

Zgodne z JDK 7

Java

Kompendium programisty

Wydanie VIII

Najbardziej aktualne źródło wiedzy o Javie!

**Helion****Herbert Schildt**

Tytuł oryginału: Java The Complete Reference, 8th Edition

Tłumaczenie: Mikołaj Szczepaniak

ISBN: 978-83-246-3767-6

© Helion 2012

All rights reserved

Original edition copyright © 2011 by McGraw-Hill Companies, Inc.

All rights reserved.

Polish edition copyright © 2012 by HELION SA

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 jakiejkolwiek 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.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:

<ftp://ftp.helion.pl/przyklady/javkp8.zip>

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/javkp8>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

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

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

Spis treści

O autorze	23
O redaktorze merytorycznym	23
Przedmowa	25
Część I Język Java	29
Rozdział 1. Historia i ewolucja języka Java	31
Rodowód Javy	31
Narodziny nowoczesnego języka — C	32
Język C++ — następny krok	33
Podwaliny języka Java	34
Powstanie języka Java	34
Powiązanie z językiem C#	36
Jak język Java zmienił internet	36
Aplety Javy	37
Bezpieczeństwo	37
Przenośność	38
Magia języka Java — kod bajtowy	38
Serwlety — Java po stronie serwera	39
Hasła języka Java	40
Prostota	40
Obiektowość	40
Niezawodność	41
Wielowątkowość	41
Neutralność architektury	42
Interpretowalność i wysoka wydajność	42
Rozproszenie	42
Dynamika	42
Ewolucja Javy	42
Java SE 7	44
Kultura innowacji	46
Rozdział 2. Podstawy języka Java	47
Programowanie obiektowe	47
Dwa paradygmaty	47
Abstrakcja	48
Trzy zasady programowania obiektowego	48

Pierwszy przykładowy program	53
Wpisanie kodu programu	54
Kompilacja programów	54
Bliższe spojrzenie na pierwszy przykładowy program	55
Drugi prosty program	57
Dwa wyrażenia sterujące	59
Wyrażenie if	59
Pętla for	60
Bloki kodu	61
Kwestie składniowe	63
Znaki białe	63
Identyfikatory	63
Stałe	63
Komentarze	64
Separatory	64
Słowa kluczowe języka Java	64
Biblioteki klas Javy	65
Rozdział 3. Typy danych, zmienne i tablice	67
Java to język ze ścisłą kontrolą typów	67
Typy proste	67
Typy całkowitoliczbowe	68
Typ byte	69
Typ short	69
Typ int	69
Typ long	69
Typy zmiennoprzecinkowe	70
Typ float	71
Typ double	71
Typ znakowy	71
Typ logiczny	73
Bliższe spojrzenie na stałe	74
Stałe całkowitoliczbowe	74
Stałe zmiennoprzecinkowe	75
Stałe logiczne	76
Stałe znakowe	76
Stałe łańcuchowe	77
Zmienne	77
Deklaracja zmiennej	78
Inicjalizacja dynamiczna	78
Zasięg i czas życia zmiennych	79
Konwersja typów i rzutowanie	81
Automatyczna konwersja typów	81
Rzutowanie niezgodnych typów	82
Automatyczne rozszerzanie typów w wyrażeniach	83
Zasady rozszerzania typu	84
Tablice	85
Tablice jednowymiarowe	85
Tablice wielowymiarowe	87
Alternatywna składnia deklaracji tablicy	91
Kilka słów o łańcuchach	92
Uwaga dla programistów języka C lub C++ na temat wskaźników	92

Rozdział 4. Operatory	93
Operatory arytmetyczne	93
Podstawowe operatory arytmetyczne	94
Operator reszty z dzielenia	95
Operatory arytmetyczne z przypisaniem	95
Inkrementacja i dekrementacja	96
Operatory bitowe	98
Logiczne operatory bitowe	99
Przesunięcie w lewo	101
Przesunięcie w prawo	103
Przesunięcie w prawo bez znaku	104
Operatory bitowe z przypisaniem	105
Operatory relacji	106
Operatory logiczne	107
Operatory logiczne ze skracaniem	109
Operator przypisania	109
Operator ?	110
Kolejność wykonywania operatorów	111
Stosowanie nawiasów okrągłych	112
Rozdział 5. Wyrażenia sterujące	113
Instrukcje wyboru	113
Konstrukcja if	113
Konstrukcja switch	116
Instrukcje iteracyjne	121
Pętla while	121
Pętla do-while	122
Pętla for	125
Wersja for-each pętli for	128
Pętle zagnieżdżone	133
Instrukcje skoku	134
Instrukcja break	134
Instrukcja continue	138
Instrukcja return	139
Rozdział 6. Wprowadzenie do klas	141
Klasy	141
Ogólna postać klasy	141
Prosta klasa	142
Deklarowanie obiektów	145
Bliższe spojrzenie na operator new	146
Przypisywanie zmiennych referencyjnych do obiektów	147
Wprowadzenie do metod	148
Dodanie metody do klasy Box	148
Zwracanie wartości	150
Dodanie metody przyjmującej parametry	151
Konstruktor	153
Konstruktor sparametryzowany	155
Słowo kluczowe this	156
Ukrywanie zmiennych składowych	157
Mechanizm odzyskiwania pamięci	157
Metoda finalize()	158
Klasa stosu	158

Rozdział 7. Dokładniejsze omówienie metod i klas	161
Przeciążanie metod	161
Przeciążanie konstruktorów	164
Obiekty jako parametry	166
Dokładniejsze omówienie przekazywania argumentów	168
Zwracanie obiektów	170
Rekurencja	171
Wprowadzenie do kontroli dostępu	173
Składowe statyczne	176
Słowo kluczowe final	178
Powtórka z tablic	179
Klasy zagnieżdżone i klasy wewnętrzne	181
Omówienie klasy String	183
Wykorzystanie argumentów wiersza poleceń	186
Zmienna liczba argumentów	187
Przeciążanie metod o zmiennej liczbie argumentów	190
Zmienna liczba argumentów i niejednoznaczności	191
Rozdział 8. Dziedziczenie	193
Podstawy dziedziczenia	193
Dostęp do składowych a dziedziczenie	195
Bardziej praktyczny przykład	196
Zmienna klasy bazowej może zawierać referencję do obiektu podklasy	198
Słowo kluczowe super	199
Wykorzystanie słowa kluczowego super do wywołania konstruktora klasy bazowej	199
Drugie zastosowanie słowa kluczowego super	202
Tworzenie hierarchii wielopoziomowej	203
Kiedy dochodzi do wywołania konstruktorów?	206
Przesłanianie metod	207
Dynamiczne przydzielanie metod	209
Dlaczego warto przesłaniać metody?	211
Zastosowanie przesłaniania metod	211
Klasy abstrakcyjne	213
Słowo kluczowe final i dziedziczenie	216
Słowo kluczowe final zapobiega przesłanianiu	216
Słowo kluczowe final zapobiega dziedziczeniu	216
Klasa Object	217
Rozdział 9. Pakiety i interfejsy	219
Pakiety	219
Definiowanie pakietu	220
Znajdowanie pakietów i ścieżka CLASSPATH	220
Prosty przykład pakietu	221
Ochrona dostępu	222
Przykład dostępu	223
Import pakietów	226
Interfejsy	228
Definiowanie interfejsu	228
Implementacja interfejsu	229
Interfejsy zagnieżdżone	232
Stosowanie interfejsów	233
Zmienne w interfejsach	236
Interfejsy można rozszerzać	238

Rozdział 10. Obsługa wyjątków	239
Podstawy obsługi wyjątków	239
Typy wyjątków	240
Nieprzechwycone wyjątki	241
Stosowanie konstrukcji try i catch	242
Wyświetlenie opisu wyjątku	243
Wiele klauzul catch	244
Zagnieżdżone konstrukcje try	245
Instrukcja throw	247
Klauzula throws	249
Słowo kluczowe finally	250
Wyjątki wbudowane w język Java	251
Tworzenie własnej podklasy wyjątków	252
Łącuch wyjątków	255
Trzy nowe cechy wyjątków w wersji JDK 7	256
Wykorzystanie wyjątków	258
Rozdział 11. Programowanie wielowątkowe	259
Model wątków języka Java	260
Priorytety wątków	261
Synchronizacja	262
Przekazywanie komunikatów	262
Klasa Thread i interfejs Runnable	262
Wątek główny	263
Tworzenie wątku	265
Implementacja interfejsu Runnable	265
Rozszerzanie klasy Thread	267
Wybór odpowiedniego podejścia	268
Tworzenie wielu wątków	269
Stosowanie metod isAlive() i join()	270
Priorytety wątków	272
Synchronizacja	273
Synchronizacja metod	274
Konstrukcja synchronized	276
Komunikacja międzywątkowa	277
Zakłeszczenie	282
Zawieszanie, wznawianie i zatrzymywanie wątków	284
Zawieszanie, wznawianie i zatrzymywanie wątków do wersji Java 1.1	284
Nowoczesny sposób zawieszania, wznawiania i zatrzymywania wątków	286
Uzyskiwanie stanu wątku	289
Korzystanie z wielowątkowości	290
Rozdział 12. Wyliczenia, automatyczne opakowywanie typów prostych i adnotacje (metadane)	291
Typy wyliczeniowe	291
Podstawy wyliczeń	292
Metody values() i valueOf()	294
Wyliczenia Javy jako typy klasowe	295
Wyliczenia dziedziczą po klasie Enum	297
Inny przykład wyliczenia	299
Opakowania typów	300
Klasa Character	301
Klasa Boolean	301
Opakowania typów numerycznych	301

