztery z nich znajdują się w pakiecie `java.lang.annotation`: `@Retention`, `@Documented`, `@Target` i `@Inherited`. Pięć pozostałych należy do pakietu `java.lang`: `@Override`, `@Deprecated`, `@FunctionalInterface`, `@SafeVarargs` i `@SuppressWarnings`.



Uwaga

W wydaniu JDK 8 do interfejsu `java.lang.annotation` zostały dodane dwie nowe adnotacje: `Repeatable` oraz `Native`. Pierwsza z nich obsługuje adnotacje, które mogą się powtarzać, opisane pod koniec tego rozdziału. Z kolei druga adnotacja, `Native`, służy do oznaczania pól, do których może się odwoływać kod rdzenny.

Adnotacja `@Retention`

Tę adnotację zaprojektowano z myślą o stosowaniu tylko dla innych adnotacji. Adnotacja `@Retention` określa strategię zachowywania (opisana wcześniej w tym rozdziale).

Adnotacja `@Documented`

Ta adnotacja pełni funkcję interfejsu znaczników informującego narzędzia o konieczności udokumentowania adnotacji. Powinna być stosowana jedynie przy deklaracji innych adnotacji.

Adnotacja `@Target`

Ta adnotacja służy do określania typów elementów, dla których można stosować właściwą adnotację. Powinna być stosowana jedynie przy innych adnotacjach. Adnotacja `@Target` przyjmuje jeden argument, który musi być tablicą stałych typu wyliczeniowego `ElementType`. Argument określa, dla jakich typów deklaracji można używać adnotacji. Znaczenie poszczególnych stałych wyliczeniowych opisano w tabeli 12.1.

Tabela 12.1. Stałe wyliczenia `ElementType`

Stałe adnotacji <code>@Target</code>	Adnotację można stosować dla
<code>ANNOTATION_TYPE</code>	Innej adnotacji
<code>CONSTRUCTOR</code>	Konstruktora
<code>FIELD</code>	Pola
<code>LOCAL_VARIABLE</code>	Zmiennej lokalnej
<code>METHOD</code>	Metody
<code>PACKAGE</code>	Pakietu
<code>PARAMETER</code>	Parametru
<code>TYPE</code>	Klasy, interfejsu lub wyliczenia
<code>TYPE_PARAMETER</code>	Parametr typu (dodana w JDK 8)
<code>TYPE_USE</code>	Zastosowanie typu (dodana w JDK 8)

W adnotacji `@Target` można wskazać więcej niż jedną ze stałych, ale trzeba je oddzielić przecinkami i umieścić w nawiasach klamrowych. Aby wskazać, że dana adnotacja ma zastosowanie tylko dla pól i zmiennych lokalnych, należy użyć adnotacji `@Target` w następującej formie:

```
@Target( { ElementType.FIELD, ElementType.LOCAL_VARIABLE } )
```

Jeśli nie zostanie użyta adnotacja `@Target`, to z wyjątkiem parametrów typów adnotacja może być używana w dowolnych deklaracjach. Z tego względu często bardzo dobrym rozwiązaniem jest jawne określanie, dla jakich elementów można stosować daną adnotację, co w jasny i precyzyjny sposób określi jej zamierzone zastosowanie.

Adnotacja @Inherited

@Inherited to adnotacja znacznikowa, którą można stosować tylko przed deklaracjami innych adnotacji. Co więcej, ta adnotacja wpływa tylko na adnotacje stosowane przed definicjami klas. Adnotacja powoduje dziedziczenie adnotacji klasy bazowej przez jej podklasę. Oznacza to, że w odpowiedzi na żądanie konkretnej, brakującej adnotacji podklasy Java sprawdza, czy odpowiednia adnotacja została zdefiniowana dla klasy bazowej. Jeśli klasa bazowa została oznaczona żadaną adnotacją i jeśli oznaczono tę adnotację adnotacją @Inherited, to właśnie ona zostanie zwrócona.

Adnotacja @Override

Ta adnotacja znacznikowa może być stosowana tylko przed deklaracjami metod. Metoda oznaczona taką adnotacją musi przykrywać odpowiednią metodę klasy bazowej. Jeśli ten warunek nie zostanie spełniony, kompilator zgłosi stosowny błąd. W ten sposób można zagwarantować, że metoda nadklasy rzeczywiście zostanie przykryta, nie przeciążona.

Adnotacja @Deprecated

@Deprecated to adnotacja znacznikowa. Za pomocą tej adnotacji można wskazać przestarzałą deklarację i zasugerować istnienie nowszej, zalecanej formy.

Adnotacja @FunctionalInterface

@FunctionalInterface jest adnotacją znacznikową dodaną w JDK 8 i przeznaczoną do oznaczania interfejsów. Informuje ona, że interfejs, do którego została dodana, jest interfejsem funkcyjnym. Mianem **funkcyjnych** określane są interfejsy, które deklarują jedną i tylko jedną metodę abstrakcyjną. Interfejsy funkcyjne są używane przez wyrażenia lambda. (Więcej informacji o interfejsach funkcyjnych oraz wyrażeniach lambda można znaleźć w rozdziale 15.). Jeśli interfejs, do którego dodano tę adnotację, nie jest interfejsem funkcyjnym, to kompilator zgłosi błąd. Koniecznie należy zrozumieć, że adnotacja ta wcale nie jest niezbędna do tworzenia interfejsów funkcyjnych. Każdy interfejs deklarujący dokładnie jedną metodę abstrakcyjną z definicji jest interfejsem funkcyjnym. Z tego względu adnotacja @FunctionalInterface ma czysto informacyjny charakter.

Adnotacja @SafeVarargs

@SafeVarargs to adnotacja znacznikowa, którą można stosować tylko dla metod i konstruktorów. Adnotacja oznacza, że metoda lub konstruktor nie zawiera żadnych niebezpiecznych działań związanych z parametrem typu vararg. Adnotacja służy do wstrzymywania generowania ostrzeżeń związanych z tworzeniem egzemplarzy nieuprzedmiotowionych (ang. *non-reifiable*) typów vararg i tablic sparametryzowanych. (Typ nieuprzedmiotowiony to w istocie typ sparametryzowany; typy sparametryzowane zostaną omówione w rozdziale 14.). Adnotację @SafeVarargs można stosować tylko dla statycznych i finalnych metod i konstruktorów otrzymujących parametry typu vararg. Adnotację wprowadzono w wydaniu JDK 7.

Adnotacja @SuppressWarnings

Ta adnotacja wymusza wstrzymanie generowania określonych ostrzeżeń kompilatora. W adnotacji należy podać nazwę wstrzymwanego ostrzeżenia w formie łańcucha. Adnotacje tego typu można stosować dla dowolnych deklaracji.

Adnotacje typów

Zaczynając od wydania JDK 8, zwiększyła się także liczba miejsc w kodzie, w których można umieszczać adnotacje. Jak już wcześniej wspominałem, początkowo można je było dodawać wyłącznie do deklaracji. Jednak w JDK 8 adnotacje można w większości przypadków umieszczać także w miejscach, gdzie są podawane typy. Ten rozszerzony aspekt adnotacji jest nazywany **adnotacjami typów**. Na przykład adnotację można dodać do typu wyniku zwracanego przez metodę, do typu referencji this

wewnątrz metody, do rzutowania, do poszczególnych poziomów tablic, do dziedziczonych klas oraz do klauzuli `throws`. Adnotacje można także dodawać do typów uogólnionych, w tym także do ograniczeń parametrów typów oraz argumentów typów ogólnych. (Szczegółowe omówienie zagadnień typów uogólnionych można znaleźć w rozdziale 14.).

Adnotacje typów są ważne, gdyż pozwalają wykonywać dodatkowe testy, chroniące przed występowaniem błędów. Trzeba wiedzieć, że kompilator `javac` z założenia sam nie przeprowadza tych testów. Do tego celu służy odrębne narzędzie, choć może ono działać jako wtyczka kompilatora.

Adnotacje typów można dodawać do elementów określanych jako zastosowanie typu (`Element` ↪ `Type.TYPE_USE`). (Warto sobie przypomnieć, że ważne elementy, dla których można stosować adnotacje, są określane przy użyciu opisanej wcześniej adnotacji `@Target`). Adnotacja typu odnosi się do typu, przed którym została ona podana. Na przykład przy założeniu, że dostępna jest adnotacja typu o nazwie `@TypeAnno`, poniższa definicja będzie prawidłowa:

```
void myMeth() throws @TypeAnno NullPointerException { //...
```

W tym przypadku adnotacja `@TypeAnno` została dodana do typu `NullPointerException` w klauzuli `throws`.

Istnieje także możliwość dodania adnotacji do typu referencji `this` (nazywanej **odbiorcą**). Jak wiadomo, `this` jest niejawnym argumentem wszystkich metod instancyjnych, które odwołują się do wywoływanego obiektu. Aby dodać adnotację do typu `this`, konieczne jest zastosowanie kolejnej nowej możliwości, dodanej w JDK 8. Otóż zaczynając do JDK 8, istnieje możliwość jawnego deklarowania `this` jako pierwszego parametru metody. W takiej deklaracji typ `this` musi odpowiadać klasie, do której należy metoda; oto przykład:

```
class SomeClass {
    int myMeth(SomeClass this, int i, int j) { //...
```

Ponieważ metoda `myMeth()` jest definiowana w klasie `SomeClass`, zatem typem `this` musi być `SomeClass`. W przypadku zastosowania takiej deklaracji zyskujemy możliwość dodania adnotacji do typu `this`. Jeśli ponownie założymy, że dostępna jest adnotacja typu o nazwie `@TypeAnno`, to poniższy wiersz kodu będzie prawidłowy:

```
int myMeth(@TypeAnno SomeClass this, int i, int j) { //...
```

Warto zwrócić uwagę, że nie trzeba jawnie deklarować referencji `this`, chyba że chcemy dodać do niej adnotację. (Nawet jeśli `this` nie zostanie jawnie zadeklarowana jako parametr, to i tak będzie ona przekazywana do metody. Pod tym względem w JDK 8 *nic nie uległo zmianie*). Poza tym jawne deklarowanie `this` w żaden sposób nie zmienia sygnatury metody, gdyż i tak parametr ten jest domyślnie deklarowany, choć niejawnie. Warto zatem powtórzyć, że parametr `this` będziemy deklarować wyłącznie w sytuacjach, kiedy będziemy chcieli dodać do niego adnotację typu. Jeśli jednak `this` zostanie zadeklarowany, to *musi* on być pierwszym parametrem.

Zamieszczony poniżej program przykładowy przedstawia kilka miejsc, w których można stosować adnotacje typów. Definiuje on kilka adnotacji, z których parę jest adnotacjami typów. Nazwy adnotacji, jak również elementy, dla których można je stosować, zostały przedstawione w tabeli 12.2.

Tabela 12.2. Zastosowane w przykładzie adnotacje oraz elementy, dla których można je stosować

Nazwa adnotacji	Określenie elementu, dla którego można ją stosować:
<code>@TypeAnno</code>	<code>ElementType.TYPE_USE</code>
<code>@MaxLen</code>	<code>ElementType.TYPE_USE</code>
<code>@NotZeroLen</code>	<code>ElementType.TYPE_USE</code>
<code>@Unique</code>	<code>ElementType.TYPE_USE</code>
<code>@What</code>	<code>ElementType.TYPE_PARAMETER</code>
<code>@EmptyOK</code>	<code>ElementType.FIELD</code>
<code>@Recommended</code>	<code>ElementType.METHOD</code>

Warto zwrócić uwagę, że @EmptyOK, @Recommended oraz @What nie są adnotacjami typów. Zostały one dołączone do programu w celach porównawczych. Spośród nich szczególnie interesująca jest adnotacja @What, która jest przeznaczona do dodawania do deklaracji parametru typu. To kolejna nowa możliwość stosowania adnotacji wprowadzona w JDK 8. Komentarze zamieszczone w kodzie programu wyjaśniają zastosowanie każdej z adnotacji.

```
// Prezentacja kilku adnotacji typów.
import java.lang.annotation.*;
import java.lang.reflect.*;

// Adnotacja znacznikowa, którą można dodawać do typów.
@Target(ElementType.TYPE_USE)
@interface TypeAnno { }

// Inna adnotacja znacznikowa, którą można dodawać do typów.
@Target(ElementType.TYPE_USE)
@interface NotZeroLen { }

// Jeszcze inna adnotacja znacznikowa, którą można dodawać
// do typów.
@Target( ElementType.TYPE_USE )
@interface Unique { }

// Adnotacja sparametryzowana, którą można dodawać do typów.
@Target(ElementType.TYPE_USE)
@interface MaxLen {
    int value();
}

// Adnotacja, którą można dodawać do parametru typu.
@Target(ElementType.TYPE_PARAMETER)
@interface What {
    String description();
}

// Adnotacja, którą można dodawać do deklaracji pola.
@Target(ElementType.FIELD)
@interface EmptyOK { }

// Adnotacja, którą można dodawać do deklaracji metody.
@Target(ElementType.METHOD)
@interface Recommended { }

// Zastosowanie adnotacji dla parametru typu
class TypeAnnoDemo<@What(description = "Ogólny typ danych") T> {

    // Zastosowanie adnotacji typu dla konstruktora.
    public @Unique TypeAnnoDemo() {}

    // Adnotacja typu (w tym przypadku typu String), a nie pola.
    @TypeAnno String str;

    // A to jest adnotacja pola test.
    @EmptyOK String test;

    // Zastosowanie adnotacji typu dla this (odbiorycy).
    public int f(@TypeAnno TypeAnnoDemo<T> this, int x) {
        return 10;
    }

    // Adnotacja dodana do typu wyniku zwracanego przez metodę.
    public @TypeAnno Integer f2(int j, int k) {
```

```

    return j+k;
}

// Adnotacja dodana do deklaracji metody.
public @Recommended Integer f3(String str) {
    return str.length() / 2;
}

// Zastosowanie adnotacji typu w klauzuli throws.
public void f4() throws @TypeAnno NullPointerException {
    // ...
}

// Adnotacje zastosowane do różnych poziomów tablicy.
String @MaxLen(10) [] @NotZeroLen [] w;

// Adnotacja zastosowana dla typu elementów tablicy.
@TypeAnno Integer[] vec;

public static void myMeth(int i) {

    // Zastosowanie adnotacji typu dla argumentu typu.
    TypeAnnoDemo<@TypeAnno Integer> ob =
        new TypeAnnoDemo<@TypeAnno Integer>();

    // Zastosowanie adnotacji typu dla new.
    @Unique TypeAnnoDemo<Integer> ob2 = new @Unique TypeAnnoDemo<Integer>();

    Object x = new Integer(10);
    Integer y;

    // Zastosowanie adnotacji typu dla rzutowania.
    y = (@TypeAnno Integer) x;
}

public static void main(String args[]) {
    myMeth(10);
}

// Zastosowanie adnotacji typu w deklaracji klasy.
class SomeClass extends @TypeAnno TypeAnnoDemo<Boolean> {}
}

```

Choć w większości przypadków element, do którego odnoszą się adnotacje, w tym przykładzie jest oczywisty, to jednak cztery zastosowania wymagają dokładniejszego omówienia. Pierwszym z nich jest adnotacja typu wyniku zwracanego przez metodę oraz adnotacja deklaracji metody. W powyższym programie należy zatem zwrócić szczególną uwagę na dwie przedstawione poniżej deklaracje metod:

```

// Adnotacja dodana do typu wyniku zwracanego przez metodę.
public @TypeAnno Integer f2(int j, int k) {
    return j+k;
}

// Adnotacja dodana do deklaracji metody.
public @Recommended Integer f3(String str) {
    return str.length() / 2;
}

```

W tych deklaracjach należy zwrócić uwagę na to, że adnotacja jest umieszczona przed typem zwracanego wyniku (którym w obu metodach jest `Integer`). Jednak w każdej z tych metod adnotacja została dodana do innego elementu. W pierwszym przypadku adnotacja `@TypeAnno` dotyczy typu wyniku zwracanego przez metodę `f2()`. Wynika to z faktu, że typ elementu, dla którego można stosować tę adnotację, został określony jako `ElementType.TYPE_USE`, co oznacza, że można jej używać

w miejscach zastosowania typów. W drugim przypadku adnotacja `@Recommended` dotyczy samej deklaracji metody. Wynika to z faktu, że typ elementu, dla którego można stosować tę adnotację, został określony jako `ElementType.METHOD`. W efekcie adnotacja `@Recommended` odnosi się do deklaracji metody, a nie typu jej wyniku. Jak widać, określenie elementu, dla którego można stosować adnotację, pozwala wyeliminować pozorną niejednoznaczność pomiędzy adnotacją deklaracji metody oraz adnotacją typu wyniku zwracanego przez metodę.

I jeszcze jedna uwaga dotycząca adnotacji typu wyniku zwracanego przez metodę: jeśli tym typem jest `void`, to nie będzie można dodać do niego adnotacji.

Kolejnym interesującym aspektem przedstawionym w powyższym programie przykładowym są adnotacje pól. Ich przykłady pokazano ponownie w poniższym fragmencie kodu:

```
// Adnotacja typu (w tym przypadku typu String), a nie pola.
@TypeAnno String str;
```

```
// A to jest adnotacja pola test.
@EmptyOK String test;
```

W tym przypadku adnotacja `@TypeAnno` dotyczy typu `String`, natomiast adnotacja `@EmptyOK` — pola `test`. Choć każda z ich jest umieszczona na samym początku deklaracji, to elementy, których dotyczą te adnotacje, są inne i zależą od typu elementów, dla których dane adnotacje można stosować. Jeśli typ elementów, dla których można stosować adnotację, został określony jako `ElementType.TYPE_USE`, to adnotacja dotyczy typu. Jeśli natomiast typ elementów, dla których można stosować adnotację, został określony jako `ElementType.FIELD`, to adnotacja dotyczy pola. A zatem sytuacja jest dosyć podobna do tej, która występowała wcześniej w przypadku metod, więc nie ma mowy o żadnych niejednoznacznościach. Dokładnie ten sam mechanizm pozwala uniknąć niejednoznaczności w razie dodawania adnotacji do zmiennych lokalnych.