Automatyczne opakowywanie typów prostych	303
Automatyczne opakowywanie i metody	304
Automatyczne opakowywanie i rozpakowywanie w wyrażeniach	304
Automatyczne opakowywanie typów znakowych i logicznych	306
Automatyczne opakowywanie pomaga zapobiegać błędom	307
Słowo ostrzeżenia	308
Adnotacje (metadane)	308
Podstawy tworzenia adnotacji	308
Określanie strategii zachowywania adnotacji	309
Odczytywanie adnotacji w trakcie działania programu za pomocą refleksji	310
Interfejs AnnotatedElement	315
Wartości domyślne	315
Adnotacje znacznikowe	317
Adnotacje jednoelementowe	318
Wbudowane adnotacje	319
Ograniczenia	321
Rozdział 13. Wejście-wyjście, aplety i inne tematy	323
Podstawowa obsługa wejścia i wyjścia	323
Strumienie	324
Strumienie znakowe i bajtowe	324
Predefiniowane strumienie	326
Odczyt danych z konsoli	327
Odczyt znaków	327
Odczyt łańcuchów	328
Wyświetlanie informacji na konsoli	330
Klasa PrintWriter	331
Odczyt i zapis plików	332
Automatyczne zamykanie pliku	338
Podstawy apletów	341
Modyfikatory transient i volatile	344
Operator instanceof	345
Modyfikator strictfp	347
Metody napisane w kodzie rdzennym	347
Problemy z metodami rdzennymi	351
Stosowanie asercji	351
Opcje włączania i wyłączania asercji	354
Import statyczny	354
Wywoływanie przeciążonych konstruktorów za pomocą this()	357
Rozdział 14. Typy sparametryzowane	361
Czym są typy sparametryzowane?	362
Prosty przykład zastosowania typów sparametryzowanych	362
Typy sparametryzowane działają tylko dla obiektów	366
Typy sparametryzowane różnią się, jeśli mają inny argument typu	366
W jaki sposób typy sparametryzowane zwiększają bezpieczeństwo?	366
Klasa sparametryzowana z dwoma parametrami typu	369
Ogólna postać klasy sparametryzowanej	370
Typy ograniczone	370
Zastosowanie argumentów wieloznacznych	373
Ograniczony argument wieloznaczny	375
Tworzenie metody sparametryzowanej	380
Konstruktory sparametryzowane	382
Interfejsy sparametryzowane	383
Typy surowe i starszy kod	385

Hierarchia klas sparametryzowanych	388
Zastosowanie sparametryzowanej klasy bazowej	388
Podklasa sparametryzowana	390
Porównywanie typów w hierarchii klas sparametryzowanych w czasie wykonywania	391
Rzutowanie	393
Przykrywanie metod w klasach sparametryzowanych	394
Wnioskowanie typów a typy sparametryzowane	395
Znoszenie	397
Metody mostu	398
Błędy niejednoznaczności	400
Pewne ograniczenia typów sparametryzowanych	401
Nie można tworzyć egzemplarza parametru typu	401
Ograniczenia dla składowych statycznych	402
Ograniczenia tablic typów sparametryzowanych	402
Ograniczenia wyjątków typów sparametryzowanych	404

Część II Biblioteka języka Java 405

Rozdział 15. Obsługa łańcuchów 407

Konstruktory klasy String	408
Długość łańcucha	410
Specjalne operacje na łańcuchach	410
Literały tekstowe	410
Konkatenacja łańcuchów	411
Konkatenacja łańcuchów z innymi typami danych	411
Konwersja łańcuchów i metoda toString()	412
Wyodrębnianie znaków	413
Metoda charAt()	413
Metoda getChars()	414
Metoda getBytes()	414
Metoda toCharArray()	415
Porównywanie łańcuchów	415
Metody equals() i equalsIgnoreCase()	415
Metoda regionMatches()	416
Metody startsWith() i endsWith()	416
Metoda equals() kontra operator ==	417
Metoda compareTo()	417
Przeszukiwanie łańcuchów	419
Modyfikowanie łańcucha	420
Metoda substring()	421
Metoda concat()	422
Metoda replace()	422
Metoda trim()	422
Konwersja danych za pomocą metody valueOf()	423
Zmiana wielkości liter w łańcuchu	424
Dodatkowe metody klasy String	425
Klasa StringBuffer	425
Konstruktory klasy StringBuffer	425
Metody length() i capacity()	427
Metoda ensureCapacity()	427
Metoda setLength()	427
Metody charAt() i setCharAt()	428
Metoda getChars()	428
Metoda append()	429

Metoda insert()	429
Metoda reverse()	430
Metody delete() i deleteCharAt()	430
Metoda replace()	431
Metoda substring()	431
Dodatkowe metody klasy StringBuffer	432
Klasa StringBuilder	433
Rozdział 16. Pakiet java.lang	435
Opakowania typów prostych	436
Klasa Number	436
Klasy Double i Float	436
Klasy Byte, Short, Integer i Long	440
Klasa Character	448
Dodatki wprowadzone w celu obsługi punktów kodowych Unicode	450
Klasa Boolean	451
Klasa Void	451
Klasa Process	452
Klasa Runtime	454
Zarządzanie pamięcią	454
Wykonywanie innych programów	456
Klasa ProcessBuilder	457
Klasa System	459
Wykorzystanie metody currentTimeMillis() do obliczania czasu wykonywania programu	461
Użycie metody arraycopy()	462
Właściwości środowiska	462
Klasa Object	463
Wykorzystanie metody clone() i interfejsu Cloneable	464
Klasa Class	466
Klasa ClassLoader	468
Klasa Math	469
Funkcje trygonometryczne	469
Funkcje wykładnicze	469
Funkcje zaokrągleń	470
Różnorodne metody klasy Math	470
Klasa StrictMath	471
Klasa Compiler	472
Klasy Thread i ThreadGroup oraz interfejs Runnable	472
Interfejs Runnable	473
Klasa Thread	473
Klasa ThreadGroup	473
Klasy ThreadLocal i InheritableThreadLocal	479
Klasa Package	479
Klasa RuntimePermission	480
Klasa Throwable	480
Klasa SecurityManager	481
Klasa StackTraceElement	481
Klasa Enum	482
Klasa ClassValue	482
Interfejs CharSequence	483
Interfejs Comparable	483
Interfejs Appendable	483
Interfejs Iterable	484
Interfejs Readable	484
Interfejs AutoCloseable	484

Interfejs Thread.UncaughtExceptionHandler	485
Podpakiety pakietu java.lang	485
Podpakiet java.lang.annotation	486
Podpakiet java.lang.instrument	486
Podpakiet java.lang.invoke	486
Podpakiet java.lang.management	486
Podpakiet java.lang.ref	486
Podpakiet java.lang.reflect	486
Rozdział 17. Pakiet java.util, część 1. — kolekcje	487
Wprowadzenie do kolekcji	488
Zmiany w kolekcjach wprowadzone w JDK 5	489
Typy sparametryzowane w znaczący sposób zmieniają kolekcje	489
Automatyczne opakowywanie ułatwia korzystanie z typów prostych	490
Pętla for typu for-each	490
Interfejsy kolekcji	490
Interfejs Collection	491
Interfejs List	493
Interfejs Set	494
Interfejs SortedSet	495
Interfejs NavigableSet	495
Interfejs Queue	496
Interfejs Deque	496
Klasy kolekcji	499
Klasa ArrayList	500
Klasa LinkedList	504
Klasa HashSet	505
Klasa LinkedHashSet	506
Klasa TreeSet	507
Klasa PriorityQueue	508
Klasa ArrayDeque	509
Klasa EnumSet	510
Dostęp do kolekcji za pomocą iteratora	510
Korzystanie z iteratora Iterator	512
Pętla typu for-each jako alternatywa dla iteratora	514
Przechowywanie w kolekcjach własnych klas	515
Interfejs RandomAccess	516
Korzystanie z map	516
Interfejsy map	517
Klasy map	519
Komparatory	526
Wykorzystanie komparatora	527
Algorytmy kolekcji	529
Klasa Arrays	534
Dlaczego kolekcje są sparametryzowane?	539
Starsze klasy i interfejsy	541
Interfejs Enumeration	542
Klasa Vector	542
Klasa Stack	546
Klasa Dictionary	548
Klasa Hashtable	549
Klasa Properties	552
Wykorzystanie metod store() i load()	555
Ostatnie uwagi na temat kolekcji	557

Rozdział 18. Pakiet java.util, część 2. — pozostałe klasy użytkowe	559
Klasa StringTokenizer	559
Klasa BitSet	561
Klasa Date	564
Klasa Calendar	566
Klasa GregorianCalendar	569
Klasa TimeZone	570
Klasa SimpleTimeZone	571
Klasa Locale	572
Klasa Random	574
Klasa Observable	576
Interfejs Observer	577
Przykład użycia interfejsu Observer	577
Klasy Timer i TimerTask	580
Klasa Currency	582
Klasa Formatter	583
Konstruktory klasy Formatter	584
Metody klasy Formatter	585
Podstawy formatowania	585
Formatowanie łańcuchów i znaków	588
Formatowanie liczb	588
Formatowanie daty i godziny	589
Specyfikatory %n i %%	591
Określanie minimalnej szerokości pola	591
Określanie precyzji	593
Używanie znaczników (flag) formatów	594
Wyrównywanie danych wyjściowych	594
Znaczniki spacji, plusa, zera i nawiasów	595
Znacznik przecinka	596
Znacznik #	596
Opcja wielkich liter	597
Stosowanie indeksu argumentu	597
Zamykanie obiektu klasy Formatter	599
Metoda printf() w Javie	599
Klasa Scanner	600
Konstruktory klasy Scanner	600
Podstawy skanowania	600
Kilka przykładów użycia klasy Scanner	604
Ustawianie separatorów	609
Pozostałe elementy klasy Scanner	610
Klasy ResourceBundle, ListResourceBundle i PropertyResourceBundle	611
Dodatkowe klasy i interfejsy użytkowe	616
Podpakiety pakietu java.util	616
java.util.concurrent, java.util.concurrent.atomic oraz java.util.concurrent.locks	617
java.util.jar	617
java.util.logging	617
java.util.prefs	617
java.util.regex	617
java.util.spi	617
java.util.zip	617