Trzeba także zwrócić uwagę na sposób, w jaki dodano adnotację do referencji `this` (odbiorcy):

```
public int f(@TypeAnno TypeAnnoDemo<T> this, int x) {
```

Jak widać, `this` jest pierwszym parametrem metody `f()`, a jego typ został określony jako `TypeAnnoDemo` (czyli odpowiada klasie, której składową jest metoda `f()`). Jak już zaznaczyłem, zaczynając od JDK 8, istnieje możliwość jawnego podawania w deklaracjach parametru `this`, właśnie po to, by można było do niego dodawać adnotacje typu.

I w końcu adnotacje poziomów tablicy zastosowane w poniższej instrukcji:

```
String @MaxLen(10) [] @NotZeroLen [] w;
```

W tej deklaracji adnotacja `@MaxLen` dotyczy typu pierwszego poziomu tablicy, a adnotacja `@NotZeroLen` — typu jej drugiego poziomu. Z kolei w poniższej deklaracji:

```
@TypeAnno Integer[] vec;
```

adnotacja została dodana do typu elementów tablicy, którym jest `Integer`.

Adnotacje powtarzalne

Kolejną nowością związaną z adnotacjami wprowadzoną w JDK 8 jest możliwość powtarzania adnotacji odnoszących się do tego samego elementu. Są one nazywane **adnotacjami powtarzalnymi**. Aby zapewnić możliwość powtarzania konkretnej adnotacji, należy do niej dodać adnotację `@Repeatable`, zdefiniowaną w pakiecie `java.lang.annotation`. Jej składowa `value` określa typ **kontenera** dla adnotacji powtarzalnej. Kontener jest określany jako adnotacja, której składowa `value` jest tablicą, przy czym jej typ odpowiada typowi adnotacji powtarzalnej. Innymi słowy, aby utworzyć aplikację powtarzalną, należy utworzyć adnotację kontenera, a następnie zastosować typ tej adnotacji jako argument adnotacji `@Repeatable`.

Aby uzyskać dostęp do adnotacji powtarzalnych na przykład za pomocą metody `getAnnotation()`, należy skorzystać z adnotacji kontenera, a nie z samej adnotacji powtarzalnej. Poniższy program przedstawia takie rozwiązanie — zmienia on przedstawioną wcześniej adnotację `MyAnno` w adnotację powtarzalną i pokazuje, jak można jej używać.

```
// Prezentacja adnotacji powtarzalnych.

import java.lang.annotation.*;
import java.lang.reflect.*;

// Zapewnia możliwość powtarzania adnotacji MyAnno.
@Retention(RetentionPolicy.RUNTIME)
@Repeatable(MyRepeatedAnnos.class)
@interface MyAnno {
    String str() default "Testujemy";
    int val() default 9000;
}

// To jest adnotacja kontenera.
@Retention(RetentionPolicy.RUNTIME)
@interface MyRepeatedAnnos {
    MyAnno[] value();
}

class RepeatAnno {

    // Do myMeth() dodawane są dwie adnotacje MyAnno.
    @MyAnno(str = "Pierwsza adnotacja", val = -1)
    @MyAnno(str = "Druga adnotacja", val = 100)
    public static void myMeth(String str, int i)
    {
        RepeatAnno ob = new RepeatAnno();

        try {
            Class<?> c = ob.getClass();

            // Pobiera adnotacje dotyczące metody myMeth().
            Method m = c.getMethod("myMeth", String.class, int.class);

            // Wyświetla powtórzone adnotacje MyAnno.
            Annotation anno = m.getAnnotation(MyRepeatedAnnos.class);
            System.out.println(anno);

        } catch (NoSuchMethodException exc) {
            System.out.println("Nie znaleziono metody.");
        }
    }

    public static void main(String args[]) {
        myMeth("test", 10);
    }
}

```

Ten program generuje następujące wyniki:

```
@MyRepeatedAnnos(value=@MyAnno(val=-1, str=Pierwsza adnotacja), @MyAnno(val=100, str=Druga adnotacja))
```

Zgodnie z wcześniejszymi wyjaśnieniami, aby adnotacja `MyAnno` mogła być powtarzana, należy do niej dodać adnotację `@Repeatable` określającą adnotację kontenera. W tym przykładzie adnotacją kontenera jest `MyRepeatedAnnos`. Program pobiera adnotacje powtarzalne, wywołując metodę `getAnnotation()` i przekazując do niej klasę adnotacji kontenera, a nie klasę adnotacji powtarzalnych. Jak pokazują wyniki działania programu, adnotacje powtarzalne są od siebie oddzielane przecinkami, a nie zwracane pojedynczo.

Innym sposobem pobierania adnotacji powtarzalnych jest korzystanie z jednej z nowych metod dodanych w JDK 8 do interfejsu `AnnotatedElement` i pozwalających na bezpośrednie operowanie na adnotacjach powtarzalnych. Chodzi konkretnie o dwie metody: `getAnnotationsByType()` oraz `getDeclaredAnnotationsByType()`. Przyjrzyjmy się pierwszej z nich:

```
<T extends Annotation> T[] getAnnotationsByType(Class<T> annoType)
```

Metoda ta zwraca tablicę adnotacji typu `annoType`, dodanych do obiektu, na rzecz którego metoda została wywołana. Jeśli takich adnotacji nie ma, metoda zwróci pustą tablicę. Poniżej przedstawiony został krótki przykład użycia metody `getAnnotationsByType()` w celu pobrania adnotacji typu `MyAnno`:

```
Annotation[] annos = m.getAnnotationsByType(MyAnno.class);
for(Annotation a : annos)
    System.out.println(a);
```

Jak widać, w wywołaniu metody `getAnnotationsByType()` przekazywany jest typ adnotacji przetwarzanej. Tablica zwrócona przez tę metodę zawiera wszystkie egzemplarze adnotacji `MyAnno` dodane do metody `myMeth()`; w tym przykładzie są dwie takie adnotacje. Do każdej ze zwróconych adnotacji można odwołać się przy użyciu indeksu tablicy. Powyższy przykład wyświetla te adnotacje, korzystając z wersji `for-each` pętli `for`.

Ograniczenia

Ze stosowaniem deklaracji adnotacji wiąże się kilka ograniczeń. Po pierwsze, żadna adnotacja nie może dziedziczyć po innej adnotacji. Po drugie, wszystkie metody deklarowane w adnotacji muszą być bezparametrowe. Co więcej, metody adnotacji mogą zwracać tylko wartości następujących typów:

- typ prosty, na przykład `int` lub `double`;
- obiekt typu `String` lub `Class`;
- typ wyliczeniowy;
- typ innej adnotacji;
- tablicę jednego z wymienionych wcześniej typów.

Adnotacje nie mogą korzystać z typów sparametryzowanych (ich dokładne omówienie znajduje się w rozdziale 14.). Nie mogą też zawierać klauzuli `throws`.

Skorowidz

A

Abstract Window Toolkit,

Patrz: AWT

abstrakcja, 46

adnotacja, 265, 444, 631

@Deprecated, 274, 275

@Documented, 274

@EmptyOK, 276, 277

@FunctionalInterface, 274, 275

@Inherited, 274, 275

@MaxLen, 276

@Native, 274

@NotZeroLen, 276

@Override, 274, 275

@Recommended, 276, 277

@Repeatable, 274, 279, 280

@Retention, 274

@SafeVarargs, 274, 275

@SuppressWarnings, 274, 275

@Target, 274

@TypeAnno, 276

@Unique, 276

@What, 276, 277

deklaracji, 265

deklaracji metody, 278

dziedziczenie, 281

jednoelementowa, 272

odczytywanie, 266, 269

pola, 279

pomijanie, 266

powtarzalna, 279, 280

strategia zachowywania, 266
typu, 265, 275, 276, 278

referencji this, 276

tworzenie, 265

wartość domyślna, 271

wbudowana, 274

znacznikowa, 272

adres

internetowy, *Patrz:* domena

IP, 659, 894

URI, 1087

URL, 39, 652, 658, 665, 679, 680, 1077, 1078, 1087, 1088

składniki, 653

akcelerator, 957, 970, 1041, 1049

akcja, 967, 968

algorytm, 448, 490

round-robin, 227

analizator leksykalny, 513

API, 313, 349, 567

dat i czasu, 897

strumieni, 42, 855, 856, 877

aplet, 35, 36, 37, 52, 124, 290,

298, 300, 649, 663, 664, 710, 880

architektura, 667

budowa, 667, 668, 670

dane wejściowe, 679

dane wyjściowe, 671, 672, 681, 713

definiowane, 298

kolor tła, 670

kontekst graficzny, 672

lokalny, 665

na bazie biblioteki Swing, 917

podpis cyfrowy, 299, 665
przeglądarka,

Patrz: appletviewer

uruchamianie, 299

wdrażanie, 664

zniszczenie, 665

aplikacja

JavaFX, 984

klient-serwer, 891, 893

na bazie biblioteki Swing, 910
uruchomieniowa, 51

appletviewer, 664, 710

architektura

model-delegacja, 908

z odrębnym modelem,

Patrz: architektura

model-delegacja

architektura model-widok-
kontroler, *Patrz:* MVC

argument

wiersza poleceń, 159

wyrażenie lambda, 357

zmienna liczba,

Patrz: varargs

ARM, 295, 443

asercja, 306, 308

automatic resource

management, *Patrz:* ARM

AWT, 298, 663, 683, 685, 709, 712, 741, 905

komponent ciężki, 906

B

basic multilingual plane,
Patrz: BMP

baza danych, 510, 1077

bezpieczeństwo, 36, 37, 46, 439,
449, 880, 1077

biblioteka, 42
API, *Patrz:* API
AWT, *Patrz:* AWT
klas, 60
Swing, 663, 685, 709, 741,
905, 906
GUI, 908
hierarchia zawierania,
Patrz: hierarchia
zawierania
menu, *Patrz:* menu
PLAF, *Patrz:* PLAF
rysowanie, 919, 920
tworzenie, 182

blok
finally, 209, 218, 576
kodu, *Patrz:* kod blok
try, 209, 211, 216, 218

blokada, 835, 836
StampedLock, 837
wielobieźna, 836

BMP, 413

branch node, *Patrz:* węzeł gałęzi

bufor, 612
bajtowy, 640
ByteBuffer, 621
ograniczenie, 612
pojemność, 612
pozycja, 612
tworzenie, 622

C

ciało
blokowe, 354
wyrażeniowe, 354

closure, *Patrz:* domknięcie

code point, *Patrz:* punkt
kodowy

Common Gateway Interface,
Patrz: interfejs CGI

compact profile, *Patrz:* profil
kompaktowy

concurrency utilities, *Patrz:*
programowanie współbieżne
narzędzia

czas, 447, 519, 521, 524, 654,
879, 881, 896, 897, 898
formatowanie, 541, 896,
899, 900
Greenwich, 525
letni, 525
strefa, 525, 897
uniwersalny, 525
zimowy, 525

czcionka, 711, 729, 734, 995
dostępna, 730
rodzina, 729
tworzenie, 731

D

dane
konwersja formatu, 536, 537
sterujące dostępem do
kodu, 45
wejściowe, 551, 569
buforowanie, 600
odczyt, 549
wzorzec, 558
wyjściowe, 569
wyrównywanie, 545

data, 447, 519, 521, 524, 654,
879, 881, 894, 896, 897, 898
formatowanie, 540, 896,
899, 900
kalendarz gregoriański, 524,
525

datagram, 646, 659

deadlock, *Patrz:* zakleszczenie

default method, *Patrz:* metoda
domyślna

dekoder, 614

delegacja
interfejsu użytkownika, 908

demon, *Patrz:* wątek demon

deserializacja, 605, 606, 608, 893

divide-and-conquer, *Patrz:*
strategia dziel i zwyciężaj

dokończenie, *Patrz:* metoda
finalize

domena, 646, 647, 1089

domknięcie, 349

drukowanie, 880

drzewo, 448, 464, 943, 1029
katalogów, 636
ścieżka, 1029

dyrektywa #define, 201

dziecko, 1029

dziedziczenie, 46, 47, 49, 165,
167, 168, 180, 184
blokowanie, 184
hierarchia, 166, 176, 180
wielopoziomowa, 173
po interfejsie, 202
wielokrotne, 206

dziennik, 567

E

edytor tekstu, 288

egzekutor, 812, 830

encapsulation, *Patrz:*
hermetyzacja

enumeration, *Patrz:* wyliczenie

etykieta, 118, 119, 742, 743, 923,
988
ekranowa, 959, 964, 1038
ikona, 924
obraz, 1001
tekst, 924

event filter, *Patrz:* zdarzenie filtr

extension method, *Patrz:*
metoda rozszerzenia

F

finalizacja, *Patrz:* metoda
finalize

flat map, *Patrz:* mapa płaska

format
GIF, 787
JPEG, 787
PNG, 788

framework
Collections, 112, 116, 260,
315, 367, 447, 448, 449,
450, 490, 813
Fork/Join, 42, 226, 811, 812,
830, 839, 843, 845, 852
GUI, *Patrz:* GUI
NIO, *Patrz:* NIO

Frank Ed, 33

functional interface,
Patrz: interfejs funkcyjny

funkcja, 47
 mieszająca, 462
 trygonometryczna, 64, 429
 hiperboliczna, 430
 odwrotna, 429
 wykładnicza, 430
 zaokrąglenia, 430, 431

G

Gausa rozkład, 529
 Glassfish, 1078
 głodzenie zasobów, 576
 gniazdo, 645, 658
 Berkeley, 645
 numerowane,
Patrz: port protokołu
 TCP/IP, 649
 Gosling James, 33
 Graphical User Interface,
Patrz: GUI
 grupa dyskusyjna, 646
 GUI, 298, 663, 683, 741, 880,
 905, 1071
 biblioteki Swing, 908
 tworzenie, 709

H

hasło, 759
 hermetyzacja, 46, 47, 49, 123,
 137, 149, 170
 hierarchia zawierania, 908
 HotSpot, 37
 HSB, 726

I

identyfikator, 58
 URI, 658
 import statyczny, 309, 310
 inheritance,
Patrz: dziedziczenie
 instrukcja
 break, 102, 103, 113, 114,
 116, 117, 118
 z etykietą, 118, 119
 catch, 211
 continue, 116, 120
 for, *Patrz:* pętla for
 goto, 118

if, 54, 57, 99, 148
 zagnieżdżona, 100
 if-else, 105
 if-else-if, 101, 102
 if-then-else, 97
 import, 192, 193
 import static, 310
 iteracyjna, 99, *Patrz:* pętla
 package, 187, 188
 pusta, 106
 return, 116, 121
 skoku, 99, 116
 sterująca, 54, 99
 switch, 99, 102, 103, 118, 252
 wyrażenie sterujące, 102,
 104
 zagnieżdżona, 105
 synchronized, 239
 throw, 216
 try, 211, 212
 zagnieżdżanie, 212, 214,
 215
 try-with-resources, 210, 296,
 297, 298, 444, 549, 575,
 577, 579, 613, 622, 651
 wyboru, 99
 interfejs, 187, 194, 198, 254
 Action, 967
 ActionListener, 696, 754,
 915, 932, 956
 AdjustmentListener, 696,
 756
 AnnotatedElement, 270,
 280, 445
 Annotation, 265, 444
 Appendable, 399, 442, 590,
 602
 AppletContext, 663, 679, 680
 AppletInitializer, 1071
 AppletStub, 663, 681
 AudioClip, 663, 681
 AutoCloseable, 291, 399,
 443, 549, 555, 575, 578,
 579, 581, 590, 593, 602,
 613, 650
 autor, 1100
 BaseStream, 856
 BasicFileAttributes, 619, 620
 BeanInfo, 1068, 1070, 1071
 BiConsumer, 564
 BiFunction, 564

BinaryOperator, 373, 564
 BiPredicate, 564
 BooleanSupplier, 565
 ButtonModel, 927
 Callable, 813, 832, 851
 CGI, 37, 1077
 ChangeListener, 1011
 Channel, 613
 CharSequence, 399, 442
 ClassDefinition, 444
 ClassFileTransformer, 444
 Cloneable, 399, 425, 515
 Closeable, 570, 576, 578,
 579, 581, 590, 593, 602,
 613
 Collection, 448, 450, 451,
 452, 858
 Collector, 870
 Comparable, 332, 399, 442,
 573, 615
 Comparator, 448, 450, 483,
 484, 485
 ComponentListener, 696,
 697
 Concurrent API, 811, 812,
 813, 835
 Consumer, 373, 471, 565
 ContainerListener, 696, 697
 ContentHandlerFactory,
 647
 CookiePolicy, 647
 CookieStore, 647
 Customizer, 1071, 1072
 DatagramSocketImplFactory,
 647
 DataInput, 570, 594
 DataOutput, 570, 593, 594
 deklarowanie, 194, 195
 zmienna, 195
 Deque, 448, 450, 455, 457
 DesignMode, 1071
 DosFileAttributes, 619
 DoubleBinaryOperator, 565
 DoubleConsumer, 565
 DoubleFunction, 565
 DoublePredicate, 565
 DoubleStream, 858
 DoubleSupplier, 565
 DoubleToIntFunction, 565
 DoubleToLongFunction, 565
 DoubleUnaryOperator, 565