Rozdział 19. Operacje wejścia-wyjścia: analiza pakietu java.io	619
Klasy i interfejsy obsługujące operacje wejścia-wyjścia	620
Klasa File	621
Katalogi	624
Stosowanie interfejsu FilenameFilter	625
Alternatywna metoda listFiles()	626
Tworzenie katalogów	626
Interfejsy AutoCloseable, Closeable i Flushable	627
Wyjątki operacji wejścia-wyjścia	628
Dwa sposoby zamykania strumieni	628
Klasy strumieni	630
Strumienie bajtów	630
Klasa InputStream	630
Klasa OutputStream	631
Klasa FileInputStream	632
Klasa FileOutputStream	634
Klasa ByteArrayInputStream	636
Klasa ByteArrayOutputStream	638
Filtrowane strumienie bajtów	639
Buforowane strumienie bajtów	639
Klasa SequenceInputStream	644
Klasa PrintStream	645
Klasy DataOutputStream i DataInputStream	648
Klasa RandomAccessFile	650
Strumienie znaków	651
Klasa Reader	651
Klasa Writer	651
Klasa FileReader	651
Klasa FileWriter	653
Klasa CharArrayReader	654
Klasa CharArrayWriter	655
Klasa BufferedReader	657
Klasa BufferedWriter	658
Klasa PushbackReader	659
Klasa PrintWriter	660
Klasa Console	661
Serializacja	663
Interfejs Serializable	664
Interfejs Externalizable	664
Interfejs ObjectOutput	664
Klasa ObjectOutputStream	664
Interfejs ObjectInput	665
Klasa ObjectInputStream	665
Przykład serializacji	666
Korzyści wynikające ze stosowania strumieni	669
Rozdział 20. System NIO	671
Klasy systemu NIO	671
Podstawy systemu NIO	672
Bufory	672
Kanały	673
Zestawy znaków i selektory	675

Udoskonalenia dodane do systemu NIO w wydaniu JDK 7	676
Interfejs Path	676
Klasa Files	678
Klasa Paths	679
Interfejsy atrybutów plików	680
Klasy FileSystem, FileSystems i FileStore	682
Stosowanie systemu NIO	683
Stosowanie systemu NIO dla operacji wejścia-wyjścia na kanałach	684
Stosowanie systemu NIO dla operacji wejścia-wyjścia na strumieniach	693
Stosowanie systemu NIO dla operacji na ścieżkach i systemie plików	695
Przykłady stosowania kanałów w wersjach sprzed JDK 7	703
Odczytywanie plików (wersje sprzed JDK 7)	704
Zapisywanie plików (wersje sprzed JDK 7)	707
Rozdział 21. Obsługa sieci	711
Podstawy działania sieci	711
Klasy i interfejsy obsługujące komunikację sieciową	713
Klasa InetAddress	714
Metody fabryczne	714
Metody klasy	715
Klasy InetAddress oraz Inet6Address	716
Gniazda klientów TCP/IP	716
URL	720
Klasa URLConnection	722
Klasa HttpURLConnection	724
Klasa URI	727
Pliki cookie	727
Gniazda serwerów TCP/IP	727
Datagramy	728
Klasa DatagramSocket	728
Klasa DatagramPacket	729
Przykład użycia datagramów	730
Rozdział 22. Klasa Applet	733
Dwa rodzaje apletów	733
Podstawy apletów	734
Klasa Applet	735
Architektura apletu	735
Szkielec apletu	738
Inicjalizacja i przerywanie działania apletu	739
Przykrycie metody update()	740
Proste metody wyświetlania składników apletów	740
Żądanie ponownego wyświetlenia	742
Prosty aplet z paskiem reklamowym	744
Wykorzystywanie paska stanu	746
Znacznik APPLET języka HTML	747
Przekazywanie parametrów do apletów	748
Udoskonalenie apletu z paskiem reklamowym	750
Metody getDocumentBase() i getCodeBase()	751
Interfejs AppletContext i metoda showDocument()	752
Interfejs AudioClip	754
Interfejs AppletStub	754
Wyświetlanie danych wyjściowych na konsoli	754

Rozdział 23. Obsługa zdarzeń	755
Dwa mechanizmy obsługi zdarzeń	756
Model obsługi zdarzeń oparty na ich delegowaniu	756
Zdarzenia	757
Źródła zdarzeń	757
Obiekty nasłuchujące zdarzeń	758
Klasy zdarzeń	758
Klasa ActionEvent	759
Klasa AdjustmentEvent	761
Klasa ComponentEvent	762
Klasa ContainerEvent	762
Klasa FocusEvent	763
Klasa InputEvent	764
Klasa ItemEvent	765
Klasa KeyEvent	765
Klasa MouseEvent	766
Klasa MouseWheelEvent	768
Klasa TextEvent	769
Klasa WindowEvent	770
Źródła zdarzeń	771
Interfejsy nasłuchujące zdarzeń	772
Interfejs ActionListener	772
Interfejs AdjustmentListener	773
Interfejs ComponentListener	773
Interfejs ContainerListener	773
Interfejs FocusListener	773
Interfejs ItemListener	773
Interfejs KeyListener	774
Interfejs MouseListener	774
Interfejs MouseMotionListener	774
Interfejs MouseWheelListener	774
Interfejs TextListener	775
Interfejs WindowFocusListener	775
Interfejs WindowListener	775
Stosowanie modelu delegowania zdarzeń	775
Obsługa zdarzeń generowanych przez mysz	776
Obsługa zdarzeń generowanych przez klawiaturę	779
Klasy adapterów	782
Klasy wewnętrzne	784
Anonimowa klasa wewnętrzna	785
 Rozdział 24. Wprowadzenie do AWT: praca z oknami, grafiką i tekstem	 787
Klasy AWT	788
Podstawy okien	790
Klasa Component	790
Klasa Container	791
Klasa Panel	791
Klasa Window	792
Klasa Frame	792
Klasa Canvas	792
Praca z oknami typu Frame	792
Ustawianie wymiarów okna	793
Ukrywanie i wyświetlanie okna	793
Ustawianie tytułu okna	793
Zamykanie okna typu Frame	793

Tworzenie okna typu Frame z poziomu apletu	794
Obsługa zdarzeń w oknie typu Frame	796
Tworzenie programu wykorzystującego okna	800
Wyświetlanie informacji w oknie	801
Praca z grafiką	802
Rysowanie odcinków	802
Rysowanie prostokątów	803
Rysowanie elips, kół i okręgów	804
Rysowanie łuków	805
Rysowanie wielokątów	806
Dostosowywanie rozmiarów obiektów graficznych	807
Praca z klasą Color	808
Metody klasy Color	809
Ustawianie bieżącego koloru kontekstu graficznego	810
Aplet demonstrujący zastosowanie klasy Color	810
Ustawianie trybu rysowania	811
Praca z czcionkami	812
Określanie dostępnych czcionek	813
Tworzenie i wybieranie czcionek	815
Uzyskiwanie informacji o czcionkach	817
Zarządzanie tekstowymi danymi wyjściowymi z wykorzystaniem klasy FontMetrics	818
Wyświetlanie tekstu w wielu wierszach	819
Wyśrodkowanie tekstu	821
Wyrównywanie wielowierszowych danych tekstowych	822

Rozdział 25. Stosowanie kontroltek AWT, menedżerów układu graficznego oraz menu 827

Podstawy kontroltek	828
Dodawanie i usuwanie kontroltek	828
Odpowiadanie na zdarzenia kontroltek	828
Wyjątek HeadlessException	829
Etykiety	829
Stosowanie przycisków	830
Obsługa zdarzeń przycisków	831
Stosowanie pól wyboru	834
Obsługa zdarzeń pól wyboru	834
Klasa CheckboxGroup	836
Kontrolki list rozwijanych	838
Obsługa zdarzeń list rozwijanych	839
Stosowanie list	840
Obsługa zdarzeń generowanych przez listy	841
Zarządzanie pasekami przewijania	843
Obsługa zdarzeń generowanych przez paski przewijania	844
Stosowanie kontroltek typu TextField	846
Obsługa zdarzeń generowanych przez kontrolkę TextField	848
Stosowanie kontroltek typu TextArea	849
Wprowadzenie do menedżerów układu graficznego komponentów	850
FlowLayout	852
BorderLayout	854
Stosowanie obramowań	855
GridLayout	857
Klasa CardLayout	858
Klasa GridBagLayout	861
Menu i paski menu	866
Okna dialogowe	872
FileDialog	877

Obsługa zdarzeń przez rozszerzanie dostępnych komponentów AWT	878
Rozszerzanie kontrolki Button	879
Rozszerzanie kontrolki Checkbox	880
Rozszerzanie komponentu grupy pól wyboru	881
Rozszerzanie kontrolki Choice	882
Rozszerzanie kontrolki List	883
Rozszerzanie kontrolki Scrollbar	884
Przykrywanie metody paint()	885
Rozdział 26. Obrazy	887
Formaty plików	888
Podstawy przetwarzania obrazów: tworzenie, wczytywanie i wyświetlanie	888
Tworzenie obiektu obrazu	888
Ładowanie obrazu	889
Wyświetlanie obrazu	889
Interfejs ImageObserver	891
Podwójne buforowanie	892
Klasa MediaTracker	895
Interfejs ImageProducer	898
Klasa MemoryImageSource	898
Interfejs ImageConsumer	900
Klasa PixelGrabber	900
Klasa ImageFilter	902
Klasa CropImageFilter	903
Klasa RGBImageFilter	905
Animacja poklatkowa	916
Dodatkowe klasy obsługujące obrazy	919
Rozdział 27. Narzędzia współbieżności	921
Pakiety interfejsu Concurrent API	922
Pakiet java.util.concurrent	922
Pakiet java.util.concurrent.atomic	923
Pakiet java.util.concurrent.locks	924
Korzystanie z obiektów służących do synchronizacji	924
Klasa Semaphore	924
Klasa CountDownLatch	930
CyclicBarrier	931
Klasa Exchanger	934
Klasa Phaser	936
Korzystanie z egzekutorów	944
Przykład prostego egzekutora	945
Korzystanie z interfejsów Callable i Future	946
Obiekty typu TimeUnit	949
Kolekcje współbieżne	950
Blokady	951
Operacje atomowe	954
Programowanie równoległe przy użyciu frameworku Fork/Join	955
Najważniejsze klasy frameworku Fork/Join	956
Strategia dziel i zwyciężaj	959
Prosty przykład użycia frameworku Fork/Join	960
Znaczenie poziomu równoległości	962
Przykład użycia klasy RecursiveTask<V>	966
Asynchroniczne wykonywanie zadań	968
Anulowanie zadania	968
Określanie statusu wykonania zadania	969

Ponowne uruchamianie zadania	969
Pozostałe zagadnienia	970
Wskazówki dotyczące stosowania frameworku Fork/Join	971
Pakiet Concurrency Utilities a tradycyjne metody języka Java	972

Rozdział 28. Wyrażenia regularne i inne pakiety 973

Pakiety głównego API języka Java	973
Przetwarzanie wyrażeń regularnych	973
Klasa Pattern	975
Klasa Matcher	976
Składnia wyrażeń regularnych	977
Przykład dopasowywania do wzorca	977
Dwie opcje dopasowywania do wzorca	983
Przegląd wyrażeń regularnych	983
Refleksje	983
Zdalne wywoływanie metod (RMI)	987
Prosta aplikacja typu klient-serwer wykorzystująca RMI	987
Formatowanie tekstu	991
Klasa DateFormat	991
Klasa SimpleDateFormat	993

Część III Pisanie oprogramowania w języku Java 997

Rozdział 29. Java Beans 999

Czym jest komponent typu Java Bean?	999
Zalety komponentów Java Beans	1000
Introspekcja	1000
Wzorce właściwości	1000
Wzorce projektowe dla zdarzeń	1002
Metody i wzorce projektowe	1002
Korzystanie z interfejsu BeanInfo	1002
Właściwości ograniczone	1003
Trwałość	1003
Interfejs Customizer	1004
Interfejs Java Beans API	1004
Klasa Introspector	1004
KlasaPropertyDescriptor	1006
Klasa EventSetDescriptor	1006
Klasa MethodDescriptor	1007
Przykład komponentu Java Bean	1007

Rozdział 30. Wprowadzenie do pakietu Swing 1011

Geneza powstania biblioteki Swing	1012
Bibliotekę Swing zbudowano na bazie zestawu narzędzi AWT	1012
Podstawowe cechy biblioteki Swing	1013
Komponenty biblioteki Swing są lekkie	1013
Biblioteka Swing obsługuje dołączany wygląd i sposób obsługi	1013
Podobieństwo do architektury MVC	1014
Komponenty i kontenery	1015
Komponenty	1015
Kontenery	1016
Panele kontenerów najwyższego poziomu	1016
Pakiety biblioteki Swing	1017
Prosta aplikacja na bazie biblioteki Swing	1018

Obsługa zdarzeń	1022
Tworzenie apletu na bazie biblioteki Swing	1026
Rysowanie w bibliotece Swing	1028
Podstawy rysowania	1028
Wyznaczanie obszaru rysowania	1029
Przykład rysowania	1030
Rozdział 31. Przewodnik po pakiecie Swing	1035
Klasy JLabel i ImageIcon	1036
Klasa JTextField	1038
Przyciski biblioteki Swing	1039
Klasa JButton	1040
Klasa JToggleButton	1042
Pola wyboru	1045
Przyciski opcji	1046
Klasa JTabbedPane	1049
Klasa JScrollPane	1051
Klasa JList	1053
Klasa JComboBox	1056
Drzewa	1059
Klasa JTable	1062
Dalsze poznawanie biblioteki Swing	1065
Rozdział 32. Serwlety	1067
Podstawy	1067
Cykl życia serwletu	1068
Alternatywne sposoby tworzenia serwletów	1069
Korzystanie ze środowiska Tomcat	1069
Przykład prostego serwletu	1071
Tworzenie i kompilacja kodu źródłowego serwletu	1071
Uruchamianie serwera Tomcat	1072
Uruchamianie przeglądarki i generowanie żądania	1072
Interfejs Servlet API	1073
Pakiet javax.servlet	1073
Interfejs Servlet	1074
Interfejs ServletConfig	1074
Interfejs ServletContext	1074
Interfejs ServletRequest	1075
Interfejs ServletResponse	1075
Klasa GenericServlet	1075
Klasa ServletInputStream	1075
Klasa ServletOutputStream	1077
Klasy wyjątków związanych z serwletami	1077
Odczytywanie parametrów serwletu	1077
Pakiet javax.servlet.http	1079
Interfejs HttpServletRequest	1079
Interfejs HttpServletResponse	1079
Interfejs HttpSession	1080
Interfejs HttpSessionBindingListener	1081
Klasa Cookie	1081
Klasa HttpServlet	1083
Klasa HttpSessionEvent	1083
Klasa HttpSessionBindingEvent	1084

Obsługa żądań i odpowiedzi HTTP	1085
Obsługa żądań GET protokołu HTTP	1085
Obsługa żądań POST protokołu HTTP	1086
Korzystanie ze znaczników kontekstu użytkownika	1088
Śledzenie sesji	1090

Część IV Zastosowanie Javy w praktyce 1093

Rozdział 33. Aplety i serwlety finansowe 1095

Obliczanie raty pożyczki	1096
Pola apletu	1099
Metoda init()	1100
Metoda makeGUI()	1100
Metoda actionPerformed()	1102
Metoda compute()	1103
Obliczanie przyszłej wartości inwestycji	1104
Obliczanie wkładu początkowego wymaganego do uzyskania przyszłej wartości inwestycji	1108
Obliczanie początkowej wartości inwestycji niezbędnej do uzyskania odpowiedniej emerytury	1112
Obliczanie maksymalnej emerytury dla danej inwestycji	1116
Obliczenie pozostałej kwoty do spłaty kredytu	1120
Tworzenie serwletów finansowych	1123
Konwersja apletu RegPay do serwletu	1124
Serwlet RegPayS	1124
Możliwe rozszerzenia	1127

Rozdział 34. Tworzenie menedżera pobierania plików w Javie 1129

Sposoby pobierania plików z internetu	1130
Omówienie programu	1130
Klasa Download	1131
Zmienne pobierania	1135
Konstruktor klasy	1135
Metoda download()	1135
Metoda run()	1135
Metoda stateChanged()	1138
Metody akcesorów i działań	1139
Klasa ProgressRenderer	1139
Klasa DownloadsTableModel	1140
Metoda addDownload()	1142
Metoda clearDownload()	1142
Metoda getColumnClass()	1143
Metoda getValueAt()	1143
Metoda update()	1143
Klasa DownloadManager	1144
Zmienne klasy DownloadManager	1149
Konstruktor klasy	1149
Metoda verifyUrl()	1150
Metoda tableSelectionChanged()	1150
Metoda updateButtons()	1151
Obsługa zdarzeń akcji	1152
Kompilacja i uruchamianie programu	1152
Rozszerzanie możliwości programu	1152

Dodatek A	Komentarze dokumentujące	1155
	Znaczniki narzędzia javadoc	1155
	Znacznik @author	1156
	Znacznik {@code}	1156
	Znacznik @deprecated	1157
	Znacznik {@docRoot}	1157
	Znacznik @exception	1157
	Znacznik {@inheritDoc}	1157
	Znacznik {@link}	1157
	Znacznik {@linkplain}	1157
	Znacznik {@literal}	1158
	Znacznik @param	1158
	Znacznik @return	1158
	Znacznik @see	1158
	Znacznik @serial	1159
	Znacznik @serialData	1159
	Znacznik @serialField	1159
	Znacznik @since	1159
	Znacznik @throws	1159
	Znacznik {@value}	1159
	Znacznik @version	1160
	Ogólna postać komentarzy dokumentacyjnych	1160
	Wynik działania narzędzia javadoc	1160
	Przykład korzystający z komentarzy dokumentacyjnych	1161
	Skorowidz	1163

Rozdział 9.

Pakiety i interfejsy

Niniejszy rozdział omawia dwie najbardziej innowacyjne cechy języka Java: pakiety i interfejsy. **Pakiety** to pojemniki na klasy używane do odpowiedniego podziału przestrzeni nazw klas. Programista może utworzyć w swoim pakiecie na przykład klasę `List` bez ryzyka konfliktu nazw z klasą `List` należącą do innego pakietu. Pakiety zapewniają strukturę podobną do hierarchicznej i są jawnie importowane do nowych definicji klas.

W poprzednich rozdziałach przedstawiłem, w jaki sposób metody definiują interfejs dla danych klasy. Słowo kluczowe `interface` umożliwia całkowite oddzielenie interfejsu od jego implementacji. Interfejs w Javie określa zbiór metod, które będą implementowane przez jedną lub więcej klas. Sam interfejs nie definiuje żadnej implementacji. Interfejsy przypominają klasy abstrakcyjne, ale mają jedną bardzo ważną zaletę: klasa może implementować dowolną liczbę interfejsów. Przypomnę, iż klasa może dziedziczyć tylko po jednej klasie (abstrakcyjnej lub tradycyjnej).

Pakiety

W dotychczasowych przykładach nazwa każdej z przykładowych klas pochodziła z jednej, tej samej przestrzeni nazw. Oznacza to, że każda klasa musiała mieć unikatową nazwę, aby nie dochodziło do konfliktów nazw. Gdyby nie przestrzenie nazw, z pewnością w większych projektach szybko wyczerpałaby się lista opisowych nazw dla klas. Co więcej, trzeba zapewnić, by różni programiści nie tworzyli klas o takich samych nazwach. (Wyobraź sobie małą grupę programistów walczących o to, kto będzie mógł nadać swojej klasie nazwę `FooBar`, lub sytuację, gdy na wszystkich grupach dyskusyjnych toczy się spór o to, kto pierwszy nadał klasie nazwę `Espresso`). Projektanci Javy wymyślili bardzo elastyczny mechanizm dzielenia przestrzeni nazw na mniejsze kawałki. Mechanizm ten nosi nazwę pakietu. Pakiet to nie tylko przestrzeń nazw, ale też sposób kontroli widoczności i dostępu. Można zdefiniować klasy dostępne jedynie dla innych klas z tego samego pakietu. Co więcej, można zdefiniować metody dostępne dla innych klas tego samego pakietu, ale nieosiągalne dla pozostałych klas. W ten sposób klasy z pakietu mają bardzo dobrą wiedzę o sobie, ale nie udostępniają tych informacji całemu światu.