- instrukcja
 - dziedziczenie,
 - Patrz:* dziedziczenie po interfejsie
 - Enumeration, 448, 499, 500
 - EventHandler, 990
 - EventListener, 448
 - ExceptionListener, 1071
 - Executor, 812, 830
 - ExecutorService, 830
 - Externalizable, 570, 606
 - FileFilter, 570
 - FilenameFilter, 570, 574, 575
 - FileNameMap, 647
 - FileVisitor, 636
 - Flushable, 570, 576, 579, 581, 590, 593, 602
 - FocusListener, 696, 697
 - Formattable, 448
 - Function, 373, 484
 - funkcyjny, 42, 275, 350, 351, 564, 747, 881
 - predefiniowany, 372
 - referencja typu, 351, 353
 - sparametryzowany, 356
 - Future, 813, 832
 - HttpServletRequest, 1086
 - HttpServletResponse, 1086, 1087, 1089
 - HttpSession, 1086, 1087, 1088
 - ImageConsumer, 797, 806
 - ImageObserver, 790
 - ImageProducer, 796, 800
 - implementacja, 195, 196
 - częściowa, 197
 - domyślna, 42, 195, 203
 - Instrumentation, 444
 - IntBinaryOperator, 565
 - IntConsumer, 565
 - IntFunction, 565
 - IntPredicate, 565
 - IntStream, 858
 - IntSupplier, 565
 - IntToDoubleFunction, 565
 - IntToLongFunction, 565
 - IntUnaryOperator, 565
 - ItemListener, 696, 697, 747, 748, 751, 929
 - ItemSelectable, 690
 - Iterable, 399, 443, 450, 451, 474, 615
 - Iterator, 448, 450, 466, 467, 873
 - java.io.Externalizable, 1071
 - java.io.Serializable, 1071
 - KeyListener, 696, 697
 - kolekcji, 450
 - LayoutManager, 762
 - List, 448, 450, 453, 458, 461, 500
 - ListIterator, 448, 450, 466
 - ListSelectionListener, 938
 - ListSelectionModel, 946
 - Lock, 813, 836
 - LongBinaryOperator, 566
 - LongConsumer, 566
 - LongFunction, 566
 - LongPredicate, 566
 - LongStream, 858
 - LongSupplier, 566
 - LongToDoubleFunction, 566
 - LongToIntFunction, 566
 - LongUnaryOperator, 566
 - Map, 448, 474, 475, 480, 504
 - Map.Entry, 448, 474, 477, 479
 - Member, 445, 888
 - MouseListener, 951
 - metoda, *Patrz:* metoda interfejsu
 - MouseListener, 696, 697, 699, 701, 962
 - MouseMotionListener, 696, 698, 699, 701, 704
 - MouseWheelListener, 696, 698, 699
 - MutableTreeNode, 943
 - nasłuchujący, 696
 - implementacja metod, 704
 - NavigableMap, 448, 474, 477, 478, 481
 - NavigableSet, 448, 450, 455, 464
 - ObjDoubleConsumer, 566
 - ObjectInput, 570
 - ObjectInputValidation, 570
 - ObjectOutput, 570, 606, 607, 608
 - ObjectStreamConstants, 570
 - ObjIntConsumer, 566
 - ObjLongConsumer, 566
 - ObservableList, 988, 1017, 1021, 1022, 1043
 - Observer, 448, 530, 531
 - Path, 570, 615, 631
 - PluginFilter, 804
 - PosixFileAttributes, 619
 - Predicate, 373, 566
 - PrimitiveIterator, 448, 449
 - PrimitiveIterator.OfDouble, 448, 449
 - PrimitiveIterator.OfInt, 448
 - PrimitiveIterator.OfLong, 448
 - PropertyChangeListener, 1070, 1071, 1072
 - PropertyEditor, 1071
 - ProtocolFamily, 647
 - Queue, 448, 450, 455, 461, 465
 - RandomAccess, 448, 450, 474
 - Readable, 399, 443, 549
 - ReadableByteChannel, 549
 - Runnable, 228, 350, 399, 433, 533, 811, 851
 - implementacja, 230
 - ScheduledExecutorService, 830
 - SeekableByteChannel, 622, 626
 - Serializable, 570, 606, 852
 - Servlet, 1081, 1082
 - ServletConfig, 1081, 1082
 - ServletContext, 1081, 1083
 - ServletRequest, 1081, 1083, 1084, 1085
 - ServletResponse, 1081, 1083, 1084
 - Set, 448, 450, 454, 462, 464, 466
 - składowy, *Patrz:* interfejs zagnieżdżony
 - SocketImplFactory, 647
 - SocketOption, 647
 - SocketOptions, 647
 - SortedList, 450
 - SortedMap, 448, 474, 477
 - SortedSet, 448, 454, 455, 464
 - sparametryzowany, 315, 333, 334

Spliterator, 448, 450, 468,
 470, 472, 874
 Spliterator.OfDouble, 448,,
 473
 Spliterator.OfInt, 448,, 473
 Spliterator.OfLong, 448, 473
 Spliterator.OfPrimitive, 448,
 473
 Stream, 856
 Supplier, 373, 566, 872
 TableColumnModel, 946
 TableModel, 946
 TextListener, 696, 698
 Thread.UncaughtException
 Handler, 399, 444
 ToDoubleBiFunction, 566
 ToDoubleFunction, 566,
 869
 Toggle, 1005, 1008
 ToIntBiFunction, 566
 ToIntFunction, 566
 ToLongBiFunction, 566
 ToLongFunction, 566
 TreeExpansionListener, 943
 TreeModelListener, 943
 TreeNode, 943
 TreeSelectionEvent, 943
 TreeSelectionListener, 943
 Type, 445
 UnaryOperator, 373, 566
 URLStreamHandlerFactory,
 647
 użytkownika
 delegacja, *Patrz:* delegacja
 interfejsu użytkownika
 graficzny, 43, 60, *Patrz:*
 GUI
 VetoableChangeListener,
 1070, 1071, 1072
 Visibility, 1071
 Watchable, 615
 WindowFocusListener, 696,
 698
 WindowListener, 696, 698,
 714
 zagnieżdżony, 197
 zmienna, *Patrz:* zmienna
 interfejsu
 referencyjna, *Patrz:*
 zmienna referencyjna
 interfejsu

Internet Protocol,
Patrz: protokół IP
 interpreter kodu bajtowego, 36
 introspekcja, 1068, *Patrz:*
 refleksja
 iterator, 448, 450, 451, 466, 468,
 491, 502, 873
 mapy, 474
 PrimitiveIterator, 873
 PrimitiveIterator.OfDouble,
 873
 PrimitiveIterator.OfInt, 873
 PrimitiveIterator.OfLong,
 873
 tryAdvance, 874

J

Java, 33, 38
 maszyna wirtualna,
Patrz: JVM
 numer wydania, 41
 wersja, 40, 41, 42, 50
 Java Bean, 1067
 Java Beans, 888, 1067, 1071
 Java Native Interface, *Patrz:* JNI
 Java Network Launch Protocol,
Patrz: protokół JNLP
 java.rmi.server, 880
 JavaFX, 709, 741, 905, 981, 1064
 efekt, 1033, 1036, 1064
 Bloom, 1034
 BoxBlur, 1034
 DropShadow, 1034
 Glow, 1034
 InnerShadow, 1034
 Lighting, 1034
 Reflection, 1034
 kontrolka, *Patrz:* kontrolka
 JavaFX
 menu, 1041, 1042, 1045,
 1051, 1053
 pasek narzędzi, 1057
 program, 987
 transformacja, 1033, 1035,
 1036, 1064
 zdarzenie, 990
 JavaFX Script, 982
 jednostka kompilacji, 50
 język
 assemblerowy, 32, 33
 BASIC, 32

BCPL, 32
 C, 31, 32
 funkcja, *Patrz:* funkcja
 C#, 35
 C++, 31, 33
 FORTRAN, 32, 33
 FXML, 982
 Java, *Patrz:* Java
 LDML, *Patrz:* LDML
 Oak, 33
 Pascal, 32
 proceduralny, 45
 strukturalny, *Patrz:*
 programowanie
 strukturalne
 JNI, 304
 Joy Bill, 34
 JVM, 36, 37, 444

K

kanał, 611, 612, 613, 621, 880
 bajtowy, 614
 FileChannel, 614
 SeekableByteChannel, 614
 SocketChannel, 614
 w wersjach sprzed JDK 7,
 638
 katalog, 571, 573, 617, 633, 637
 bazowy
 dokumentu, 679
 kodu, 679
 drzewo, *Patrz:* drzewo
 katalogów
 tworzenie, 575
 Kernighan Brian, 32
 klasa, 47, 48, 123, 124, 128
 AbstractAction, 967
 AbstractButton, 926, 950
 AbstractCollection, 447, 458
 AbstractList, 447, 458, 500
 AbstractMap, 447, 479, 481
 AbstractQueue, 447, 458
 AbstractSequentialList, 447,
 458, 461
 AbstractSet, 447, 458
 abstrakcyjna, 181, 182, 183,
 197
 AccessibleObject, 888
 ActionEvent, 686, 695, 754,
 777, 915, 927

- klasa
 - adaptera, 704
 - ComponentAdapter, 704
 - ContainerAdapter, 704
 - FocusAdapter, 704
 - KeyAdapter, 704
 - MouseAdapter, 704
 - MouseMotionAdapter, 704
 - WindowAdapter, 704
 - addElement, 500
 - AdjustmentEvent, 686, 687, 695, 756
 - Appendable, 539
 - Applet, 663, 664, 665, 666, 701
 - Application, 983, 984
 - AreaAveragingScaleFilter, 800
 - ArithmeticException, 211
 - Array, 888
 - ArrayDeque, 447, 458, 465
 - ArrayList, 447, 458, 459, 474, 500
 - Arrays, 447, 495
 - AtomicInteger, 813, 838
 - AtomicLong, 813, 838
 - AudioClip, 665, 666, 680
 - Authenticator, 647
 - autor, 1100
 - AWTEvent, 686, 710
 - AWTEventMulticaster, 710
 - Base64, 447, 563
 - bazowa, 165, 166
 - konstruktor, 170
 - sparametryzowana, 337
 - BeanDescriptor, 1072
 - BeanInfo, 1070, 1073
 - Beans, 1072
 - biblioteka, *Patrz*: biblioteka
 - klas
 - BitSet, 447, 514, 516
 - Blur, 808
 - Boolean, 259, 263, 399, 414, 415
 - BorderFactory, 922
 - BorderLayout, 710, 764
 - BorderPane, 1047
 - Buffer, 612
 - BufferedInputStream, 570, 586
 - BufferedOutputStream, 570, 587
 - BufferedReader, 286, 287, 288, 570, 600
 - BufferedWriter, 570, 601
 - Button, 710, 744, 991, 1003
 - ButtonBase, 1014
 - ButtonGroup, 923
 - Byte, 259, 399, 403, 404
 - ByteArrayInputStream, 570, 583
 - ByteArrayOutputStream, 570, 584
 - ByteBuffer, 613, 622
 - CacheRequest, 647
 - CacheResponse, 647
 - Calendar, 447, 519, 521, 522, 528, 541, 897
 - stała, 523
 - CallSite, 444
 - Canvas, 710, 713, 995
 - CardLayout, 710, 768
 - Character, 259, 263, 399, 411, 413
 - Character.Subset, 399
 - Character.UnicodeBlock, 399
 - CharArrayReader, 570, 598
 - CharArrayWriter, 570, 599
 - CharBuffer, 613
 - Checkbox, 710, 747
 - CheckBox, 1014
 - CheckboxGroup, 710, 749
 - CheckboxMenuItem, 710, 775, 776
 - CheckMenuItem, 1042, 1052
 - Choice, 710
 - Class, 399, 426, 428
 - ClassLoader, 399, 429
 - ClassValue, 399, 442
 - Collection, 448
 - Collections, 447, 490, 494
 - Collectors, 870
 - Color, 670, 710, 725, 727
 - ColorsBeanInfo, 1076
 - ComboBox, 1022
 - Compiler, 399, 433
 - Component, 665, 672, 701, 710, 712, 742, 908
 - ComponentEvent, 686, 688, 694
 - ConcurrentHashMap, 813
 - ConcurrentLinkedQueue, 813
 - Console, 603
 - Constructor, 888
 - Container, 665, 710, 712, 762, 908
 - ContainerEvent, 686, 688
 - ContentHandler, 647
 - ContextMenu, 1042, 1053
 - Contrast, 806
 - Control, 1038
 - Convolver, 806
 - Cookie, 1086, 1089
 - CookieHandler, 647
 - CookieManager, 647
 - CopyOnWriteArrayList, 813
 - CountDownLatch, 812, 813, 818
 - CountedCompleter, 840
 - CropImageFilter, 800
 - Currency, 447, 535
 - Cursor, 710
 - CyclicBarrier, 812, 813, 819, 823
 - DatagramPacket, 647, 659, 660, 661
 - DatagramSocket, 614, 647, 659, 660
 - DatagramSocketImpl, 647
 - DataInputStream, 570, 593
 - DataOutputStream, 570, 593
 - Date, 447, 519, 541
 - DateFormat, 519, 528, 894
 - DateTimeFormatter, 899
 - DefaultPersistenceDelegate, 1072
 - definicja, 52
 - deklaracja, 123
 - destruktor, 136
 - Dialog, 710
 - Dictionary, 447, 448, 499, 504, 505
 - Dimension, 710
 - double, 259
 - Double, 399, 400, 402, 403
 - DoubleAccumulator, 838
 - DoubleAdder, 838
 - DoubleBuffer, 613
 - DoubleSummaryStatistics, 447, 563

- egzemplarz, 47, 133,
 - Patrz też:* obiekt
- Encoder, 1072
- Enum, 399
- EnumMap, 447, 479, 483
- EnumSet, 447, 458, 466
- Error, 210, 222
- Event, 990
- EventHandler, 1072
- EventListener, 563
- EventListenerProxy, 447, 563
- EventObject, 447, 563, 685, 686
- EventQueue, 710
- EventSetDescriptor, 1070, 1072, 1073
- Exception, 210, 214, 220, 222
- Exchanger, 812, 813, 821, 822
- Executable, 888
- Executors, 812
- ExecutorService, 812
- Expression, 1072
- FeatureDescriptor, 1072
- Field, 888
- File, 570
- FileChannel, 640
- FileDescriptor, 570
- FileDialog, 710, 784
- FileInputStream, 290, 291, 570, 579, 614
- FileOutputStream, 290, 291, 570, 581, 608, 614
- FilePermission, 570
- FileReader, 570, 596
- Files, 570, 614, 615
- FileStore, 620
- FileSystem, 620
- FileSystems, 620
- FileWriter, 570, 597
- FilteredImageSource, 796
- FilterInputStream, 570, 585
- FilterOutputStream, 570, 585
- FilterReader, 570
- FilterWriter, 570
- Float, 259, 399, 400, 401, 403
- FloatBuffer, 613
- FlowLayout, 710, 762
- FocusEvent, 686, 689
- Font, 711, 729
- FontMetrics, 711, 734, 735
- ForkJoinPool, 813, 830, 839, 840, 841, 850, 852
 - wykradanie zadań, 842
- ForkJoinTask, 813, 839, 840, 841, 842, 843, 850, 851, 852
- Formattable, 563
- FormattableFlags, 447, 563
- Formatter, 447, 513, 536, 537, 538, 539, 540, 542, 543, 544, 545, 546, 548, 549, 591
- Frame, 711, 712, 713, 714
- GenericServlet, 1080, 1082, 1083
- GetField, 608
- GraphicContext, 998
- Graphics, 711, 722
- Graphics2D, 722
- GraphicsContext, 994
- GraphicsDevice, 711
- GraphicsEnvironment, 711, 730
- Grayscale, 804
- GregorianCalendar, 447, 524, 528, 897
- GridBagConstraints, 711, 771, 772
- GridBagLayout, 711, 771, 773
- GridLayout, 711, 767
- HashMap, 447, 479, 480, 481, 483
- HashSet, 447, 458, 462, 463
- Hashtable, 447, 458, 499, 505, 506
- hierarchia, 48, 49, 337,
 - Patrz:* dziedziczenie
 - hierarchia
- HttpCookie, 647
- HttpServlet, 1086, 1090
- HttpSession, 1095
- HttpURLConnection, 647, 656
- IdentityHashMap, 447, 479, 483
- IDN, 647
- Image, 665, 666, 680, 711, 787, 788, 999
- ImageFilter, 800
- ImageIcon, 923, 924
- ImageView, 999, 1001
- IndexedPropertyChange
 - ↳Event, 1072
- IndexedPropertyDescriptor, 1072
- Inet4Address, 647
- Inet6Address, 647
- InetAddress, 647, 648, 649, 661
- InetSocketAddress, 647, 659
- InheritableThreadLocal, 399, 439
- InputEvent, 686, 689, 691
- InputStream, 284, 570, 577, 578
- InputStreamReader, 570
- Insets, 711, 766
- IntBuffer, 613
- Integer, 259, 399, 403, 406, 407
- InterfaceAddress, 647
- IntrospectionException, 1072
- Introspector, 1072, 1073
- IntSummaryStatistics, 447, 563
- Invert, 805
- isTemporary, 689
- ItemEvent, 686, 690
- JApplet, 663, 909, 917
- JarURLConnection, 647
- java.io.File, 631
- java.lang.Enum, 256
- JavaFXSkel, 986
- JButton, 909, 923, 926, 927, 968
- JCheckBoxMenuItem, 960, 968
- JCheckBox, 909, 923, 926, 929, 931, 968
- JCheckBoxMenuItem, 909, 950, 960, 961
- JColorChooser, 909
- JComboBox, 909, 923, 941
- JComponent, 908, 909, 923
- JDesktopPane, 909
- JDialog, 909, 977
- JEditorPane, 909
- JFileChooser, 909
- JFormattedTextField, 909, 977
- JFrame, 909