Definiowanie pakietu

Tworzenie pakietu nie jest trudne. Wystarczy zastosować polecenie `package` jako pierwsze wyrażenie w pliku źródłowym Javy. Wszystkie klasy zdefiniowane w tak rozpoczętym pliku zostaną przypisane do wskazanego pakietu. Pakiet określa przestrzeń nazw dla klas. Pominięcie instrukcji `package` powoduje przypisanie klasy do domyślnej przestrzeni nazw (przestrzeni bez nazwy). (Dzięki temu aż do tego rozdziału nie musieliśmy się zajmować pakietami). Pakiet domyślny nadaje się w zasadzie tylko i wyłącznie do pisania krótkich programów, więc nie jest stosowany w rzeczywistych aplikacjach. Na ogół stosuje się własne przestrzenie nazw, czyli definiuje się własne pakiety.

Oto ogólna postać instrukcji `package`.

```
package pakiet;
```

Element *pakiet* to nazwa pakietu. Poniższa instrukcja tworzy pakiet o nazwie `MyPackage`.

```
package MyPackage;
```

Java używa katalogów systemu plików do przechowywania pakietów. Oznacza to, że każdy plik `.class` utworzony dla pakietu `MyPackage` musi zostać umieszczony w katalogu o nazwie *MyPackage*. Pamiętaj, że wielkość liter ma znaczenie i nazwa katalogu musi być dokładnie taka sama jak nazwa pakietu.

Ta sama instrukcja `package` może pojawić się w więcej niż jednym pliku źródłowym. Po prostu określa ona, do jakiego pakietu należą klasy zdefiniowane w danym pliku. Inne klasy zdefiniowane w innym pliku źródłowym również mogą należeć do tego pakietu. W rzeczywistych aplikacjach pakiet składa się z wielu plików.

Można tworzyć hierarchię pakietów. W tym celu należy oddzielić nazwy poszczególnych pakietów znakiem kropki (`.`). Ogólna postać wielopoziomowego pakietu jest następująca.

```
package pakiet1[.pakiet2[.pakiet3]];
```

Hierarchia pakietów musi być odtworzona w systemie plików środowiska, w którym działa Java. Na przykład pakiet zadeklarowany za pomocą wyrażenia:

```
package java.awt.image;
```

w systemie Windows musi znaleźć się w katalogu `java\awt\images`. Warto ostrożnie dobrać nazwę pakietu. Nie można zmienić nazwy pakietu bez jednoczesnej zmiany nazwy katalogu.

Znajdowanie pakietów i ścieżka CLASSPATH

Jak już wspomniano, pakiety są reprezentowane w systemie operacyjnym przez katalogi. Powstaje więc pytanie: jak to się dzieje, że system wykonawczy Javy „wie”, gdzie ma szukać utworzonych przez nas pakietów? Odpowiedź musi uwzględniać trzy aspekty. Po pierwsze, domyślnie system wykonawczy używa aktualnego katalogu jako początku ścieżki wyszukiwania. Jeśli pakiet znajduje się w aktualnym katalogu lub jego podkatalogu, zostanie odnaleziony.

Po drugie, zmienna środowiskowa `CLASSPATH` określa inne lokalizacje, w których trzeba szukać pakietów. Po trzecie, programista może wskazać ścieżkę do swoich klas za pomocą opcji `-classpath` poleceń `java` i `javac`.

Rozważmy następującą specyfikację pakietu.

```
package MyPack;
```

Program odnajdzie pakiet `MyPack`, jeśli spełniony został jeden z trzech warunków. Program został uruchomiony z poziomu katalogu znajdującego się bezpośrednio nad katalogiem `MyPack` w hierarchii katalogów, zmienna środowiskowa `CLASSPATH` zawiera ścieżkę do katalogu `MyPack` lub opcja `-classpath` polecenia `java` wskazuje ścieżkę do katalogu `MyPack`.

W przypadku stosowania drugiego z wymienionych rozwiązań ścieżka do klas nie może obejmować samego katalogu `MyPack`. Powinna raczej wskazywać ścieżkę do katalogu nadrzędnego względem katalogu `MyPack`. Jeśli na przykład w środowisku Windows ścieżka do katalogu `MyPack` jest następująca:

```
C:\MyPrograms\Java\MyPack
```

prawidłowa ścieżka do klas powinna mieć następującą postać:

```
C:\MyPrograms\Java
```

Najprostszym sposobem sprawdzenia przykładów znajdujących się w niniejszym rozdziale jest utworzenie nowego katalogu o nazwie zgodnej z nazwą pakietu w aktualnym katalogu testowym i umieszczenie w nim wszystkich plików `.class`. Programy należy uruchamiać z katalogu testowego. W kolejnych przykładach zakładam takie właśnie podejście.

Prosty przykład pakietu

Pamiętając o wcześniejszej wskazówce, warto skompilować i uruchomić poniższy przykład.

```
// prosty pakiet
package MyPack;

class Balance {
    String name;
    double bal;

    Balance(String n, double b) {
        name = n;
        bal = b;
    }

    void show() {
        if(bal<0)
            System.out.print("--->> ");
        System.out.println(name + ": " + bal+ " zł");
    }
}

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

```
Balance current[] = new Balance[3];

current[0] = new Balance("K. J. Fielding", 123.23);
current[1] = new Balance("Willhelm Tell", 157.02);
current[2] = new Balance("Tom Jackson", -12.33);

for(int i=0; i<3; i++) current[i].show();
    }
}
```

Nadaj plikowi źródłowemu nazwę *AccountBalance.java* i umieść go w katalogu *MyPack*.

Następnie skompiluj kod. Umieść wynikowy plik *.class* w katalogu *MyPack*. Uruchom program, używając poniższego polecenia.

```
java MyPack.AccountBalance
```

Należy pamiętać, że powyższe polecenie powinno być wykonane z poziomu katalogu powyżej katalogu *MyPack*. (Alternatywnym rozwiązaniem jest użycie jednej z dwóch pozostałych technik opisanych w poprzednim punkcie i umożliwiających wskazanie ścieżki do katalogu *MyPack*).

Klasa *AccountBalance* stanowi część pakietu *MyPack*. Oznacza to, iż nie można jej wykonać bezpośrednio, czyli zastosować poniższego polecenia.

```
java AccountBalance
```

Nazwę klasy trzeba poprzedzić nazwą pakietu.

Ochrona dostępu

W poprzednich rozdziałach wspominałem o wielu aspektach mechanizmu sterowania dostępem i modyfikatorach dostępu. Na przykład teraz każdy wie, iż składowa prywatna klasy może być odczytywana tylko i wyłącznie przez inne składowe tej klasy. Pakiety dodają nowy wymiar do sterowania dostępem. Java zapewnia wiele poziomów ochrony, aby programista mógł szczegółowo określić widoczność zmiennych i metod w klasach, podklasach i pakietach.

Klasy i pakiety to rozwiązania pozwalające hermetyzować zmienne oraz metody, a także określać przestrzenie nazw. Pakiety działają jak pojemniki na klasy i inne pakiety. Klasy to pojemniki przechowujące dane i kod. Klasa stanowi najmniejszą jednostkę abstrakcji w języku Java. Dzięki wzajemnemu współdziałaniu klas i pakietów możliwe są cztery kategorie widoczności składowych klas:

- ♦ podklasy z tego samego pakietu,
- ♦ inne klasy z tego samego pakietu,
- ♦ podklasy z innych pakietów,
- ♦ inne klasy z innych pakietów.

Trzy modyfikatory dostępu (`private`, `public` i `protected`) pozwalają tworzyć wiele poziomów dostępu dla wymienionych kategorii. Tabela 9.1 podsumowuje wszystkie możliwe kombinacje.

Tabela 9.1. *Dostęp do składowych klasy*

	private	Brak modyfikatora	protected	public
Ta sama klasa	Tak	Tak	Tak	Tak
Ten sam pakiet, podklasa	Nie	Tak	Tak	Tak
Ten sam pakiet, inna klasa	Nie	Tak	Tak	Tak
Inny pakiet, podklasa	Nie	Nie	Tak	Tak
Inny pakiet, inna klasa	Nie	Nie	Nie	Tak

Choć mechanizm sterowania dostępem w Javie wydaje się złożony, bardzo łatwo go zrozumieć. Cokolwiek zadeklarowane z modyfikatorem `public` jest dostępne z dowolnego miejsca. Cokolwiek zadeklarowane z modyfikatorem `private` nie jest dostępne poza własną klasą. Gdy element nie ma jawnie określonego modyfikatora, jest widoczny w podklasach oraz w innych klasach tego samego pakietu. Jest to dostęp domyślny. Jeżeli element ma być widoczny poza pakietem, ale tylko dla klas dziedziczących po danej klasie, należy użyć modyfikatora `protected`.

Tabela 9.1 dotyczy tylko składowych klasy. Sama klasa może posiadać tylko dwa poziomy dostępu: domyślny i publiczny. Gdy klasa zostanie zadeklarowana z modyfikatorem `public`, jest dostępna dla dowolnego innego kodu. Zastosowanie domyślnego dostępu powoduje, iż klasa dostępna jest tylko dla innych klas z tego samego pakietu.

Przykład dostępu

Kolejny przykład ilustruje wykorzystanie wszystkich kombinacji modyfikatorów dostępu. Przykład składa się z pięciu klas i dwóch pakietów. Należy pamiętać, że klasy znajdujące się w pakietach muszą zostać umieszczone w katalogach nazwanych tak samo jak odpowiednie pakiety — w tym przypadku w katalogach `p1` i `p2`.

Plik źródłowy pierwszego pakietu definiuje trzy klasy: `Protection`, `Derived` i `SamePackage`. Pierwsza klasa definiuje cztery zmienne typu `int` o różnym dostępie. Zmienna `n` ma dostęp domyślny, `n_pri` dostęp prywatny, `n_pro` dostęp chroniony, a `n_pub` dostęp publiczny.

Pozostałe klasy z przykładu będą próbowały uzyskać dostęp do zmiennych egzemplarza klasy `Protection`. Wiersze, które spowodowałyby błąd kompilacji w związku z ograniczeniami dostępu, zostały opatrzone komentarzem jednowierszowym (znaki `//`). Przed każdym z tych wierszy dodatkowo umieszczono komentarz informujący o poziomie ochrony i ograniczeniach dostępu.

Druga klasa, `Derived`, jest podklasą klasy `Protection` z tego samego pakietu `p1`. Powoduje to, iż klasa `Derived` ma dostęp do wszystkich zmiennych klasy bazowej poza `n_pri`, zmienną prywatną. Trzecia klasa, `SamePackage`, nie jest podklasą klasy `Protection`, ale znajduje się w tym samym pakiecie, więc również ma dostęp do wszystkich zmiennych poza `n_pri`.

Kod z pliku *Protection.java*.