klasa

- JInternalFrame, 909
- JLabel, 909, 923
- JLayer, 909
- JLayeredPane, 909
- JList, 909, 923, 938
- JMenu, 909, 950, 951, 952
- JMenuBar, 909, 950, 951, 953
- JMenuItem, 909, 950, 951, 953, 968
- JOptionPane, 909, 977
- JPanel, 909
- JPasswordField, 909
- JPopupMenu, 909, 950, 962, 968
- JProgressBar, 909
- JRadioButton, 909, 923, 926, 929, 932, 968
- JRadioButtonMenuItem, 909, 950, 960, 961, 968
- JRootPane, 909
- JScrollBar, 909
- JScrollPane, 909, 923, 936, 938
- JSeparator, 909, 950
- JSlider, 909
- JSpinner, 909, 977
- JSplitPane, 909
- JTabbedPane, 909, 923, 934
- JTable, 909, 923, 945
- JTextArea, 909
- JTextComponent, 925
- JTextField, 909, 923, 925
- JTextPane, 909
- JToggleButton, 909, 923, 926, 929
- JToolBar, 909, 950, 965, 968
- JToolTip, 909
- JTree, 909, 923, 943
- JViewport, 909
- JWindow, 909
- KeyCombination, 1050
- KeyEvent, 686, 689, 691
 - kolekcji, 458
- konstruktor, 127, 133, 134, 170
 - domyślny, 134
 - sparametryzowany, 134
- Label, 711, 743, 988
- LineNumberInputStream, 570
- LineNumberReader, 570
- LinkedHashMap, 447, 479, 482
- LinkedHashSet, 447, 458, 463
- LinkedList, 447, 458, 461, 473
- List, 711, 753
- ListResourceBundle, 447, 561
- ListView, 1017, 1020
- LoadedImage, 804
- LocalDate, 898, 899, 901
- LocalDateTime, 898, 899, 901
- Locale, 447, 527, 666
 - stałe, 527
- Locale.Builder, 528
- LocalTime, 898, 899, 901
- Long, 259, 399, 403, 408
- LongAccumulator, 838
- LongAdder, 838
- LongBuffer, 613
- LongSummaryStatistics, 447
- MappedByteBuffer, 613
- Matcher, 879, 882, 884
- Math, 309, 399, 429, 430, 432
- MediaTracker, 711, 793, 794
- MemoryImageSource, 796
- Menu, 711, 775, 1042, 1043
- MenuBar, 711, 775, 1042, 1043
- MenuComponent, 711
- MenuItem, 711, 775, 1042, 1044
- MenuShortcut, 711
- Method, 888
- MethodDescriptor, 1070, 1072, 1073
- MethodHandle, 444
- MethodType, 444
- Modifier, 888
- MouseEvent, 686, 689, 691, 693
- MouseEvents, 701
- MouseMotionAdapter, 704
- MouseEvent, 686, 693
- MulticastSocket, 647
- nadrzędna, 170, *Patrz:* klasa bazowa
- nazwa, 58
- NetPermission, 647
- NetworkInterface, 647
- Node, 983
- Number, 399, 400
- Object, 185, 216, 240, 350, 399, 424
- ObjectInputStream, 570, 608
- ObjectInputStream.GetField, 570
- ObjectOutputStream, 570, 607, 608
- ObjectOutputStream.PutField, 570
- Objects, 447, 563
- ObjectStreamClass, 570
- ObjectStreamField, 570
- Observable, 447, 530, 531
- opakowująca, 400
- Optional, 447, 517, 518
- OptionalDouble, 447, 517, 519
- OptionalInt, 447, 517, 519
- OptionalLong, 447, 517, 519
- OutputStream, 284, 289, 570, 577, 578, 579
- OutputStreamWriter, 570
- Package, 399, 439, 440
- Panel, 665, 711, 712, 713
- Parameter, 888
- ParameterDescriptor, 1072
- PasswordAuthentication, 647
- Paths, 618
- Pattern, 879, 881, 884
- PersistenceDelegate, 1072
- Phaser, 812, 813, 823
- PipedInputStream, 570
- PipedOutputStream, 570
- PipedReader, 570
- PipedWriter, 570
- PixelGrabber, 797
- Platform, 1049
- pochodna, 173
- Point, 711
- Polygon, 711
- PopupControl, 1053
- PopupMenu, 711
- potomna, 165, 168, 177
- PrintJob, 711
- PrintStream, 549, 570, 590, 591
- PrintWriter, 289, 549, 570, 602

- PriorityQueue, 447, 458, 465
- Process, 399, 415, 416
- ProcessBuilder, 399, 415, 419, 420
- ProcessBuilder.Redirect, 399
- Properties, 447, 448, 499, 508, 509
- PropertyChangeEvent, 1072
- PropertyChangeListenerProxy, 1072
- PropertyChangeSupport, 1072
- PropertyDescriptor, 1070, 1072, 1073
- PropertyEditorManager, 1072
- PropertyEditorSupport, 1072
- PropertyPermission, 447, 563
- PropertyResourceBundle, 447, 561
- PropertyVetoException, 1072
- Proxy, 647, 888
- ProxySelector, 647
- przestarzała, 1100
- pseudolosowa, 528
- PushbackInputStream, 570, 586, 588, 589
- PushbackReader, 570, 601
- RadioButton, 1008
- RadioMenuItem, 1042, 1052
- Random, 447, 528
- RandomAccessFile, 570, 594, 614
- Reader, 284, 286, 570, 577, 595
- Rectangle, 711
- RecursiveAction, 813, 839, 840, 841, 843, 848
- RecursiveTask, 813, 839, 840, 841, 843, 848
- ReflectPermission, 888
- ReplicateScaleFilter, 800
- ResourceBundle, 447, 559, 561
- ResourceBundle.Control, 560
- ResponseCache, 647
- RGBImageFilter, 801
- Robot, 711
- Runtime, 399, 415, 416, 417
- RuntimeException, 210, 219, 222
- RuntimePermission, 399, 439
- Scanner, 447, 513, 549, 551, 552, 553, 554, 555, 557, 558
- Scene, 983
- ScheduledThreadPool
 - ↳ Executor, 830
- Scrollbar, 711, 755
- ScrollPane, 711, 1026
- SecureCacheResponse, 647
- SecurityManager, 399, 439
- Semaphore, 812, 813
- SeparatorMenuItem, 1042, 1044
- SequenceInputStream, 570, 589
- SerializablePermission, 570
- ServerSocket, 614, 647, 649, 658
- ServiceLoader, 447, 563
- Servlet API, 1081
- ServletException, 1082
- ServletInputStream, 1082, 1083
- ServletOutputStream, 1082, 1085
- Short, 259, 399, 403, 405
- ShortBuffer, 613
- SimpleBeanInfo, 1070, 1072, 1076
- SimpleDateFormat, 896
- SimpleFormat, 528
- SimpleTimeZone, 447, 525, 526
- składowa, 47
 - prywatna, 167, 170
- Socket, 614, 647, 649, 650, 651
- SocketAddress, 647
- SocketImpl, 647
- SocketPermission, 647
- sparametryzowana, 315, 316, 317, 321, 337, 340, 366, 369
 - bez argumentów typu, 335
 - rzutowanie, 342
 - składnia, 322
- SplitableRandom, 447
- Spliterators, 447
- Stack, 137, 150, 198, 447, 448, 458, 499, 503
- StackTraceElement, 399, 439, 441
- Stage, 983
- StandardSocketOption, 647
- Statement, 1072
- stosu, *Patrz:* stos
- StreamTokenizer, 570
- StrictMath, 399, 433
- String, 52, 69, 80, 102, 139, 157, 158, 216, 377, 382, 383, 384, 391, 399, 442
 - konstruktor, 377
- StringBuffer, 391, 393, 397, 399, 442
- StringBufferInputStream, 570
- StringBuilder, 398, 399, 442
- StringJoiner, 447, 563
- StringReader, 570
- StringTokenizer, 447, 513, 514
- StringWriter, 570
- System, 53, 60, 399, 421, 422
- SystemColor, 711
- TextArea, 711, 760
- TextComponent, 711
- TextEvent, 686, 694, 695
- TextField, 711, 758, 1024
- Thread, 228, 229, 399, 433, 434, 533, 811
 - rozszerzanie, 232
- ThreadGroup, 433, 435, 444
- ThreadLocal, 399, 439
- ThreadPoolExecutor, 830
- Throwable, 210, 212, 216, 220, 348, 399, 439
- Timer, 447, 533, 534, 535
- TimerTask, 447, 533, 535
- TimeZone, 447, 525
- ToggleButton, 1005, 1006
- ToolBar, 1057
- Toolkit, 711
- Transform, 1035
- TreeMap, 447, 479, 481, 482, 483

- klasa
- TreeSet, 458, 464, 483
 - TreeView, 1029
 - UnavailableException, 1082
 - URI, 647, 658
 - URL, 647, 653, 679, 680
 - URLClassLoader, 647
 - URLConnection, 647, 654
 - URLDecoder, 647
 - URLEncoder, 647
 - URLPermission, 647
 - URLStreamHandler, 647
 - UUID, 447, 563
 - Vector, 447, 448, 458, 474, 499, 500
 - VetoableChangeListener
 - ↳Proxy, 1072
 - VetoableChangeSupport, 1072
 - Void, 399, 415
 - WeakHashMap, 447, 479, 480
 - WebView, 1064
 - wewnętrzna, 155, 157, 705
 - anonimowa, 157, 707
 - Window, 711, 713
 - WindowEvent, 686, 694, 695
 - Writer, 284, 570, 577
 - wystąpienie,
 - Patrz: egzemplarz
 - XMLDecoder, 1072
 - XMLEncoder, 1072
 - zagnieżdżona, 155
 - zewnętrzna, 155
 - zmienna, *Patrz: zmienna*
 - składowa, *Patrz: zmienna*
 - składowa
 - znaków, 886
- klauzula
- case, 103, 252
 - catch, 211, 212, 223
 - wielokrotna, 213
 - default, 102, 103
 - extends, 265
 - finally, 297
 - implements, 195
 - throws, 217, *Patrz też: słowo*
 - kluczowe throws
- klawiatura, 689, 695, 697, 701
- kod
- bajtowy, 36
 - blok, 56, 99
 - działający na danych, 45
 - rdzenny, 303
 - spaghetti, 32
 - złośliwy, 36
 - koder, 614
 - kolejka, 455
 - priorytetowa, 465
 - kolekcja, 112, 116, 315, 367, 449, 473, 495, 858
 - element, 451
 - modyfikowalna, 450
 - niemodyfikowalna, 450
 - tworzenie, 870, 872
 - widok, 449
 - wielowątkowa, 812
 - kolor, 725
 - bieżący, 727
 - HSB, 726
 - jasność, 726
 - nasycenie, 726
 - RGB, 726, 727
 - zarządzanie, 710
 - komentarz, 51
 - dokumentacyjny, 52, 58, 1099, 1103
 - jednowierszowy, 52
 - wielowierszowy, 51
 - komparator, 485, 487, 490
 - kompilator, 34, 48, 50
 - javac, 51
 - JIT, 37, 39
 - komponent, 908
 - lekki, 923
 - komunikacja
 - międzyprocesowa, 225, 240
 - międzywątkowa, 225, 240
 - konsola, 53, 286, 288
 - konstruktor, 888
 - klasy, 127, *Patrz: klasa*
 - konstruktor
 - kolejność, 176
 - przeciążanie, 141, 142, 311
 - referencja, 368, 369, 372
 - sparametryzowany, 332
 - kontekst graficzny, 722
 - kontener, 908, 909
 - JApplet, 909
 - JDialog, 909
 - JFrame, 909
 - JPanel, 909
 - JWindow, 909
 - lekki, 909, 936
 - najwyższego poziomu, 908, 909, 917
 - panel, *Patrz: panel*
 - serwletów, *Patrz: serwer*
 - serwletów
 - kontrola dostępu, 149, 150, 167, 189
 - kontrolka, 741, 742
 - dezaktywacja, 1039
 - dodawanie, 742
 - edycji, 758
 - etykiety, 988
 - HTMLEditor, 1026
 - JavaFX, 999
 - listy, 1017, 1020
 - kombinowanej, 1022
 - odpowiednik, 906
 - paska przewijania, 711
 - PasswordField, 1026
 - pola
 - tekstowego, 711
 - wyboru, 710, 1014
 - przełącznika, 1005
 - przycisku, 710, 991
 - przycisku opcji, 1008
 - ScrollPane, 1026, 1027
 - tekstowa, 1024
 - TextArea, 1026
 - TreeView, 1029
 - zdarzenie, 742
- korzeń, 983, 1029
- krzywa
 - Gaussa, 529
 - gładka, 529
- kursor w postaci bitmapy, 710
- kwantyfikator, 884
- L**
- lambda wyrażeniowa, 354
- LDML, 528
- liczba
 - całkowita, 66
 - bit, 87
 - kodowanie, 87, 88
 - ze znakiem, 62, 87, 88
 - dziesiętna, 66

- formatowanie, 540
 - losowa, 529
 - ósemkowa, 66
 - pseudolosowa, 447, 528
 - ziarno, 528
 - rzeczywista, *Patrz:* liczba zmiennoprzecinkowa
 - szesnastkowa, 66, 68
 - zmiennoprzecinkowa, 63, 67, 68
 - notacja, 67, 68
 - pojedynczej precyzji, 64
 - Lindholm Tim, 34
 - lista, 448, 495, 510, 742, 753, 938, 939, 1017
 - kombinowana, 941, 1022
 - edycja, 1023
 - obsługa zdarzeń, 754
 - rozwijana, 710, 742, 751
 - obsługa zdarzeń, 751
 - wielokrotnego wyboru, 1021
 - liść, 983, 1029
 - literał, 58, 882
 - tekstowy, 379
 - Locale Data Markup Language, *Patrz:* LDML
- Ł**
- łańcuch
 - przydzielania zdarzeń, 991
 - znaków, 69, 80, 102, 104, 157, 377, 379, 382, 387
 - długość, 379
 - formatowanie, 539
 - konkatenacja, 380, 388, 390
 - konkatenacja z innymi typami danych, 380
 - konwersja, 381
 - wejściowy sformatowany, 513
- M**
- mapa, 449, 476, 477, 478, 482, 483, 492
 - iteracja, 482
 - płaska, 870
 - posortowana, 477
 - usuwanie najstarszych wpisów, 483
 - maszyna wirtualna Javy, *Patrz:* JVM
 - mechanizm
 - dekodujący, *Patrz:* dekodery
 - kodujący, *Patrz:* koder
 - konwersji formatów, *Patrz:* dane konwersja formatu
 - obsługi zdarzeń, *Patrz:* zdarzenie obsługa
 - mechanizm odzyskiwania pamięci, 136, 416, 421, 445
 - memory leak, *Patrz:* wyciek pamięci
 - menedżer układu graficznego, 712, 741, 762, 916
 - CardLayout, 768
 - domyślny, *Patrz:* klasa FlowLayout
 - FlowLayout, 986
 - GridLayout, 767
 - LayoutManager, 804
 - menu, 107, 775, 949, 951, 952, 1041, 1043, 1051
 - dodawanie obrazów, 959
 - element, 776
 - główne, 741, 953
 - tworzenie, 1045
 - kontekstowe, *Patrz:* menu podręczne
 - obsługa zdarzeń, 777
 - pasek, *Patrz:* pasek menu
 - podręczne, 949, 950, 962, 1042, 1053
 - wyświetlanie, 962
 - rozwijane, 711
 - separator, 1042, 1044
 - standardowe, 949
 - tworzenie, 953
 - metadane, *Patrz:* adnotacje
 - metoda, 47, 124, 128
 - abs, 431
 - abstrakcyjna, 182, 350
 - accept, 471, 564, 574, 658
 - actionPerformed, 696, 744, 746, 956, 967, 969
 - activeCount, 434
 - add, 451, 452, 453, 454, 462, 468, 521, 872, 951, 952
 - addAll, 451, 452, 453, 454, 490, 872
 - addCookie, 1088
 - addElement, 501
 - addEventFilter, 991
 - addExact, 432
 - addFirst, 457, 458, 461, 873
 - addLast, 457, 458, 461
 - addListener, 1011, 1030
 - addMatch, 877
 - addObserver, 530
 - addSeparator, 952
 - addShutdownHook, 417
 - addSuppressed, 220
 - addTListener, 1069
 - adjustmentValueChanged, 696
 - after, 520, 521
 - allocate, 622, 623
 - and, 515
 - andNot, 515
 - annotationType, 265
 - anyMatch, 877
 - append, 394, 442, 596
 - apply, 564, 566
 - applyAsDouble, 565, 566, 869
 - applyAsInt, 565, 566
 - applyAsLong, 565, 566
 - argument, 128, 131
 - array, 612
 - arraycopy, 421, 423
 - arrayOffset, 612
 - arrive, 824
 - arriveAndAwaitAdvance, 824, 829
 - arriveAndDeregister, 824
 - asLifoQueue, 490
 - asList, 495
 - available, 578, 608, 609
 - availableProcessors, 848
 - average, 323
 - await, 818, 820, 836
 - before, 520, 521
 - binarySearch, 490, 495
 - bitCount, 406, 408
 - boolean, 503
 - booleanValue, 415
 - bulkRegister, 829
 - ByteBuffer, 613
 - bytesWidth, 735
 - byteValue, 259, 264, 401, 404, 405, 406
 - cancel, 533, 534, 535, 850

metoda

- capacity, 393, 501, 612
- cardinality, 515
- ceil, 431
- ceiling, 456
- ceilingEntry, 478
- ceilingKey, 478
- Character.Subset, 413
- Character.UnicodeBlock, 413
- characteristics, 471, 472
- charAt, 158, 382, 393, 442
- charCount, 414
- chars, 442
- charWidth, 735
- checkAccess, 434
- checkedCollection, 490
- checkedList, 490, 494
- checkedMap, 491, 494
- checkedQueue, 491
- checkedSet, 494
- checkedSortedMap, 491
- checkedSortedSet, 491
- checkID, 794
- ciało, 52
- Class, 424
- clear, 451, 452, 475, 506, 515, 521, 612
- clearCache, 560
- clearChanged, 530
- clearProperty, 421
- clone, 425, 501, 506, 515, 520, 526, 1089
- Clone, 185
- close, 291, 443, 549, 575, 576, 578, 579, 582, 584, 595, 596, 598, 606, 607, 608, 609, 651, 660, 856
- codePointAt, 413, 414
- codePointBefore, 414
- codePoints, 442
- collect, 857, 870
- command, 420
- commonPool, 842
- comparator, 454, 465, 477
- compare, 401, 404, 405, 406, 408, 415, 483, 485
- compareAndSet, 813, 838
- compareTo, 256, 385, 401, 402, 404, 405, 406, 408, 413, 415, 441, 442, 520, 573
- compareUnsigned, 406, 408
- comparing, 485
- comparingByKey, 479
- comparingByValue, 479
- compile, 881
- componentAdded, 697
- componentRemoved, 697
- compute, 475, 841
- computeIfAbsent, 475
- computeIfPresent, 475
- concat, 388
- connect, 650
- console, 421
- contains, 451, 452, 462, 501, 506
- containsAll, 451, 452
- containsHeader, 1088
- containsKey, 475, 506, 560
- containsValue, 475, 506
- convert, 835
- copyInto, 501
- copyOf, 466, 495
- copyOfRange, 496
- copySign, 432
- count, 857, 877
- counter, 531
- countObservers, 530
- countTokens, 514
- createDirectory, 617
- createFile, 617
- createImage, 788, 796
- creationTime, 619
- currentThread, 434
- currentTimeMillis, 421, 422
- decode, 404, 405, 406, 408, 730
- decrementAndGet, 813, 838
- decrementExact, 432
- deepEquals, 496
- deepHashCode, 498
- deepToString, 498
- delete, 395, 572, 617
- deleteCharAt, 395
- deleteObserver, 530
- deleteObservers, 530
- deleteOnExit, 572
- descendingIterator, 456, 457, 458
- descendingKeySet, 478
- descendingMap, 478
- destroy, 416, 419, 665, 667, 669, 1078, 1082
- destroyForcibly, 416
- digit, 413
- directory, 420
- disjoint, 491
- distinct, 877
- divideUnsigned, 406, 408
- doDelete, 1090
- doGet, 1090
- doHead, 1090
- domyślna, 203, 204, 205, 206, 351
- doOptions, 1090
- doPost, 1090
- doPut, 1090
- doTrace, 1090
- doubles, 530
- doubleToLongBits, 402
- doubleValue, 259, 260, 323, 400, 401, 402, 404, 406, 408
- dowiązanie
 - późne, 184
 - wczesne, 184
- drawArc, 723
- drawImage, 791, 793
- drawLine, 722, 919
- drawOval, 723
- drawPolygon, 723
- drawRect, 722, 919
- drawString, 664, 670, 734
- dumpStack, 434
- element, 455, 456
- elementAt, 500, 501
- elements, 504, 505, 506
- empty, 503, 518
- emptyList, 491
- emptyListIterator, 491
- emptyNavigableMap, 491
- emptyNavigableSet, 491
- emptySortedMap, 492
- emptySortedSet, 492
- encodeRedirectURL, 1088
- encodeURL, 1088
- end, 882
- endsWith, 616
- ensureCapacity, 393, 460, 501
- entrySet, 475, 481, 507
- enumerate, 434
- enumeration, 492

- environment, 420
- equals, 158, 185, 265, 350, 383, 384, 401, 402, 404, 405, 406, 413, 415, 424, 441, 451, 452, 475, 484, 496, 505, 515, 518, 520, 522, 649, 730
- equalsIgnoreCase, 383
- estimateSize, 471
- exchange, 821
- exec, 415, 417, 418, 419
- execute, 842, 850
- exists, 571, 617, 632
- exit, 417, 421
- exitValue, 416, 419
- extends, 467, 518
- fabryczna, 647
- fileKey, 619
- fill, 496
- fillArc, 723
- fillInStackTrace, 220
- fillOval, 995
- fillPolygon, 723
- fillRect, 722, 995
- fillText, 995
- filter, 518, 804, 857, 858, 861
- filterRGB, 804
- finalize, 136, 185, 422, 424
- find, 615
 - dopasowanie oszczędne, 885
 - dopasowanie zachłanne, 885
- findInLine, 558
- findWithinHorizon, 559
- fire, 1008, 1011
- first, 454
- firstElement, 501
- firstEntry, 478
- firstKey, 477
- flatMap, 518, 870
- flatMapToInt, 870
- flatMapToLong, 870
- flip, 515, 612, 627
- floatToIntBits, 401
- floatToRawIntBits, 401
- floatValue, 259, 401, 404, 406, 408
- floor, 431
- floorDiv, 431
- floorEntry, 478
- floorKey, 478
- floorMod, 431
- flush, 538, 579, 587, 596, 604, 606, 607
- focusGained, 697
- focusLost, 697
- forceTermination, 829
- forDigit, 413
- forEach, 475, 857, 866
- forEachOrdered, 866
- forEachRemaining, 468, 471, 875
- fork, 840
- format, 538, 592, 603, 604
- forName, 427
- freeMemory, 416, 417
- from, 520
- gc, 416
- get, 453, 505, 515, 518, 522, 613, 618, 621, 838
- getAccessibleContext, 665
- getActionCommand, 745, 754, 933, 956
- getActiveThreadCount, 852
- getAddListenerMethod, 1073
- getAddress, 649, 661
- getAdjustable, 687
- getAdjustmentType, 756
- getAllByName, 648
- getAndSet, 813, 838
- getAnnotation, 267, 270, 279, 440
- getAnnotations, 269, 270, 427, 440
- getAnnotationsByType, 270, 280, 281
- getApplet, 680
- getAppletContext, 665, 679
- getAppletInfo, 665
- getApplets, 680
- getArrivedParties, 829
- getAsBoolean, 565
- getAscent, 735
- getAsInt, 565
- getAttribute, 1083, 1084, 1088, 1095
- getAttributeNames, 1088, 1095
- getAudioClip, 665, 680, 681
- getAuthType, 1087
- getAvailableCurrencies, 536
- getAvailableFontFamilyNames, 730
- getAvailableIDs, 526
- getBackground, 670
- getBaseBundleName, 560
- getBeanInfo, 1073
- getBlue, 727
- getBoolean, 415
- getBundle, 560
- getButton, 692
- getByAddress, 648
- getByName, 648
- getBytes, 382, 581
- getCause, 220, 222
- getChannel, 614, 638
- getCharacterEncoding, 1084
- getChars, 382, 394
- getChildren, 988, 1030
- getClass, 185, 266
- getClasses, 427
- getClassLoader, 427
- getClassName, 441
- getClickCount, 692
- getCodeBase, 665, 679
- getComment, 1089
- getCommonPoolParallelism, 848
- getComparator, 471
- getComponent, 688, 964
- getConstructor, 267
- getConstructors, 427, 888
- getContainer, 688
- getContentLength, 654, 1084
- getContentLengthLong, 654
- getContentPane, 913, 916
- getContents, 561
- getContentType, 654, 655, 1084
- getCookies, 1087, 1094
- getCreationTime, 1088
- getCurrencyCode, 536
- getData, 661
- getDate, 654, 655
- getDateHeader, 1087
- getDateInstance, 894
- getDeclaredAnnotation, 270, 440
- getDeclaredAnnotations, 270, 427

metoda

- getDeclaredAnnotationsBy
 - ↳ Type, 270, 280, 427, 440
- getDeclaredConstructors, 427
- getDeclaredFields, 427
- getDeclaredMethods, 427
- getDeclaringClass, 441
- getDefault, 526
- getDefaultFractionDigits, 536
- getDescent, 735
- getDirectionality, 413
- getDisplayName, 536
- getDocumentBase, 665, 679
- getDomain, 1089
- getDouble, 519
- getenv, 421
- getErrorMessage, 416
- getEventSetDescriptors, 1076
- getExactSizeIfKnown, 471
- getExpiration, 654
- getExponent, 432
- getField, 267
- getFields, 427, 888
- getFileAttributeView, 620
- getFileName, 441, 616
- getFirst, 457, 461
- getFollowRedirects, 656
- getFont, 730, 733, 995
- getFontName, 730
- getForeground, 670
- getForkJoinTaskTag, 852
- getFreeSpace, 572
- getGraphics, 672, 722
- getGraphicsContext2D, 995
- getGreen, 727
- getHeader, 1087
- getHeaderField, 654, 655
- getHeaderFieldKey, 654
- getHeaderFields, 654, 655, 658
- getHeaderNames, 1087
- getHeight, 735
- getHostAddress, 649
- getHostName, 649
- getHvalue, 1027
- getId, 1088
- getID, 434, 686
- getImage, 665, 666, 680, 788
- getImplementationTitle, 440
- getImplementationVendor, 440
- getImplementationVersion, 440
- getInetAddress, 650, 660
- getInitParameter, 1082
- getInitParameterNames, 1082
- getInputStream, 416, 419, 650, 654, 1084
- getInsets, 766, 920
- getInstance, 522, 536
- getInt, 519
- getInteger, 406
- getInterfaces, 428
- getIntHeader, 1087
- getItem, 690
- getItemCount, 751
- getItemSelectable, 690, 754
- getKey, 479, 481
- getKeys, 561
- getLabel, 744
- getLast, 457
- getLastAccessedTime, 1088
- getLastModified, 654, 1090
- getLeading, 735
- getLength, 661
- getLineNumber, 441
- getListenerType, 1073
- getLocale, 561, 666
- getLocalGraphicsEnvironment, 731
- getLocalHost, 648
- getLocalizedMessage, 220
- getLocalPort, 650, 660
- getLocationOnScreen, 693
- getLong, 408, 519
- getMaxAdvance, 735
- getMaxAge, 1089
- getMaxAscent, 735
- getMaxDescent, 735
- getMenuComponentCount, 952
- getMenuComponents, 952
- getMenus, 1043
- getMessage, 220
- getMethod, 267, 268, 428, 1073, 1087
- getMethodName, 441
- getMethods, 428, 888
- getMimeType, 1083
- getModifiersEx, 690
- getName, 228, 230, 428, 434, 615, 616, 632, 730, 1073, 1089
- getNameCount, 615, 616
- getNumericCode, 536
- getObject, 561
- getOffset, 661
- getOppositeComponent, 689
- getOrDefault, 475
- getOutputStream, 416, 419, 650, 1084
- getPackage, 440
- getPackages, 440
- getParallelism, 848
- getParameter, 666, 676, 677, 1084
- getParameterInfo, 666
- getParameterNames, 1084
- getParameterValues, 1084
- getParent, 571, 616, 632, 829
- getPath, 1089
- getPathInfo, 1087
- getPathTranslated, 1087
- getPoint, 692
- getPoolSize, 852
- getPort, 650, 660, 661
- getPriority, 228, 434
- getProperties, 421
- getProperty, 422, 508, 509
- getPropertyDescriptors, 1076
- getProtocol, 1084
- getQueryString, 1087
- getQueuedTaskCount, 852
- getRawOffset, 526
- getReader, 1084
- getRealPath, 1083
- getRed, 727
- getRemoteAddr, 1084
- getRemoteHost, 1084
- getRemoteUser, 1087
- getRemoveListenerMethod, 1073
- getRequestedSessionId, 1087
- getRequestMethod, 656
- getRequestURI, 1087
- getRequestURL, 1087
- getResponseCode, 656
- getResponseMessage, 656
- getRGB, 727

- getRoot, 616
- getRuntime, 417
- getScheme, 1084
- getScript, 528
- getScrollType, 693
- getSecure, 1089
- getSecurityManager, 439
- getSelectedCheckbox, 749
- getSelectedIndex, 751, 753
- getSelectedIndexes, 753
- getSelectedItem, 751, 753, 941
- getSelectedItems, 753
- getSelectedText, 758
- getSelectionModel, 1030
- getServerInfo, 1083
- getServerName, 1084
- getServerPort, 1084
- getServletConfig, 1082
- getServletContext, 1082
- getServletInfo, 1082
- getServletName, 1082
- getServletPath, 1087
- getSession, 1087, 1095
- getSize, 725, 730
- getSource, 685, 933, 991
- getSpecificationTitle, 440
- getSpecificationVendor, 440
- getSpecificationVersion, 440
- getStackTrace, 220, 434, 439
- getState, 247, 434
- getStateChange, 691
- getStream, 680
- getStreamKeys, 680
- getString, 561
- getStringArray, 561
- getStyle, 730
- getSuperclass, 428
- getSuppressed, 220
- getSurplusQueuedTask
 - ↳ Count, 852
- getSymbol, 536
- getText, 743
- getTime, 520, 522
- getTimeZone, 522, 526
- getTotalSpace, 572
- getTransforms, 1035
- getUnarrivedParties, 829
- getUncaughtExceptionHandler, 434
- getUsableSpace, 572
- getValue, 479, 481, 967, 1089
- getVersion, 1089
- getVvalue, 1027
- getWheelRotation, 693
- getWidths, 735
- getWindow, 695
- getWriter, 1084
- getX, 692
- getY, 692
- grabPixels, 797
- group, 620, 882
- handle, 993
- handleGetObject, 561
- hasArray, 612
- hasChanged, 530
- hasCharacteristics, 471, 472
- hashCode, 185, 265, 401, 402, 404, 405, 406, 408, 413, 415, 424, 440, 441, 452, 479, 505, 515, 518, 520, 730
- hasMoreElements, 499, 514
- hasMoreTokens, 514
- hasNext, 468, 471, 551, 873
- hasNextBigDecimal, 551
- hasNextBigInteger, 552
- hasNextBoolean, 552
- hasNextByte, 552
- hasNextInt, 552
- hasNextLong, 552
- hasNextShort, 552
- hasPrevious, 468
- hasRemaining, 612
- headMap, 477, 478
- headSet, 454
- higher, 456
- higherEntry, 478
- higherKey, 478
- highestOneBit, 406, 408
- holdsLock, 434
- HSBtoRGB, 726
- HTTP, 1087
- hypot, 432
- identityHashCode, 422
- IEEEremainder, 432
- ifPresent, 518
- imageUpdate, 790
- incrementExact, 432
- indexOf, 386, 453, 454, 500, 501
- indexOfSubList, 492
- InetAddress, 648
- inheritedChannel, 422
- init, 666, 667, 668, 669, 670, 983, 986, 1078, 1082
- initCause, 220, 222
- insert, 395
- insertSeparator, 952
- instancyna
 - referencja, 363
- interfejsu, 196
- interrupted, 434
- intersects, 515
- ints, 530
- intValue, 259, 260, 401, 405, 406, 408
- invalidate, 1088
- invoke, 842, 850
- invokeAll, 840, 848
- invokeAndWait, 918
- invokeLater, 918
- IOException, 538
- isAbsolute, 572, 616
- isActive, 666
- isAlive, 228, 234, 416, 434
- isAltDown, 690
- isAltGraphDown, 690
- isAnnotationPresent, 270, 272
- isArchive, 619
- isBmpCodePoint, 414
- isBold, 730
- isBound, 650, 660
- isCancelled, 851
- isClosed, 650
- isCompletedAbnormally, 851
- isCompletedNormally, 851
- isConnected, 650, 660
- isConstrained, 1073
- isControlDown, 690
- isDaemon, 434
- isDefined, 412
- isDigit, 412, 413
- isDirect, 612
- isDirectory, 617, 619
- isEditable, 758
- isEmpty, 452, 476, 505, 506, 515
- isEnabled, 776
- isExecutable, 617, 632
- isFile, 572

metoda

- isFinite, 401
- isHidden, 572, 617, 619, 632
- isHighSurrogate, 414
- isHttpOnly, 1089
- isIdentifierIgnorable, 412
- isIndeterminate, 1016
- isInfinite, 402, 403
- isInterface, 428
- isInterrupted, 434
- isISOControl, 412
- isItalic, 730
- isJavaIdentifierPart, 412
- isJavaIdentifierStart, 412
- isLeapYear, 524
- isLetter, 412, 413
- isLetterOrDigit, 412
- isLowerCase, 412
- isLowSurrogate, 414
- isMetaDown, 690
- isMirrored, 412
- isMulticastAddress, 649
- isNaN, 401, 402, 403
- isNativeMethod, 441
- isOther, 619
- isParallel, 856
- isPlain, 730
- isPopupTrigger, 692, 964
- isPresent, 518
- isQuiescent, 852
- isReadable, 617, 632
- isReadOnly, 612, 619
- isRegularFile, 617, 619
- isRequestedSessionIdFromCookie, 1087
- isSealed, 440
- isSelected, 929, 1006, 1007, 1016
- isSet, 522
- isShiftDown, 690
- isSpaceChar, 412
- isSupplementaryCodePoint, 414
- isSurrogatePair, 414
- isSymbolicLink, 619
- isSystem, 619
- isTitleCase, 412
- isUnicodeIdentifierPart, 412
- isUnicodeIdentifierStart, 412
- isUpperCase, 412
- isValidCodePoint, 414
- isWhitespace, 412
- isWritable, 617, 632
- itemStateChanged, 697, 748, 751, 929
- iterator, 443, 451, 452, 468, 856
- join, 228, 234, 390, 435, 840
- keyCombination, 1050
- keyPressed, 697, 701
- keyReleased, 697, 701
- keys, 504, 506
- keySet, 476, 507, 658
- keyTyped, 697, 701
- końcowa, 858
- last, 454, 455
- lastAccessTime, 619
- lastElement, 500, 501
- lastEntry, 478
- lastIndexOf, 386, 453, 454, 500, 501
- lastIndexOfSubList, 492
- lastKey, 477
- lastModifiedTime, 619
- launch, 984, 986
- length, 158, 393
- limit, 612
- lines, 615
- lineSeparator, 422
- list, 508, 571, 573, 615
 - filtrowanie, 574
- listFiles, 575
- listIterator, 453, 468
- load, 417, 422, 508, 510
- loadLibrary, 304, 417, 422
- locale, 538
- Locale, 522
- lock, 836
- lockInterruptibly, 836
- log, 1083
- logicalAnd, 415
- logicalOr, 415
- logicalXor, 415
- longBitsToFloat, 402
- longs, 530
- longValue, 259, 404, 405, 406, 408
- loop, 681
- lower, 456
- lowerEntry, 478
- lowerKey, 478
- lowestOneBit, 406, 408
- main, 52, 124, 986
- map, 625, 628
- mapLibraryName, 422
- mapToDouble, 857
- mapToInt, 857
- mapToLong, 857
- mark, 578, 579, 583, 586, 589, 595, 600, 612
- markSupported, 578, 586, 595
- matches, 882, 887
- Math.pow, 309
- Math.sqrt, 309
- max, 367, 401, 406, 408, 431, 492, 857, 862
- merge, 476
- min, 401, 406, 408, 431, 492, 857, 862
- mkdir, 575
- mkdirs, 575
- mostu, 344
- mouseClicked, 697
- mouseDragged, 698, 704, 793
- mouseEntered, 697
- mouseExited, 697
- mouseMoved, 698, 704, 793
- mousePressed, 697, 962, 963
- mouseReleased, 697, 962, 963, 964
- mouseWheelMoved, 698
- move, 617
- multiplyExact, 432
- myMeth, 265
- name, 441
- nanoTime, 422
- naturalOrder, 484
- navigableKeySet, 478
- NavigableMap, 491
- NavigableSet, 491
- nazwa, 58
- negateExact, 432
- newAudioClip, 666
- newByteChannel, 614, 617, 622, 623, 626, 628
- newCachedThreadPool, 830
- newCondition, 836
- newDirectoryStream, 617, 633
- newFileSystem, 620