```
package p1;

public class Protection {
    int n = 1;
    private int n_pri = 2;
    protected int n_pro = 3;
    public int n_pub = 4;

    public Protection() {
        System.out.println("konstruktor klasy bazowej");
        System.out.println("n = " + n);
        System.out.println("n_pri = " + n_pri);
        System.out.println("n_pro = " + n_pro);
        System.out.println("n_pub = " + n_pub);
    }
}
```

Kod z pliku *Derived.java*.

```
package p1;

class Derived extends Protection {
    Derived() {
        System.out.println("konstruktor podklasy");
        System.out.println("n = " + n);

        // tylko ta sama klasa
        // System.out.println("n_pri = " + n_pri);

        System.out.println("n_pro = " + n_pro);
        System.out.println("n_pub = " + n_pub);
    }
}
```

Kod z pliku *SamePackage.java*.

```
package p1;

class SamePackage {
    SamePackage() {
        Protection p = new Protection();
        System.out.println("konstruktor z tego samego pakietu");
        System.out.println("n = " + p.n);

        // tylko ta sama klasa
        // System.out.println("n_pri = " + p.n_pri);

        System.out.println("n_pro = " + p.n_pro);
        System.out.println("n_pub = " + p.n_pub);
    }
}
```

Teraz zajmijmy się kodem źródłowym drugiego pakietu, p2. Dwie zdefiniowane w nim klasy pozwalają sprawdzić dwa pozostałe przypadki związane ze sterowaniem dostępem. Pierwsza klasa, *Protection2*, jest podklasą klasy p1.*Protection*. Ma ona dostęp do wszystkich zmien-

nych poza zmiennymi `n_pri` (ponieważ jest to zmienna prywatna) i `n` (gdyż stosuje domyślny poziom dostępu). Pamiętaj, że dostęp domyślny zapewnia jedynie dostęp z poziomu tej samej klasy lub pakietu. Klasy potomne należące do innych pakietów nie mają dostępu do takiej zmiennej. Klasa `OtherPackage` ma dostęp jedynie do jednej zmiennej, `n_pub`, zadeklarowanej z modyfikatorem `public`.

Kod z pliku *Protection2.java*.

```
package p2;

class Protection2 extends p1.Protection {
    Protection2() {
        System.out.println("konstruktor podklasy z innego pakietu");

        // tylko ta sama klasa lub pakiet
        // System.out.println("n = " + n);

        // tylko ta sama klasa
        // System.out.println("n_pri = " + n_pri);

        System.out.println("n_pro = " + n_pro);
        System.out.println("n_pub = " + n_pub);
    }
}
```

Kod z pliku *OtherPackage.java*.

```
package p2;

class OtherPackage {
    OtherPackage() {
        p1.Protection p = new p1.Protection();
        System.out.println("konstruktor z innego pakietu");

        // tylko ta sama klasa lub pakiet
        // System.out.println("n = " + p.n);

        // tylko ta sama klasa
        // System.out.println("n_pri = " + p.n_pri);

        // tylko ta sama klasa, pakiet lub podklasa
        // System.out.println("n_pro = " + p.n_pro);

        System.out.println("n_pub = " + p.n_pub);
    }
}
```

Poniżej znajdują się kody źródłowe dwóch testów służących do sprawdzenia działania obu pakietów. Oto kod testujący pakiet `p1`.

```
// Test pakietu p1.
package p1;

// Tworzenie egzemplarzy klas z p1.
public class Demo {
    public static void main(String args[]) {
        Protection obl = new Protection();
    }
}
```

```
        Derived ob2 = new Derived();
        SamePackage ob3 = new SamePackage();
    }
}
```

Kod testujący pakiet p2 jest następujący.

```
// Test pakietu p2.
package p2;

// Tworzenie egzemplarzy klas z p2.
public class Demo {
    public static void main(String args[]) {
        Protection2 ob1 = new Protection2();
        OtherPackage ob2 = new OtherPackage();
    }
}
```

Import pakietów

Istnienie pakietów pozwala łatwo segregować klasy i tworzyć różne grupy. Właśnie z tego powodu wszystkie wbudowane klasy Javy znajdują się w pakietach. Żadna z klas dostarczanych z Javą nie znajduje się w domyślnym pakiecie. Ponieważ nazwy klas trzeba poprzedzać nazwą pakietu, takie rozwiązanie staje się uciążliwe, jeżeli z danej klasy korzysta się bardzo często. Z tego powodu Java wykorzystuje instrukcję `import`, która umożliwia uwidocznienie konkretnych klas lub całych pakietów. Po dokonaniu importu wystarczy podać tylko i wyłącznie nazwę klasy. Instrukcja `import` jest tak naprawdę pomocą programisty. Z punktu widzenia samego języka nie trzeba jej stosować w żadnym z programów. Gdy jednak wielokrotnie w programie korzysta się z tych samych klas, importowanie pozwala zaoszczędzić mnóstwa pisania.

W plikach źródłowych Javy instrukcja importu znajduje się tuż po instrukcji `package` (o ile istnieje), a przed definicjami klas. Oto ogólna postać instrukcji `import`.

```
import pakiet1[.pakiet2].(nazwaklasa | *);
```

Element *pakiet1* to nazwa pakietu na najwyższym poziomie, natomiast element *pakiet2* to nazwa pakietu znajdującego się w głównym pakiecie. W zasadzie nie istnieje ograniczenie co do liczby poziomów pakietów, wszystko zależy od możliwości systemu plików systemu operacyjnego. Na końcu pojawia się konkretna *nazwaklasa* lub znak gwiazdki (*), który oznacza, iż kompilator powinien zaimportować cały pakiet. Poniższy fragment kodu przedstawia oba rozwiązania.

```
import java.util.Date;
import java.io.*;
```

Wszystkie standardowe klasy języka Java znajdują się w pakiecie `java`. Podstawowe metody związane z obsługą języka zostały umieszczone w pakiecie `java.lang`. Choć wszystkie inne pakiety trzeba importować w sposób jawny, Java automatycznie w sposób niejawny importuje wszystkie klasy tego pakietu. Jest to równoznaczne pisaniu poniższego wiersza na początku każdego pliku źródłowego.

```
import java.lang.*;
```

Jeśli klasa o tej samej nazwie istnieje w dwóch różnych pakietach importowanych z wykorzystaniem gwiazdki, kompilator nie zgłosi błędu. Kompilacja nie powiedzie się dopiero wtedy, gdy spróbuje się użyć jednej z tych klas. Wtedy trzeba podać pełną nazwę klasy, czyli nazwę samej klasy poprzedzoną nazwą pakietu.

Warto podkreślić, że wyrażenie `import` jest opcjonalne. Do każdej klasy można odwołać się bez importowania odpowiednich pakietów — stosując **pełne nazwy** (obejmujące wszystkie pakiety i podpakiety hierarchii). Następujący fragment kodu używa instrukcji importu.

```
import java.util.*;
class MyDate extends Date {
}
```

Poniżej znajduje się ten sam przykład bez instrukcji importu.

```
class MyDate extends java.util.Date {
}
```

W tej wersji zastosowano pełną nazwę klasy `Date`.

Zgodnie z tabelą 9.1, jeśli nie tworzy się podklasy, w momencie importu pakietu dostępne będą tylko te składowe, które zostały zadeklarowane jako `public`. Jeżeli więc klasa `Balance` pakietu `MyPack` ma być dostępna poza swoim pakietem, trzeba ją zadeklarować jako klasę publiczną i umieścić w osobnym pliku źródłowym.

```
package MyPack;

/* Teraz klasa Balance, jej konstruktor oraz metoda
   show() są publiczne. Oznacza to, że mogą być używane
   przez klasy z innych pakietów.
*/
public class Balance {
    String name;
    double bal;

    public Balance(String n, double b) {
        name = n;
        bal = b;
    }

    public void show() {
        if(bal<0)
            System.out.print("--->> ");
        System.out.println(name + ": " + bal+ " zł");
    }
}
```

Klasa `Balance` została teraz zadeklarowana z modyfikatorem `public`. Poza tym metoda `show()` i konstruktor także zostały upublicznione. Oznacza to, iż klasy z innych pakietów mają pełny dostęp do tych elementów. Klasa `TestBalance` importuje pakiet `MyPack` i może skorzystać z klasy `Balance`.

```
import MyPack.*;

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

```
/* Ponieważ klasa Balance jest klasą publiczną, można  
jej użyć oraz wywołać jej konstruktor. */  
Balance test = new Balance("J. J. Jaspers", 99.88);  
  
test.show(); // wywołanie metody show() także nie sprawi problemów  
}  
}
```

Jako eksperyment warto usunąć modyfikator `public` sprzed klasy `Balance` i spróbować ponownie skompilować kod. Kompilator zgłosi błąd.

Interfejsy

Za pomocą słowa kluczowego `interface` można w pełni oddzielić interfejs klasy od jej implementacji. Interfejs określa, co klasa musi robić, ale nie wskazuje, w jaki sposób ma te zadania wykonać. Interfejsy są składniowo podobne do klas, ale nie mogą się w nich pojawić zmienne składowe, a metody deklaruje się bez podawania jakiegokolwiek kodu. W ten sposób określa się metody, które klasa musi zaimplementować, ale nie wskazuje się w żaden sposób, jak tego dokonać. Dowolna liczba klas może implementować dany interfejs. Co więcej, jedna klasa może implementować wiele różnych interfejsów.

Aby zaimplementować interfejs, klasa musi zawierać wszystkie metody wskazane w interfejsie. Klasa może je implementować w dowolny sposób. Właśnie dzięki słowu kluczowemu `interface` język Java może w pełni realizować założenie „jeden interfejs, wiele metod”, czyli jeden z aspektów polimorfizmu.

Interfejsy zostały tak zaprojektowane, by zapewniać dynamiczne dostosowywanie się do warunków w trakcie działania programów. W tradycyjnym rozwiązaniu wywołanie metody jednej klasy przez inną klasę wymaga sprawdzenia przez kompilator, czy sygnatura wywoływanej metody jest poprawna. Gdyby nie istniało nic innego poza tym wymogiem, Java zawierałaby statyczne i nierozszerzalne środowisko klas. W takim systemie programiści dążyliby do umieszczania właściwych implementacji coraz wyżej w hierarchii klas, tak aby odpowiednie rozwiązania były dostępne dla coraz większej liczby podklas. Interfejs pozwala rozwiązać ten problem, gdyż oddziela definicję metod od hierarchii dziedziczenia. Ponieważ interfejsy tworzą inną hierarchię niż klasy, klasy niepowiązane ze sobą w hierarchii dziedziczenia mogą implementować te same interfejsy. Właśnie w tym tkwi ogromny potencjał interfejsów Javy.



Uwaga

Interfejsy Javy oferują większość rozwiązań, które w innych językach programowania (na przykład C++) można osiągnąć wyłącznie przy użyciu mechanizmu dziedziczenia wielobazowego (wielodziedziczenia).

Definiowanie interfejsu

Interfejs definiuje się podobnie jak klasę. Oto ogólna postać takiej definicji.


```
dostęp interface nazwa {  
    zwracany-typ nazwa-metody1(lista-parametrów);  
    zwracany-typ nazwa-metody2(lista-parametrów);  
    typ zmienna-finalna1 = wartość;  
    typ zmienna-finalna2 = wartość;  
    // ...  
    zwracany-typ nazwa-metodyN(lista-parametrów);  
    typ zmienna-finalnaN = wartość;  
}
```

W przypadku braku modyfikatora dostępu (elementu *dostęp*) stosuje się domyślny poziom dostępu, zatem interfejs jest dostępny tylko dla klas należących do tego samego pakietu. Zastosowanie modyfikatora `public` umożliwia stosowanie tego interfejsu w dowolnym kodzie. W takim przypadku jeden plik może zawierać tylko jeden interfejs publiczny, a nazwa tego pliku musi odpowiadać nazwie interfejsu. Element *nazwa* to nazwa interfejsu będąca dowolnym, poprawnym identyfikatorem. Zauważ, że definiowane metody nie zawierają żadnego kodu. Po liście parametrów pojawia się znak średnika. W zasadzie można powiedzieć, że są to metody abstrakcyjne, ponieważ w interfejsie nie może pojawić się implementacja żadnej z metod. Każda klasa związana z danym interfejsem musi implementować wszystkie jego metody.

Wewnątrz deklaracji interfejsu mogą pojawić się zmienne. Są one jednak niejawnie ustawiane na zmienne typu `final` i `static`, co oznacza, że nie mogą zostać zmienione przez implementującą je klasę. Co więcej, w trakcie ich deklarowania trzeba je zainicjalizować stałą. Dla wszystkich metod i zmiennych niejawnie jest stosowany modyfikator dostępu `public`.

Poniżej znajduje się przykład definicji interfejsu. Deklaruje on prosty interfejs, który zawiera jedną metodę o nazwie `callback()` przyjmującą jeden parametr typu `int`.

```
interface Callback {  
    void callback(int param);  
}
```

Implementacja interfejsu

Raz zdefiniowany interfejs może być implementowany przez wiele klas. Aby zaimplementować interfejs, trzeba dodać do klasy klauzulę `implements` oraz dodać wszystkie metody zdefiniowane w interfejsie. Ogólna postać klasy wykorzystującej interfejsy jest następująca.

```
dostęp class nazwaklasa [extends klasa-bazowa]  
    [implements interfejs[, interfejs...]] {  
    // ciało klasy  
}
```

Jeśli klasa implementuje wiele interfejsów, ich nazwy oddziela się przecinkami. Jeżeli dwa implementowane interfejsy deklarują tę samą metodę, metoda ta będzie stosowana przez klientów obu tych interfejsów. Metody implementujące interfejs muszą być metodami publicznymi. Co więcej, sygnatura typów implementowanej metody i jej deklaracji z interfejsu musi być taka sama.

Oto krótki przykład klasy implementującej zdefiniowany wcześniej interfejs `Callback`.

```
class Client implements Callback {
    //implementuje interfejs Callback
    public void callback(int p) {
        System.out.println("wywołanie callback() z wartością " + p);
    }
}
```

Zauważ, że metoda `callback()` została zadeklarowana z modyfikatorem `public`.



Implementowane metody publiczne muszą być deklarowane jako publiczne (z modyfikatorem `public`).

W pełni poprawne jest umieszczanie w klasie implementującej interfejs dodatkowych składowych, na przykład metod. Poniższa wersja klasy `Client` nie tylko implementuje metodę `callback()`, ale też dodaje własną metodę `nonIfaceMeth()`.

```
class Client implements Callback {
    //implementuje interfejs Callback
    public void callback(int p) {
        System.out.println("wywołanie callback() z wartością " + p);
    }

    void nonIfaceMeth() {
        System.out.println("Klasa implementująca interfejs " +
            "może zawierać także własne metody.");
    }
}
```

Dostęp do implementacji za pośrednictwem referencji do interfejsu

Zmienne reprezentujące referencje do obiektów można deklarować przy użyciu interfejsów (zamiast odpowiednich typów klasowych). Takiej zmiennej można przypisać egzemplarz dowolnej klasy implementującej zadeklarowany interfejs. Wywołanie metody dla takiej referencji spowoduje wywołanie odpowiedniej wersji metody należącej do właściwego egzemplarza klasy. Jest to bardzo ważny aspekt interfejsów. Metoda do wykonania jest wyszukiwana dynamicznie w trakcie działania programu, co oznacza, iż klasy można tworzyć później niż wywołujący je kod. Kod korzystający z metod interfejsu nie musi dysponować żadną wiedzą o obiekcie, którego będzie dotyczyło to wywołanie. Cała idea wywołań metod interfejsu jest bardzo podobna do wywołań metod przesłoniętych w podklasach klasy bazowej (patrz poprzedni rozdział).



Ponieważ dynamiczne wyszukiwanie metody do wykonania wiąże się ze znacznymi kosztami w porównaniu z tradycyjnymi wywołaniami metody w Javie, należy unikać stosowania interfejsów w kodzie, który ma zapewniać przede wszystkim wysoką wydajność.

Następujący kod wywołuje metodę `callback()` przy użyciu zmiennej referencyjnej interfejsu.

```
class TestIface {
    public static void main(String args[]) {
        Callback c = new Client();
        c.callback(42);
    }
}
```

Wynik działania programu jest następujący.

wywołanie callback() z wartością 42

Zauważ, że zmienna `c` jest typu `Callback`, a mimo to został jej przypisany obiekt typu `Client`. Za pośrednictwem zmiennej `c` można wywołać tylko metodę `callback()` — nie można jej użyć do uzyskania dostępu do pozostałych składowych klasy `Client`. Innymi słowy, zmienna referencyjna interfejsu „wie” tylko o metodach, które pojawiły się w deklaracji interfejsu. Z tego względu zmiennej `c` nie uda się wywołać metody `nonIfaceMeth()`, gdyż została ona zdefiniowana w klasie `Client`, ale nie w interfejsie `Callback`.

Powyższy przykład ilustruje co prawda składnię korzystania ze zmiennej referencyjnej, której typ wskazuje interfejs (nie klasę), ale nie demonstrować pełnego potencjału tego rodzaju referencji w obszarze polimorfizmu. Aby lepiej zobrazować możliwości tego mechanizmu, warto opracować drugą implementację interfejsu `Callback`.

```
// Inna implementacja interfejsu Callback.
class AnotherClient implements Callback {
    // implementuje interfejs Callback
    public void callback(int p) {
        System.out.println("Inna wersja metody callback()");
        System.out.println("p podniesione do kwadratu to " + (p*p));
    }
}
```

Wykonaj kod poniższej klasy.

```
class TestIface2 {
    public static void main(String args[]) {
        Callback c = new Client();
        AnotherClient ob = new AnotherClient();

        c.callback(42);

        c = ob; // teraz c odnosi się do obiektu AnotherClient
        c.callback(42);
    }
}
```

Program wykonujący powyższy kod wygeneruje następujące wyniki.

wywołanie callback() z wartością 42
Inna wersja metody callback()
p podniesione do kwadratu to 1764

Łatwo zauważyć, że wywoływana wersja metody `callback()` jest identyfikowana dynamicznie w zależności od typu obiektu wskazywanego przez zmienną `c` w czasie wykonywania. Choć jest to bardzo prosty przykład, wkrótce przedstawię bardziej praktyczne zastosowanie.