- newFixedThreadPool, 830
- newInputStream, 617
- newOutputStream, 617
- newScheduledThreadPool, 830
- newSetFromMap, 492
- next, 468, 471, 553, 873
- nextAfter, 431
- nextBigDecimal, 553
- nextBigInteger, 553
- nextBoolean, 528, 553
- nextByte, 553
- nextBytes, 528, 529
- nextClearBit, 515
- nextDouble, 528, 529, 553
- nextDown, 431
- nextElement, 499, 514, 590
- nextFloat, 528, 529, 553
- nextGaussian, 528, 529
- nextInt, 468
- nextInt, 528, 529, 551, 553
- nextLong, 528, 529
- nextToken, 514
- nextUp, 431
- noneMatch, 877
- notifyObservers, 530
- notExists, 618
- notify, 185, 240, 241, 243, 246, 424, 811
- notifyAll, 185, 240, 424
- notifyObservers, 530, 531
- notifyAll, 185
- nullsFirst, 484
- nullsLast, 484
- NumberFormatException, 407, 409
- numberOfLeadingZeros, 406, 408
- numberOfTrailingZeros, 407, 409
- observableArrayList, 1018, 1022
- of, 518, 877
- offer, 455, 465
- offerFirst, 457, 458, 461
- offerLast, 457, 458, 461
- onAdvance, 829
- onClose, 856
- openConnection, 654, 655
- or, 515
- ordinal, 256, 441
- orElseGet, 518
- out, 538, 539
- owner, 620
- paint, 667, 668, 669, 671, 672, 919
 - przykrywanie, 786
- paintBorder, 919
- paintChildren, 919
- paintComponent, 919, 920
- parallel, 856, 864
- parallelPrefix, 498
- parallelSetAll, 498
- parallelSort, 497
- parallelStream, 451, 858, 864
- parametr, 1100,
 - Patrz:* parametr
- parse, 901
- parseBoolean, 415
- parseByte, 410
- parseDouble, 402
- parseFloat, 401
- parseInt, 410
- parseLong, 410
- parseShort, 405, 410
- parseUnsignedInt, 407
- parseUnsignedLong, 409
- peek, 455, 456, 503
- peekFirst, 457, 461
- peekLast, 457
- permissions, 620
- Platform.exit, 1049
- play, 666, 681
- poll, 455, 456, 465
- pollFirst, 456, 457, 461
- pollLast, 456, 457
- pollLastEntry, 478
- pop, 137, 198, 200, 457, 458, 503
- position, 612
- postVisitDirectory, 637
- pośrednia, 858
- preferredLayoutSize, 762
- previous, 468
- previousClearBit, 515
- previousIndex, 468
- previousSetBit, 515
- preVisitDirectory, 636, 637
- print, 54, 289, 591
- printf, 592, 603
- println, 53, 54, 66, 289, 591, 592, 603
- printStackTrace, 220
- prywatna, 47
- przeciążanie, 139, 140, 141, 162
- przesłanianie, 177, 178, 180, 184, 342
- przydzielanie dynamiczne, 178
- publiczna, 47
- push, 137, 198, 200, 457, 458, 503
- put, 476, 481, 483, 504, 505, 506, 613
- putAll, 476, 483
- putIfAbsent, 476
- putValue, 967, 969
- quietlyInvoke, 852
- quietlyJoin, 852
- range, 466
- rdzenna, 304, 306
- read, 284, 286, 291, 443, 578, 595, 608, 609, 614
- readAttributes, 618, 620
- readBoolean, 609
- readByte, 609
- readChar, 609
- readDouble, 609
- reader, 604
- readExternal, 606
- readFloat, 609
- readFully, 609
- readInt, 609
- readLine, 287, 604
- readLong, 609
- readObject, 608, 609
- readPassword, 604, 605
- readUnsignedByte, 609
- readUnsignedShort, 609
- ready, 595
- receive, 660
- redirectError, 420
- redirectErrorStream, 420
- redirectInput, 420
- redirectOutput, 420
- reduce, 857, 862, 863
- referencja, 362, 366, 367
- regionMatches, 384
- register, 824
- rehash, 506
- reinitialize, 851
- rekurencyjna,
 - Patrz:* rekurencja

metoda

remainderUnsigned, 407, 409
 remove, 451, 452, 453, 455, 456, 462, 468, 476, 505, 506, 952
 removeActionListener, 916
 removeAll, 451, 452
 removeAllElements, 501
 removeAttribute, 1088, 1095
 removeEldestEntry, 483
 removeElement, 500
 removeElementAt, 500
 removeFirst, 457, 461
 removeIf, 452
 removeLast, 457
 removeShutdownHook, 417
 removeTlListener, 1069
 renameTo, 572
 repaint, 672, 920, 994
 replace, 388, 396, 476
 replaceAll, 453, 454, 476, 882, 886
 replaceRange, 761
 reset, 578, 579, 583, 585, 586, 589, 595, 600, 612
 resize, 666
 resolve, 615, 616
 ResourceBundle, 561
 resume, 245, 246, 434
 retainAll, 451, 452
 reverse, 395, 407, 492
 reverseBytes, 405, 407, 409
 reversed, 484, 486
 reversedOrder, 484
 reverseOrder, 492, 494
 rewind, 612, 627
 RGBtoHSB, 726
 rint, 431
 rotate, 492
 rotateLeft, 407, 409
 rotateRight, 407, 409
 round, 431
 rozszerzenia, 203
 run, 228, 350, 433, 533
 runFinalization, 417, 422
 rysujaca, 723
 save, 508
 schedule, 534
 scheduleAtFixedRate, 534
 scheduledExecutionTime, 533

search, 503
 SecurityManager, 422
 seek, 594
 send, 660
 sendError, 1088
 sendRedirect, 1088
 sequential, 856
 service, 1078, 1082, 1090
 set, 453, 454, 468, 516, 838
 Set, 536
 setAccelerator, 1049
 setActionCommand, 747
 setAddress, 661
 setAll, 498
 setAllowIndeterminate, 1016
 setAttribute, 1083, 1088, 1095
 setBackground, 670
 setBounds, 762
 setChanged, 530
 setCharAt, 394
 setColor, 727
 setComment, 1089
 setConstraints, 771
 setContent, 1027
 setContentDisplay, 1003, 1058
 setContentLength, 1084
 setContentType, 1081, 1084
 setContextClassLoader, 435
 setDaemon, 435
 setData, 661
 setDateHeader, 1088
 setDefault, 527
 setDefaultCloseOperation, 912
 setDefaultUncaught
 ↳ ExceptionHandler, 435
 setDisable, 1044
 setDisabledIcon, 926
 setDisplayedMnemonic
 ↳ Index, 957
 setDomain, 1089
 setEchoChar, 759
 setEditable, 758, 1023
 setEnabled, 776, 953, 967
 setErr, 422
 setFollowRedirects, 656
 setFont, 995
 setForeground, 670
 setForkJoinTaskTag, 852

setGraphics, 1003, 1044
 setHeader, 1088
 setHorizontalTextPosition, 959
 setHttpOnly, 1089
 setIn, 422
 setIntHeader, 1088
 setLabel, 744, 776
 setLastModified, 572
 setLayout, 762, 913
 setLength, 393, 594, 595, 661
 setMaxAge, 1089, 1094
 setMnemonic, 957
 setMnemonicParsing, 1050
 setName, 230, 435
 setOnAction, 991
 setOrientation, 1057
 setOut, 422
 setPaintMode, 728
 setPannable, 1027
 setParent, 561
 setPath, 1089
 setPort, 661
 setPrefColumnCount, 1024
 setPrefHeight, 1018
 setPrefSize, 1018
 setPrefWidth, 1018
 setPressedIcon, 926
 setPriority, 236, 435
 setProperty, 508
 setRawOffset, 526
 setReadOnly, 572
 setRequestMethod, 656
 setRolloverIcon, 926
 setRotate, 1035
 setScaleX, 1035
 setScaleY, 1035
 setScene, 987
 setSecure, 1089
 setSeed, 528
 setSelected, 1008, 1011, 1052
 setSelectedIcon, 926
 setSelectionMode, 939, 1021
 setSize, 501, 912
 setSoTimeout, 660
 setStackTrace, 220
 setStatus, 1088
 setStream, 680
 setStub, 666
 setText, 743, 1024, 1044
 setTime, 520

- setTimeZone, 522
- setTooltip, 1038
- setToolTipText, 959
- setTranslateX, 1035
- setTranslateY, 1035
- setUnitIncrement, 756
- setValue, 479, 1089
- setValues, 756
- setVersion, 1089
- setVisible, 714, 913
- setXORModel, 728
- shortValue, 259, 402, 405, 407, 409
- show, 987
- showDocument, 679, 680
- showStatus, 674, 680
- shuffle, 492
- shutdown, 842
- signal, 836
- signum, 407, 409, 432
- singleton, 493
- singletonList, 493
- singletonMap, 493
- size, 462, 501, 504, 505, 506
- skip, 578, 595, 608
- składowa, *Patrz:* metoda
- składowe, 727
- sleep, 228, 229, 435, 835
- sort, 453, 493, 497
- sorted, 857
- sparametryzowana, 315, 366
 - tworzenie, 331
- split, 887
- spliterator, 451, 452, 497, 856
- start, 228, 230, 415, 435, 666, 667, 668, 669, 882, 984
- startsWith, 384, 616
- statyczna, 153, 207, 309
 - referencja, 362
- stop, 245, 246, 434, 666, 667, 669, 681, 984, 1049
- store, 508, 510
- storeToXML, 508
- stream, 451, 516, 858, 860
- StringBuffer, 391
- strokeOval, 995
- strokeRect, 995
- strokeText, 995
- subList, 453, 454
- subMap, 477, 479
- submit, 832
- subSequence, 442
- subSet, 454, 455, 456
- substring, 387, 396
- subtractExact, 432
- sum, 401, 402, 407, 409
- suspend, 245, 246, 434
- swap, 493
- synchronizacja, 237, 238, 239
- synchronized, 813
- synchronizedList, 494
- synchronizedNavigableMap, 493
- synchronizedNavigableSet, 493
- synchronizedSet, 493, 494
- synchronizedSortedMap, 493
- synchronizedSortedSet, 493
- synchronizowana, 227
- System.exit, 1049
- System.getProperty, 424
- tailMap, 477, 479
- tailSet, 454, 456
- test, 564, 566, 862
- textValueChanged, 698
- thenComparing, 484, 488
- throws, 416, 424, 434, 441
- timedWait, 835
- toAbsolutePath, 616, 632
- toArray, 451, 452, 460, 857
- toBinaryString, 407, 409
- toByteArray, 516
- toCharArray, 383
- toChars, 414
- toCodePoint, 414
- toDegrees, 430
- toHexString, 401, 407, 409
- toIntExact, 432
- toLanguageTag, 528
- toList, 870
- toLongArray, 516
- toLowerCase, 390, 412, 413
- toOctalString, 407, 409
- toPath, 573, 615, 631
- toRadians, 430, 432
- toSet, 870
- toString, 185, 220, 221, 265, 381, 401, 402, 404, 405, 407, 409, 415, 424, 428, 435, 440, 441, 442, 460, 501, 506, 516, 517, 520, 538, 615, 649, 685, 730, 852
- totalMemory, 416
- toTitleCase, 412
- toUnsignedInt, 404, 405
- toUnsignedLong, 404, 405, 407
- toUnsignedString, 407, 409
- toUpperCase, 390
- toUpperChar, 412
- traceInstructions, 417
- traceMethodCalls, 417
- trim, 389
- trimToSize, 460, 501
- tryAdvance, 471, 874
- tryLock, 836
- trySplit, 875
- ulp, 431
- uncaughtException, 444
- unlock, 836
- unmodifiableCollection, 493
- unmodifiableList, 493
- unmodifiableMap, 493
- unmodifiableNavigableMap, 493
- unmodifiableNavigableSet, 493
- unmodifiableSet, 493
- unmodifiableSortedMap, 493
- unmodifiableSortedSet, 493
- unordered, 856, 866
- unread, 588, 602
- update, 530, 531, 669, 672, 740
- ustawiająca, 1068
- valueChanged, 938, 943
- valueOf, 253, 254, 381, 389, 401, 402, 404, 405, 407, 409, 415
- values, 253, 254, 476
- varargs, 160
 - przeciążanie, 162, 163
- visitFile, 637
- visitFileFailed, 637
- wait, 185, 240, 241, 243, 246, 424, 811
- waitFor, 416, 419
- walk, 615

metoda

walkFileTree, 636
 wartość, 1100
 windowActivated, 698
 windowClosed, 698
 windowClosing, 698, 714
 windowDeactivated, 698
 windowDeiconified, 698
 windowGainedFocus, 698
 windowIconified, 698
 windowLostFocus, 698
 write, 284, 289, 579, 596,
 606, 607, 614, 626
 writeBoolean, 607
 writeByte, 607
 writeBytes, 607
 writeChar, 607
 writeChars, 607
 writeDouble, 607
 writeExternal, 606, 1100
 writeFloat, 607
 writeInt, 607
 writeLong, 607
 writeObject, 606, 607, 608,
 1100
 writer, 604
 writeShort, 607
 wywoływanie zdalne, 39,
 605, 879, 891, 893, 1077
 xor, 516
 yield, 435
 zabroniona, 399
 zmiennargumentowa,
Patrz: metoda varargs
 zwracająca, 1068
 MIME, 1077, 1081, 1083
 mnemonik, 957, 970, 1041,
 1049
 model
 delegowania zdarzeń, *Patrz:*
 zdarzenie delegowanie
 zorientowany na procesy, 45
 Model-Delegate, *Patrz:*
 architektura model-delegacja
 Model-View-Controller, *Patrz:*
 MVC
 modyfikator
 abstract, 182
 dostępności, 52
 dostępu, 149, 150, 190, 195,
 197

private, 52, 149, 166, 170,
 190, 197
 protected, 149, 190, 197
 public, 124, 149, 190, 195, 197
 static, 124, 155
 strictfp, 303
 transient, 301
 volatile, 301
 monitor, 227, 237, 240
 multipleksing, 645
 Multipurpose Internet Mail
 Extensions, *Patrz:* MIME
 muteks, 237
 MVC, 907
 kontroler, 907
 model, 907
 widok, 907
 mysz, 691, 695, 699
 kółko, 693, 698, 699
 kursor, 698
 przycisk, 697

N

nasłuchiwanie, 645
 native code, *Patrz:* kod rdzenny
 Naughton Patrick, 33
 nazwa
 bazowa, 559
 rodziny, 559
 NetBeans, 987
 NIO, 42, 569, 570, 573, 611, 621,
 880
 na kanałach, 621, 626
 na ścieżkach i systemie
 plików, 621, 631
 na strumieniach, 621, 629
 NIO.2, 611, 621
 node, *Patrz:* węzeł
 notacja
 naukowa, 67, 68, 539, 540
 standardowa, 67

O

obiekt, 46, 123
 aktualny, 135
 cookies, 1087
 deklarowanie, 126
 deserializacja,
Patrz: deserializacja

graficzny, 725
 jako parametr, 143
 klonowanie, 425
 nasłuchujący, 684, 685, 710,
 856, 927, 990, 1069, *Patrz*
też: interfejs nasłuchujący
 obserwujący, 530
 Properties, 508
 referencja, 126, 127, 128,
 145, 169, 196
 serializacja, 570,
Patrz: obiekt serializacja
 tworzenie, 126, 133
 zwracanie, 146
 object-oriented programming,
Patrz: programowanie
 obiektowe
 obraz, 880, 1001, 1003
 filtrowanie, 804, 805
 format, *Patrz:* format
 graficzny, 787
 jądro konwolucji, 807
 kontrast, 806
 konwertowanie, 801
 ładowanie, 788, 804
 obserwator, 790
 podwójne buforowanie, 791
 pozaekranowa
 powierzchnia rysowania,
 791
 rozmywanie, 808
 tworzenie, 788, 796, 797
 wyostrzenie, 809
 wyświetlanie, 789
 obserwator, 530
 obsługa wyjątków,
Patrz: wyjątek obsługa
 obszar roboczy, 982, 983, 988
 odbiorca, 276
 odpytywanie, 226, 240
 okno, 710
 dialogowe, 710, 780, 977
 modalne, 780
 niemodalne, 780
 wybieranie plików, 784
 Frame obsługa zdarzeń, 716
 najwyższego poziomu, 713
 potomne, 713
 przewijane, 937
 tworzenie, 714, 720
 tytuł, 714

wymiary, 714
 wysokopoziomowe, 764
 zamykanie, 714
 OOP, *Patrz:* programowanie obiektowe
 opakowywanie, 260
 automatyczne, 260, 261, 263, 264, 450
 w wyrażeniu, 262
 operacja
 atomowa, 881
 redukcji zmienna, 857
 operator, 86
 !, 95
 !=, 94, 95
 %, 83, 84
 %=, 83
 &&, 95
 &, 87, 88, 95
 &=, 87, 95
 *, 83
 *=, 83
 --, 83, 86
 ., 124
 /, 83
 /=: 83
 ?, 97
 ?., 95
 ^, 87, 88, 89, 95
 ^=: 87, 95
 |, 87, 88, 89, 95
 ||, 95
 |=, 87, 93, 95
 ~, 87, 88
 +, 54, 83, 158
 ++, 83, 86
 +=, 83, 85
 <, 94
 <<, 87, 90
 <<=: 87
 <=: 94
 -=, 83
 =, 96
 ==, 94, 95, 384
 >, 94
 ->, *Patrz:* operator lambda
 >=: 94
 >>, 87, 91, 92
 >>=: 87, 93
 >>>, 87, 92
 >>>=: 87

arytmetyczny, 83, 84
 z przypisaniem, 85
 bitowy, 87
 z przypisaniem, 93
 dekrementacji,
 Patrz: operator --
 diamentu, 343
 iloczynu bitowego, 88
 inkrementacji,
 Patrz: operator ++
 instanceof, 301, 303, 340
 kolejność, 97, 98
 kropki, *Patrz:* operator .
 lambda, 350
 logiczny, 95, 96
 logiki bitowej, 88
 modulo, *Patrz:* operator %
 negacji bitowej, 88
 new, 75, 126, 127, 252
 przesunięcia
 bez znaku,
 Patrz: operator >>>
 w lewo, *Patrz:* operator <<
 w prawo,
 Patrz: operator >>
 przypisania,
 Patrz: operator =,
 operator +=
 relacji, 94
 rombu, 343
 strzałki, *Patrz:* operator
 lambda
 sumy bitowej, 89
 modulo 2, 89
 trójargumentowy,
 Patrz: operator ?

P

pakiet, 149, 187, 188
 Concurrency Utilities, 853
 hierarchia, 188
 importowanie, 192
 java.applet, 298, 663, 880
 java.awt, 683, 710, 880, 915
 java.awt.color, 880
 java.awt.datatransfer, 880
 java.awt.dnd, 880
 java.awt.event, 683, 685, 686, 696, 880, 914, 915
 java.awt.font, 880
 java.awt.geom, 880
 java.awt.im, 880
 java.awt.im.spi, 880
 java.awt.image, 787, 796, 810, 880
 java.awt.image.renderable, 880
 java.awt.print, 880
 java.beans, 880, 1070, 1071, 1072
 java.beans.beancontext, 880
 java.io, 284, 291, 569, 880
 Java.io, 570
 java.lang, 219, 274, 286, 291, 309, 399, 444, 880
 java.lang.annotation, 265, 274, 444, 880
 java.lang.instrument, 444, 880
 java.lang.invoke, 444, 880
 java.lang.management, 444, 880
 java.lang.ref, 444, 445, 880
 java.lang.reflect, 266, 270, 444, 445, 879, 880, 888
 java.math, 880
 java.net, 645, 880
 java.nio, 284, 569, 611, 880
 java.nio.channels, 611, 613, 615, 880
 java.nio.channels.spi, 611, 880
 java.nio.charset, 611, 614, 880
 java.nio.charset.spi, 611, 880
 java.nio.file, 573, 611, 615, 620, 880
 java.nio.file.attribute, 611, 615, 880
 java.nio.file.spi, 611, 615, 880
 java.rmi, 879, 880
 java.rmi.activation, 880
 java.rmi.dgc, 880
 java.rmi.registry, 880
 java.scene.canvas, 994
 java.security, 880
 java.security.acl, 881
 java.security.cert, 881
 java.security.interfaces, 881
 java.security.spec, 881

- pakiet
 - java.sql, 881
 - java.text, 879, 881, 894
 - java.text.spi, 881
 - java.time, 521, 879, 881, 897, 898
 - java.time.chrono, 881, 897
 - java.time.format, 881, 897, 899
 - java.time.temporal, 881, 897
 - java.time.zone, 881, 897
 - java.util, 447, 513, 683, 685, 881
 - java.util.concurrent, 564, 812, 834, 839, 881
 - java.util.concurrent.atomic, 564, 812, 813, 838, 881
 - java.util.concurrent.locks, 564, 812, 813, 835, 836, 881
 - java.util.function, 42, 372, 471, 564, 872, 881
 - java.util.jar, 564, 567, 881
 - java.util.logging, 564, 567, 881
 - java.util.prefs, 564, 567, 881
 - java.util.regex, 564, 567, 879, 881
 - java.util.spi, 564, 567, 881
 - java.util.stream, 42, 564, 567, 855, 870, 881
 - java.util.zip, 564, 567, 881
 - javafx.application, 982, 983, 986, 1049
 - javafx.collection, 1017
 - javafx.collections, 988
 - javafx.event, 990, 991
 - javafx.scene, 982, 986
 - javafx.scene.control, 991, 999, 1008, 1041
 - javafx.scene.effect, 1034, 1064
 - javafx.scene.image, 999
 - javafx.scene.input, 1050
 - javafx.scene.layout, 982, 983, 986
 - javafx.scene.paint, 996
 - javafx.scene.paint.Color, 1034
 - javafx.scene.shape, 998
 - javafx.scene.text, 996
 - javafx.scene.transform, 1035, 1064
 - javafx.stage, 982, 986
 - javax.imageio, 810
 - javax.servlet, 1080, 1081, 1085
 - javax.servlet.http, 1081, 1086
 - javax.swing, 910, 911, 923
 - javax.swing.border, 910
 - javax.swing.colorchooser, 910
 - javax.swing.event, 910, 914, 938, 943
 - javax.swing.filechooser, 910
 - javax.swing.plaf, 910
 - javax.swing.plaf.basic, 910
 - javax.swing.plaf.metal, 910
 - javax.swing.plaf.multi, 910
 - javax.swing.plaf.nimbus, 910
 - javax.swing.plaf.synth, 910
 - javax.swing.table, 910, 946
 - javax.swing.text, 910
 - javax.swing.text.html, 910
 - javax.swing.text.html.parser, 910
 - javax.swing.text.rtf, 910
 - javax.swing.tree, 910
 - javax.swing.undo, 910
 - specyfikacja, 440
 - ścieżka wyszukiwania, 188
 - tytuł, 440
 - wersja, 440
 - zasobów, 560
 - pamięć
 - odzyskiwanie, *Patrz:* mechanizm odzyskiwania pamięci
 - zarządzanie, 416
 - panel, 909
 - BorderPane, 983
 - FlowPane, 983
 - GridPane, 983
 - podzielony na zakładki, 934
 - szklany, 909, 910
 - treści, 909, 910
 - wielowarstwowy, 909, 910
 - para klucz-wartość, 449, 474, 476, 504, 505, 561
 - parallel programming, *Patrz:* programowanie równoległe
 - parametr, 52, 128, 131, 134
 - args, 52
 - obiektowy, 143, 144
 - przekazywanie przez referencję, 145
 - wartość, 145
 - pasek
 - menu, 711, 713, 741, 775, 949
 - tworzenie, 775
 - narzędzi, 949, 964, 965, 1042, 1057, 1058
 - niezadokowany, 965
 - przewijania, 687, 711, 742, 755, 936, 1020, 1026
 - obsługa zdarzeń, 756
 - tworzenie, 755
 - stanu, 674
 - tytułu, 713
- Payne Jonathan, 34
- pętla, 114
 - do-while, 105, 107, 109
 - for, 55, 57, 105, 109, 110
 - odmiana, 111
 - rozszerzona, 115
 - zagnieżdżona, 116
 - for-each, 109, 112, 114, 115, 116, 254, 443, 450, 469, 474
 - opuszczanie, 117
 - while, 105, 106
 - zagnieżdżanie, 116
 - zdarzeń z odpytywaniem, 226, 240
 - zmienna sterująca, 109
 - deklaracja, 109
- PLAF, 907
- plik, 570
 - .class, 51
 - .java, 50
 - archiwalny, 619
 - atrybuty, 619
 - cookie, 658, 1089
 - format, *Patrz:* format GZIP, 567
 - HTML, 664, 676
 - JAR, 881
 - Java Archive, 567
 - kopiowanie, 628
 - mapowanie na bufor, 640
 - multimedialny, 679

- plik
 - odczyt, 290, 638
 - za pośrednictwem kanału, 622
 - odwzorowanie na bufor, 613
 - systemowy, 619
 - ścieżka, 784
 - ukryty, 617, 619
 - uprawnienia dostępu, 620
 - właściciel, 620
 - wskaźnik, 594
 - wykonywalny, 617
 - XML, 1072
 - zamykanie, 291
 - automatyczne, 295
 - zapis, 290, 626, 641
 - ZIP, 567, 881
 - źródłowy, 50
 - pluggable look and feel, *Patrz:* PLAF
 - płatyczna wielojęzyczna
 - podstawowa, *Patrz:* BMP
 - poczta elektroniczna, 646
 - podklasa, *Patrz:* klasa potomna
 - sparametryzowana, 339
 - podmapa, 477
 - pole
 - tekstowe, 698, 711, 742, 758
 - obsługa zdarzeń, 759
 - tekst sformatowany, 977
 - tworzenie, 758
 - wielowierszowe, 760
 - ze strzałkami do zmiany wartości, 977
 - wyboru, 710, 742, 747, 931, 960, 1014, 1042, 1051, 1052
 - obsługa zdarzeń, 748
 - trzystanowe, 1014, 1016
 - tworzenie, 748
 - polimorfizm, 46, 48, 49, 50, 139, 141, 179, 180, 196
 - dynamiczny, 180
 - port, 645, 646, 650, 661
 - procedura pośrednicząca, 893
 - proces, 225
 - hermetyzacja, 415
 - komunikacja, *Patrz:* komunikacja międzyprocesorowa
 - zakleszczenie, *Patrz:* zakleszczenie
 - profil kompaktowy, 313
 - program
 - javadoc, 1099, 1103
 - kompilacja, 50, 51
 - sieciowy, *Patrz:* aplet
 - uruchamianie, 50
 - programowanie
 - funkcyjne, 42
 - obiektowe, 33, 45, 46, 165
 - równoległe, 42, 226, 448, 452, 470, 472, 839, 848
 - asynchroniczne
 - wykonywanie zadania, 850
 - poziom, 846, 848
 - wydajność zadań, 845, 846, 848
 - strukturalne, 32, 33
 - wielowątkowe, *Patrz:* wielowątkowość
 - współbieżne, 564, 811
 - narzędzia, 811
 - projekt Coin, 41, 42
 - protokół, 653, 1084
 - bezstanowy, 1095
 - FTP, 646
 - HTTP, 646, 1078, 1090, 1092, 1095
 - IP, 645
 - IPv4, 646, 647
 - IPv6, 646, 647
 - JNLP, 664
 - TCP, 645
 - TCP/IP, 39, 646, 659
 - Telnet, 646
 - UDP, 646, 659
 - przeglądarka, 52, 665, 1077
 - apletów, *Patrz:* appletviewer
 - pasek stanu, *Patrz:* pasek stanu
 - przełącznik, 1005
 - stan, 929
 - przenośność, 36, 37
 - przesłanie, 173, 178
 - przestrzeń nazw, 187
 - domyślna, 187
 - przycisk, 710, 741, 742, 744, 916, 926, 927, 991
 - obraz, 1003
 - obsługa zdarzeń, 744
 - opcji, 749, 932, 933, 960, 1008, 1011, 1042, 1051
 - przełącznika, 929, *Patrz:* przełącznik
 - tworzenie, 744
 - pula wspólna, 841
 - punkt kodowy, 413
 - pushback, *Patrz:* zwracania
- ## R
- referencja
 - do konstruktora, 368, 369, 372
 - do metody, 362, 366, 367
 - do metody instancyjnej, 363
 - do metody statycznej, 362
 - Object, 316
 - refleksja, 266, 445, 879, 888
 - rekurencja, 147, 148
 - Remote Method Invocation, *Patrz:* metoda wywoływania zdalnego
 - resource starvation, *Patrz:* głodzenie zasobów
 - retained mode, *Patrz:* tryb zachowany
 - RGB, 726, 727
 - Richards Martin, 32
 - Ritchie Dennis, 32
 - RMI, *Patrz:* metoda wywoływania zdalnego
 - rodzic, 983
 - rozkład Gaussa, 529
 - rozpakowywanie, 260
 - automatyczne, 264, 450
 - rzutowanie, 342
- ## S
- SAM type, *Patrz:* typ SAM
 - scena, 982, 988
 - scene, *Patrz:* obszar roboczy
 - selektor, 615
 - semafor, 812, 813, *Patrz:* monitor
 - licznik, 813, 814
 - Separable Model, *Patrz:* architektura model-delegacja
 - separator, 59, 513
 - (), 59
 - „, 59
 - ., 59

- separator
 - ::, 59
 - ;, 59
 - [], 59
 - {}, 59
 - kropki, *Patrz:* operator .
 - serializacja, 605, 606, 608, 893, 1071
 - serwer, 646, 648, 893, 1077
 - serwletów, 1078
 - Glassfish, 1078
 - Tomcat, 1078, 1079, 1081
 - serwlet, 37, 1077
 - cykl życia, 1078, 1083
 - dane konfiguracyjne, 1082
 - dziennik, 1083
 - kontekst, 1082
 - kontener, *Patrz:* serwer
 - serwletów
 - nazwa, 1082
 - parametry, 1085
 - serwer, *Patrz:* serwer
 - serwletów
 - środowisko, 1083
 - tworzenie, 1080, 1081
 - sesja, 1087, 1088
 - śledzenie, 1095
 - Sheridan Mike, 33
 - skaner, 513
 - składowa statyczna, 152
 - skrót, 462
 - skrót klawiszowy, 957, 1049
 - słownik, 504
 - słowo kluczowe, 59
 - abstract, 182, 184
 - assert, 283, 306
 - catch, 209, 211
 - class, 52, 123
 - const, 59
 - enum, 251, 441
 - extends, 165, 202
 - final, 153, 184, 361
 - finally, 209, 217, 297
 - goto, 59
 - import, 309
 - instanceof, 283
 - int, 54
 - interface, 187, 194, 265
 - native, 283, 303
 - public, 52
 - static, 52, 139, 152, 309, 606
 - strictfp, 283
 - super, 170, 173
 - synchronized, 238, 239, 811
 - this, 135, 136, 276, 311
 - throw, 209, 216
 - throws, 209, *Patrz też:*
 - klauzula throws
 - transient, 283, 606
 - try, 209, 210, 211
 - void, 52, 133
 - volatile, 283
 - socket, *Patrz:* gniazdo
 - specyfikator
 - formatu, 537, 539, 540, 542, 547
 - indeks argumentu, 548
 - pisany wielką literą, 547
 - grupowania, 547
 - konwersji formatu, 537
 - minimalnej szerokości pola, 543
 - precyzji, 544
 - spliterator, 448, 451, 470, 472, 497, 874
 - cecha, 472
 - stała, 58, 66
 - całkowitoliczbowa, 66, 67
 - deklarowanie, 153
 - E, 429
 - klasowa, 268
 - logiczna, 68
 - łańcuchowa, *Patrz:* łańcuch znaków
 - PI, 429
 - TYPE, 415
 - własnego typu, 252
 - wyliczeniowa, 252, 255
 - wartość kolejności, 256
 - zmiennoprzecinkowa, 67, 68
 - stos, 48, 137, 150, 198, 434, 439, 503
 - o dynamicznie
 - zmieniającym się
 - rozmiarze, 199
 - strategia dziel i zwyciężaj, 841, 843
 - Stroustrup Bjarne, 33
 - strumień, 284, 422, 508, 510, 569, 621, 855
 - bajtów, 284, 285, 570, 578
 - buforowanie, 586
 - filtrowanie, 585, 869
 - iterator, 856, 873
 - liczba elementów, 857
 - nieuporządkowany, 856
 - odwzorowanie elementów, 866
 - operacja
 - bezstanowa, 858
 - kończąca, 862
 - redukcji, 862
 - uwzględniająca stan, 858
 - referencja, 856
 - równoległy, 856, 864, 866
 - sekwencyjny, 856, 866
 - System.err, 286
 - System.in, 286
 - System.out, 286, 289
 - tworzenie, 857, 858
 - wejściowy, 579, 598, 650
 - dane zwracane, 588
 - przeszukiwanie, 601
 - wyjściowy, 53, 650
 - zamykanie, 576, 856
 - znaków, 284, 285, 570, 578, 595, 596
- supplemental character, *Patrz:* znak uzupełniający
- surogat
 - bardziej znaczący, 413
 - mniej znaczący, 413
- suwak, 1064
- synchronizator, 812
- system
 - NIO, *Patrz:* NIO
 - wielordzeniowy, 226

Ś

- ścieżka, 615, 616, 617, 618, 621, 625, 631, 632, 633
 - względna, 623
- środowisko, 424, 444

T

- tabela, 945
- tablica, 52, 75, 112, 114, 495
 - deklarowanie, 80
 - dynamiczna, 448, 459, 460
 - inicjalizacja, 76
 - jednowymiarowa, 75

- kopiowanie, 423, 495
- length, *Patrz:* tablica liczba elementów
- liczba elementów, 154
- mieszająca, 448, 462, 480, 505
- porównywanie, 496
- sortowanie, 497
- tablic, *Patrz:* tablica wielowymiarowa
- tworzenie, 75, 76
- typ sparametryzowany, 347
- wielowymiarowa, 77, 114
 - deklarowanie, 77
 - inicjalizacja, 79
 - nieregularna, 79
 - zagnieżdżona, 498
- target type, *Patrz:* typ docelowy
- tekst, 694, 698, 995
 - czcionka, *Patrz:* czcionka
 - formatowanie, 737, 879
 - wyśrodkowanie, 737
 - wyświetlanie, 735
- ThreadGroup, 399
- toggle button, *Patrz:* przycisk przełącznika
- token, 513, 514, 551, 887
- Tomcat, 1078, 1079
 - uruchamianie, 1081
- tooltip, *Patrz:* etykieta ekranowa
- Transmission Control Protocol, *Patrz:* protokół TCP
- TreeSet, 447
- trwałość, 1071
- tryb
 - rysowania, 728
 - zachowania, 994
- typ
 - bezpieczeństwo, 449
 - boolean, 61, 66, 73, 94, 259
 - opakowanie, 263
 - byte, 61, 62, 67, 73, 87, 400
 - Byte, 259
 - całkowitoliczbowy, 54, 61, 62, 64, 65, 73, 87
 - rozmiar, 62
 - char, 61, 64, 65, 73, 87, 216, 259, 378, 413
 - opakowanie, 263
 - Character, 259
 - docelowy, 350
 - double, 61, 64, 68, 400
 - Double, 259
 - float, 61, 64, 68, 259, 400
 - int, 61, 62, 63, 65, 87, 216, 259, 400
 - interfejsu, 324
 - Iterator, 873
 - klasowy, 146, 254
 - konwersja, 72, 74
 - automatyczna, 72
 - rozszerzająca, 72
 - zawężająca, 73
 - logiczny, 61, 66, 68
 - long, 61, 62, 63, 67, 73, 87, 259
 - numeryczny, 73
 - opakowanie, 259
 - ograniczony, 322, 323
 - opakowanie, 259, 399
 - parametr, 317, 325
 - podstawowy, *Patrz:* typ prosty
 - prosty, 61, 62, 127, 216, 264
 - opakowanie, 258, 260, 261, 262, 399
 - rozszerzanie, 74
 - rzutowanie, 72, 73, 301
 - obcięcie, 73
 - SAM, 350
 - short, 61, 62, 63, 67, 73, 87, 259, 400
 - sparametryzowany, 185, 266, 281, 315, 319, 343, 366, 449, 504, 505
 - argument typu, 319
 - argument wieloznaczny, 324, 325, 327
 - argument wieloznaczny ograniczony, 328
 - błąd niejednoznaczności, 346
 - ograniczenia, 347, 348
 - tablica, 347
 - znoszenie, 344, 346
 - Spliterator, 873, 875
 - String, 80, 377
 - surowy, 335, 336
 - Throwable, 216
 - TimeUnit, 834
 - uogólniony, *Patrz:* typ sparametryzowany
 - void, 415
 - wartość, 442
 - wnioskowanie, 343
 - wyliczeniowy, 251, 254, *Patrz też:* wyliczenie
 - zgodność, 72
 - zmiennoprzecinkowy, 61, 63, 73
 - znakowy, 61, 64, 68

U

 - UI delegate, *Patrz:* delegacja interfejsu użytkownika
 - Unicode, 64, 65, 68, 412, 595
 - 32-bitowe, 413, 414
 - titlecase, 412
 - urządzenie graficzne, 711
 - User Datagram Protocol, *Patrz:* protokół UDP
 - usługa
 - DNS, 646
 - whois, 646, 650, 652
 - ustawienia
 - regionalne, 527, 528, 559
 - użytkownika, 567
 - uwierzytelnianie, 1087

V

 - van Hoff Arthur, 34
 - varargs, 160, 161
 - viewport widok, 936
 - vItemEvent, 695

W

 - waluta, 535
 - Warth Chris, 33
 - wartość kolejności, 256
 - wątek, 225, 226, 433, 434, 534
 - activeCount, 436
 - activeGroupCount, 436
 - aktywny, 434
 - aplikacji, 988
 - blokada wzajemna, *Patrz:* zakleszczenie
 - blokowanie, 226
 - checkAccess, 436
 - demon, 435, 842
 - demonem, 434

- wątek
 - destroy, 436
 - enumerate, 436
 - falszywe budzenie, 240
 - getMaxPriority, 436
 - getName, 436
 - getParent, 436
 - główny, 228
 - grupa, 229, 435, 436
 - isDestroyed, 436
 - komunikacja, 228,
 - Patrz:* komunikacja międzywątkowa
 - nazwa, 434, 435
 - oczekujący, 227, 247, 248, 424, 533
 - parentOf, 436
 - priorytet, 227, 236, 434, 435
 - przełączenie kontekstu, 227
 - rozdzielający zdarzenia, 913, 917
 - setDaemon, 436
 - setMaxPriority, 436
 - stan, 247, 248
 - synchronizacja, 227, 237, 238, 239
 - szeregowanie, 238
 - tworzenie, 230, 232, 233
 - uncaughtException, 436
 - uruchomieniowy, 988
 - wykonywany, 227, 247, 248
 - wyścig, 238
 - wywłaszczanie, 227
 - wznawianie, 227, 245
 - zablokowany, 227, 247, 248
 - zakleszczenie,
 - Patrz:* zakleszczenie
 - zakończony, 227, 234, 247, 248
 - zawieszony, 227, 245, 247, 248, 819
 - zmienna lokalna, *Patrz:* zmienna lokalna wątku
- wejście standardowe, 54
- wejście-wyjście, 283
 - obsługa błędów, 292
- wektor, 500, 501
- węzeł, 983, 1029
 - gałęzi, 983
 - głównego poziomu, 983
 - końcowy, *Patrz:* liść
 - potomny, 983
- widok, 936
 - kolekcji, 449
- wielowątkowość, 39, 225, 226, 228, 240, 433, 811, 839
 - synchronizacja, 227, 237, 238, 239, 811, 812, 813, 821, 823
 - system wieloprocesorowy, 839
 - zmienne, 813
- wielozadaniowość, 225
 - bazująca na
 - procesach, 225
 - wątkach, 225
 - z wywłaszczaniem, 227
- wiersza poleceń argument, 159
- wirus, 36
- właściwość, 1068
 - indeksowana, 1069, 1072
 - ograniczona
 - bound, 1070, 1071
 - constrained, 1070, 1071, 1072
 - prosta, 1068
- work-stealing, *Patrz:* klasa ForkJoinPool wykradanie zadań
- wskaźnik, 81, 126
- współbieżność, 881
- wyciek pamięci, 291
- wydobywanie automatyczne, 260
- wyjątek, 209, 439, 1100
 - ArithmeticException, 211, 214, 219, 222, 223, 430
 - ArrayIndexOutOfBoundsException
 - ↳ Exception, 213, 219, 223, 496
 - ArrayStoreException, 219
 - AssertionError, 306
 - bez opisu, 221
 - ClassCastException, 219, 451, 453, 454, 455, 457, 475, 477, 479, 483, 491, 494, 495, 497
 - ClassNotFoundException, 220, 608
 - CloneNotSupportedException
 - ↳ Exception, 220, 425, 441
 - EmptyStackException, 503
 - EnumConstantNotPresentException, 219
 - FileNotFoundException, 290, 576, 579
 - finalne zgłoszenie dalej, 223
 - HeadlessException, 743
 - IllegalAccessException, 220
 - IllegalArgumentException, 219, 451, 453, 455, 457, 466, 475, 477, 496
 - IllegalFormatException, 538
 - IllegalMonitorStateException, 219
 - IllegalStateException, 219, 451, 455, 457, 458, 882, 1084
 - IllegalThreadStateException, 219
 - IndexOutOfBoundsException
 - ↳ Exception, 219, 453
 - InstantiationException, 220
 - InterruptedException, 220, 229
 - IntrospectionException, 1072
 - InvalidPathException, 619
 - IOException, 604
 - IOException, 286, 291, 538, 576, 578, 579, 595, 596, 597, 602, 608, 1087
 - łańcuch, 222
 - MalformedURLException, 653
 - MissingResourceException, 560
 - NegativeArraySizeException, 219
 - nieprzechwycony, 210, 434, 436
 - nieweryfikowany, 219
 - NoSuchElementException, 454, 455, 456, 457, 467, 468, 477, 499, 518
 - NoSuchFieldException, 220
 - NoSuchMethodException, 220
 - NotSerializableException, 609
 - NullPointerException, 216, 219, 451, 453, 454, 455, 457, 466, 475, 477, 495, 496, 517, 560, 590
 - NumberFormatException, 219

- obsługa, 116, 209
 - opis, 212
 - PatternSyntaxException, 883
 - PropertyVetoException, 1070
 - przechwytywanie
 - wielokrotne, 209, 223
 - ReflectiveOperation
 - ↳Exception, 220
 - RemoteException, 891
 - RuntimeException, 219
 - SecurityException, 219, 290, 421, 576
 - ServletException, 1085
 - SocketException, 659
 - StringIndexOutOfBoundsException
 - ↳Exception, 219
 - systemowy, 209
 - TooManyListenersException, 1069
 - TypeNotPresentException, 219
 - UnavailableException, 1082, 1085
 - UnknownHostException, 648
 - UnsupportedOperation
 - ↳Exception, 219, 450, 451, 453, 475, 494, 620
 - weryfikowany, 219, 220
 - własny, 216, 219
 - wyrażenie lambda, *Patrz:*
 - wyrażenie lambda wyjątek
 - wykładnik binarny, 68
 - wyliczenie, 256, 257, 491, 507,
 - Patrz też:* typ wyliczeniowy
 - ElementType, 444
 - klucz, 466
 - RetentionPolicy, 444
 - stała, *Patrz:* stała wyliczeniowa
 - wyrażenie, 74, 262
 - try-with-resources, 223
 - lambda, 42, 349, 351, 352, 747, 990
 - blokowe, 354, 355
 - ciało, 350
 - ciało blokowe, 354
 - ciało wyrażeniowe, 354
 - jako argument, 357
 - wyjątek, 360
 - regularne, 551, 567, 879, 881, 883, 888
 - tworzenie, 882
 - wzorzec, 879
 - sterująca, 102, 104
 - try, 210
 - try-with-resources,
 - Patrz:* instrukcja try-with-resources
 - warunkowe, 66
 - wzorzec projektowy, 1070
 - dla zdarzeń, *Patrz:* zdarzenie wzorzec projektowy
- Y**
- Yellin Frank, 34
- Z**
- zakleszczenie, 244
 - zarządzanie zasobami
 - automatyczne, *Patrz:* ARM
 - zasada rozszerzania typów,
 - Patrz:* typ rozszerzanie
 - zatrząsk, 818
 - zbiór, 454, 483
 - zdarzenie, 667, 683
 - ActionEvent, 686, 696, 744, 747, 759, 916, 927, 950, 991, 1006, 1042
 - AdjustmentEvent, 687, 696, 756
 - AWT, 710
 - ChangeListener, 1010
 - ComponentEvent, 688
 - ContainerEvent, 688
 - delegowanie, 684, 699, 914, 990, 1069
 - filtr, 991
 - FocusEvent, 689
 - grupowe rozgłaszanie, 684
 - ItemEvent, 690, 748, 751, 929
 - KEY_PRESSED, 701
 - KEY_TYPED, 701
 - KeyEvent, 691
 - kolejkowanie, 710
 - kontrolki, 742
 - ListSelectionEvent, 938
 - łańcucha przydzielania,
 - Patrz:* łańcucha przydzielania zdarzeń
 - MenuDragMouseEvent, 951
 - MenuKeyEvent, 951
 - MouseEvent, 691
 - obsługa, 683
 - delegowanie, *Patrz:* zdarzenie delegowanie w grupie, 1010
 - w oknie typu Frame, 716
 - PopupMenuEvent, 951
 - propagacja, 991
 - PropertyChangeEvent, 1070
 - rozgłaszanie, 1069
 - TableModelEvent, 946
 - TreeExpansionEvent, 943
 - TreeModelEvent, 943
 - TreeSelectionEvent, 943
 - wysyłanie jednostkowe, 685
 - wzorzec projektowy, 1069
 - związane z klawiaturą, 701
 - źródło, 684, 695, 991
 - zmienna, 53, 69, 127
 - czas życia, 70, 71
 - członkowska, *Patrz:* zmienna składowa
 - deklaracja, 69, 70, 71
 - egzemplarza, *Patrz:* zmienna składowa final, 251
 - inicjalizacja, 70, 71
 - dynamiczna, 70
 - interfejsu, 201
 - iteracyjna, 112
 - lokalna, 135, 136
 - sfinalizowana, 361
 - wątku, 439
 - nazwa, 58
 - przechwytywanie, 361
 - referencyjna, 127, 128, 169, 196
 - interfejsu, 200
 - składowa, 47, 124
 - ukrywanie, 135, 136
 - sterująca pętlą, *Patrz:* pętla zmienna sterująca
 - środowiskowa, 424
 - CLASSPATH, 188
 - tablicowa, 75
 - transient, 301

zmienna	%d, 538, 539, 540	-=, 83
volatile, 301	%e, 539, 540, 544, 547	==, 94, 95, 384
zasięg, 70, 71	%f, 537, 538, 539, 540, 543,	>, 94
globalny, 70	544, 547	->, 350
lokalny, 70	%g, 539, 540, 544	>=, 94
zagnieżdżanie, 71	%h, 539	>>, 87, 91, 92
znacznik, 852	%n, 539, 542	>>=, 87, 93
@author, 1100	%o, 539, 547	>>>, 87, 92
@code, 1100	%s, 538, 539, 544	>>>=, 87
@deprecated, 1099, 1100	%t, 539, 540	apostrof, 68
@docRoot, 1100, 1101	%tM, 541	\b, 69
@exception, 1100, 1101	%x, 539, 547	biały, 58, 389
@inheritDoc, 1100, 1101	&, 87, 88, 95	dwukropek, 118
@link, 1100, 1101	&&, 95	\f, 69
@linkplain, 1100, 1101	&=, 87, 95	klasa, <i>Patrz:</i> klasa znaków
@literal, 1100, 1101	(), 59	kropki, <i>Patrz:</i> znak .
@param, 1100, 1101	*, 83, 634, 883	lewý ukośnik, 68
@return, 1100, 1101	*/, 51, 58, 1099	lustrzany, 412
@see, 1099, 1100, 1102	*=, 83	łańcuch, <i>Patrz:</i> łańcuch
@serial, 1100	„, 59	znaków
@serialData, 1100	., 59, 124, 188, 883	\n, 69, 542
@serialField, 1100, 1102	..., 161	'\n', 68
@since, 1099, 1100, 1102	/, 83, 571	nawias okrągły, 98
@throws, 1100, 1102	/*, 51, 1099	podkreślenia, 67
@value, 1100, 1102	/**, 58, 1099	przecinek, 110, 111
@version, 1100, 1103	//, 52, 1099	\r, 69
APPLET, 299, 664, 675, 676	/=, 83	symbol wieloznaczny, 634
formatu, 545	::, 59	\t, 69
img, 787	::, 53, 59	titlecase, 412
komentarza, 1099	?, 634, 883	\u, 68
kontekstu klienta, 1089	@, 265, 1099	uzupełniający, 413
kontekstu użytkownika,	\\, 69, 571	zastępczy, 883, 884
1093	^, 87, 88, 89, 95	zestaw, 614
OBJECT, 299, 664	^=, 87, 95	zwracania, 588
przyłączany, 1099	{, 52	
samodzielny, 1099	{}, 59	
znak	, 87, 88, 89, 95	
\, 68, 69	, 95	
!, 95	=, 87, 93, 95	żądanie, 1083, 1084
!=", 94, 95	}, 52	DELETE, 1090
", 69	~, 87, 88	GET, 1090
", 571	+, 54, 83, 158, 546, 883, 884	HEAD, 1090
#, 547	++, 83	HTTP, 1077, 1078
\$, 58	+=, 83	OPTIONS, 1090
%, 83, 84	<, 94	parametry, 1085
%- , 545	<?>, 266, 268	POST, 1090, 1092
%%, 539, 542	<<, 87, 90	PUT, 1090
%=, 83	<<=, 87	TRACE, 1090
%a, 539, 540	<=, 94	użytkownik, 1087, 1088
%b, 539	<>, 343	
%c, 539	=, 96	

Ż

żądanie, 1083, 1084
 DELETE, 1090
 GET, 1090
 HEAD, 1090
 HTTP, 1077, 1078
 OPTIONS, 1090
 parametry, 1085
 POST, 1090, 1092
 PUT, 1090
 TRACE, 1090
 użytkownik, 1087, 1088

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**

Kompletne źródło informacji na temat języka Java!

Od czasu swojej premiery w 1995 roku Java cieszy się niesłabnącą popularnością wśród programistów oraz pracodawców. Język ten znajduje zastosowanie wszędzie tam, gdzie wymagane są najwyższa wydajność, niezawodność i bezpieczeństwo. Ponadto każda kolejna wersja Javy wprowadza nowości, które sprawiają, że życie programisty staje się łatwiejsze, a tworzony kod bardziej przejrzysty.

Ukazanie się na rynku wersji ósmej tego języka jest świetną okazją do zapoznania się z najnowszym wydaniem cenionego podręcznika na jego temat, poprawionego oraz uzupełnionego o wiadomości o wprowadzonych nowościach. W trakcie lektury zdobędziesz informacje na temat składni Javy, typowych konstrukcji oraz najlepszych technik programowania. Ponadto poznasz platformę JavaFX i wyrażenia lambda oraz odkryjesz sposoby radzenia sobie z zagadnieniem wielowątkowości. Książka ta jest doskonałą i pasjonującą lekturą zarówno dla programistów języka Java, jak i osób chcących poznać ten język.

Dzięki tej książce:

- poznasz składnię języka Java
- zaznajomisz się z nowościami w Javie 8
- wykorzystasz potencjał platformy JavaFX
- opanujesz wyrażenia lambda
- nauczysz się tworzyć wydajne rozwiązania wielowątkowe

Herbert Schildt — były członek komitetu ANSI/ISO, który dokonał standaryzacji języka C++. Autor popularnych publikacji poświęconych programowaniu, głównie w językach C, C++, C# i Java. Jego książki, wykorzystywane przez profesjonalistów, szkoleniowców i studentów, rozeszły się w ponad 3 milionach egzemplarzy.

Helion

34787 numer katalogowy

księgarnia internetowa

<http://helion.pl>

zamówienia telefoniczne

☎ 0 801 339900

☎ 0 601 339900

Informatyka w najlepszym wydaniu

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

sięgnij po WIĘCEJ



KOD KORZYŚCI

ISBN 978-83-283-0812-1



9 788328 308121

cena: 179,00 zł

Oracle
Press™