Implementacja częściowa

Klasa, która łączy pewien interfejs, ale nie implementuje wszystkich zdefiniowanych w nim metod, musi zostać zadeklarowana jako klasa abstrakcyjna. Oto przykład.

```
abstract class Incomplete implements Callback {
    int a, b;
    void show() {
        System.out.println(a + " " + b);
    }
    //...
}
```

Ponieważ klasa `Incomplete` nie implementuje metody `callback()`, musi zostać zadeklarowana jako klasa abstrakcyjna (modyfikator `abstract`). Klasy dziedziczące po `Incomplete` muszą albo zaimplementować metodę `callback()`, albo same zostać zadeklarowane jako abstrakcyjne.

Interfejsy zagnieżdżone

Interfejs można zadeklarować w formie składowej klasy lub innego interfejsu. Taki interfejs określa się mianem **interfejsu składowego** lub **interfejsu zagnieżdżonego**. Interfejs zagnieżdżony można zadeklarować z modyfikatorem dostępu `public`, `private` lub `protected`. W tym aspekcie interfejsy zagnieżdżone różnią się od interfejsów najwyższego poziomu, które — jak już wspomniano — należy deklarować albo z modyfikatorem `public`, albo bez modyfikatora dostępu (wówczas jest stosowany domyślny poziom dostępu). Interfejs zagnieżdżony stosowany poza zasięgiem, w którym został zadeklarowany, musi być identyfikowany przez pełną nazwę, tj. poprzedzony nazwą klasy (lub interfejsu), której jest członkiem. Oznacza to, że poza klasą lub interfejsem, w którym zadeklarowano interfejs zagnieżdżony, należy stosować pełną nazwę tego interfejsu.

Oto prosty przykład ilustrujący interfejs zagnieżdżony:

```
// Przykład interfejsu zagnieżdżonego.

// Klasa A zawiera interfejs składowy.
class A {
    // to jest interfejs zagnieżdżony
    public interface NestedIF {
        boolean isNotNegative(int x);
    }
}

// Klasa B implementuje interfejs zagnieżdżony.
class B implements A.NestedIF {
    public boolean isNotNegative(int x) {
        return x < 0 ? false: true;
    }
}

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

        // użycie referencji typu interfejsu zagnieżdżonego
        A.NestedIF nif = new B();

        if(nif.isNotNegative(10))
            System.out.println("liczba 10 nie jest ujemna ");
        if(nif.isNotNegative(-12))
```

```

        System.out.println("to nie zostanie wyświetlone ");
    }
}

```

Jak widać, klasa A definiuje interfejs składowy nazwany `NestedIF` i zadeklarowany jako interfejs publiczny. Poniższy zapis oznacza, że klasa B implementuje ten interfejs zagnieżdżony:

```
implements A.NestedIF
```

Warto zwrócić uwagę na pełną nazwę tego interfejsu (obejmującą nazwę klasy, do której należy). W ciele metody `main()` utworzono zmienną referencyjną typu `A.NestedIF` nazwaną `nif`, po czym przypisano tej zmiennej referencję obiektu klasy B. Takie rozwiązanie jest dopuszczalne, ponieważ klasa B implementuje interfejs `A.NestedIF`.

Stosowanie interfejsów

Aby lepiej zrozumieć siłę drzemiącą w interfejsach, warto przyjrzeć się bardziej praktycznemu przykładowi. We wcześniejszych rozdziałach pojawiła się klasa o nazwie `Stack`, która implementowała stos o określonym rozmiarze. Istnieje wiele różnych sposobów implementacji stosu. Na przykład stos może mieć stały rozmiar lub rozszerzać się. Co więcej, wartości ze stosu można przechowywać w tablicy, liście, drzewie binarnym itp. Niezależnie od implementacji interfejs stosu zawsze pozostaje taki sam. Innymi słowy, metody `pop()` i `push()` działają tak samo niezależnie od sposobu przechowywania danych na stosie. Z tego względu warto wydzielić interfejs stosu, a implementację pozostawić wyspecjalizowanym klasom. Przyjrzyjmy się dwóm przykładom.

Pierwszy kod to interfejs definiujący stos liczb całkowitych. Kod należy umieścić w pliku źródłowym *IntStack.java*. Interfejs zostanie wykorzystany przez dwie różne implementacje stosu.

```

// Definicja interfejsu dla stosu liczb całkowitych.
interface IntStack {
    void push(int item); //zapamiętanie elementu
    int pop(); //pobranie elementu
}

```

Następujący program tworzy klasę o nazwie `FixedStack`, która implementuje stos o stałym rozmiarze.

```

// Implementacja IntStack używająca tablicy o stałym rozmiarze.
class FixedStack implements IntStack {
    private int stck[];
    private int tos;

    // alokacja i inicjalizacja stosu
    FixedStack(int size) {
        stck = new int[size];
        tos = -1;
    }

    // umieszczenie elementu na szczycie stosu
    public void push(int item) {
        if(tos==stck.length-1) //użycie zmiennej składowej length

```

```

        System.out.println("Stos jest pełny.");
    else
        stck[++tos] = item;
    }

    //zdjęcie elementu ze szczytu stosu
    public int pop() {
        if(tos < 0) {
            System.out.println("Stos nie zawiera żadnych elementów.");
            return 0;
        }
        else
            return stck[tos--];
    }
}

class IFTest {
    public static void main(String args[]) {
        FixedStack mystack1 = new FixedStack(5);
        FixedStack mystack2 = new FixedStack(8);

        //umieszczenie pewnych liczb na stosach
        for(int i=0; i<5; i++) mystack1.push(i);
        for(int i=0; i<8; i++) mystack2.push(i);

        //zdjęcie liczb ze stosów
        System.out.println("Stos w mystack1:");
        for(int i=0; i<5; i++)
            System.out.println(mystack1.pop());

        System.out.println("Stos w mystack2:");
        for(int i=0; i<8; i++)
            System.out.println(mystack2.pop());
    }
}

```

Poniżej znajduje się druga implementacja interfejsu `IntStack`, która tworzy stos o dynamicznie zmieniającym się rozmiarze. W tej implementacji tworzony jest stos z pewnym rozmiarem początkowym. Jeśli zostanie on przekroczony, stos jest rozszerzany — jego rozmiar wzrasta dwukrotnie.

```

//Implementacja rozszerzającego się stosu.
class DynStack implements IntStack {
    private int stck[];
    private int tos;

    //alokacja i inicjalizacja stosu
    DynStack(int size) {
        stck = new int[size];
        tos = -1;
    }

    //umieszczenie elementu na szczycie stosu
    public void push(int item) {
        //jesli stos jest pełny, alokacja nowej tablicy o większym rozmiarze
        if(tos==stck.length-1) {

```

```

        int temp[] = new int[stck.length * 2]; //podwojenie rozmiaru
        for(int i=0; i<stck.length; i++) temp[i] = stck[i];
        stck = temp;
        stck[++tos] = item;
    }
    else
        stck[++tos] = item;
}

//zdjęcie elementu ze szczytu stosu
public int pop() {
    if(tos < 0) {
        System.out.println("Stos nie zawiera żadnych elementów.");
        return 0;
    }
    else
        return stck[tos--];
}
}

class IFTest2 {
    public static void main(String args[]) {
        DynStack mystack1 = new DynStack(5);
        DynStack mystack2 = new DynStack(8);

        //pętla powodującą zwiększenie rozmiaru stosu
        for(int i=0; i<12; i++) mystack1.push(i);
        for(int i=0; i<20; i++) mystack2.push(i);

        System.out.println("Stos w mystack1:");
        for(int i=0; i<12; i++)
            System.out.println(mystack1.pop());

        System.out.println("Stos w mystack2:");
        for(int i=0; i<20; i++)
            System.out.println(mystack2.pop());
    }
}

```

Poniższy kod programu korzysta z obu implementacji: `FixedStack` i `DynStack`. Korzysta przy tym ze zmiennej referencyjnej interfejsu. Oznacza to, że wywoływane metody `pop()` i `push()` są identyfikowane dynamicznie (w czasie wykonywania programu) zamiast w trakcie kompilacji.

```

/* Utworzenie zmiennej referencyjnej interfejsu i
   użycie jej do dostępu do stosu.
*/
class IFTest3 {
    public static void main(String args[]) {
        IntStack mystack; //utworzenie zmiennej referencyjnej interfejsu
        DynStack ds = new DynStack(5);
        FixedStack fs = new FixedStack(8);

        mystack = ds; //załadowanie stosu dynamicznego
        //umieszczenie wartości na stosie
        for(int i=0; i<12; i++) mystack.push(i);
    }
}

```

```

        mystack = fs; //załadowanie stosu o stałym rozmiarze
        for(int i=0; i<8; i++) mystack.push(i);

        mystack = ds;
        System.out.println("Wartości na stosie dynamicznym:");
        for(int i=0; i<12; i++)
            System.out.println(mystack.pop());

        mystack = fs;
        System.out.println("Wartości na stosie o stałym rozmiarze:");
        for(int i=0; i<8; i++)
            System.out.println(mystack.pop());
    }
}

```

W tym programie zmienna `mystack` jest typu `IntStack`. Jeśli więc zawiera taką samą wartość jak `ds`, korzysta z metod `push()` i `pop()` zdefiniowanych dla klasy `DynStack`. Jeśli zawiera taką samą wartość jak `fs`, korzysta z metod `push()` i `pop()` zdefiniowanych dla klasy `FixedStack`. Jak już wspomniałem, wybór wersji metody dokonywany jest w sposób dynamiczny w trakcie wykonywania programu. Dostęp do wielu implementacji interfejsu przez zmienne referencyjne to najbardziej użyteczny sposób wykorzystania polimorfizmu w języku Java.

Zmienne w interfejsach

Można użyć interfejsów do wprowadzenia w wielu klasach tych samych stałych. Wystarczy w interfejsie zadeklarować i zainicjalizować zmienne. Klasa implementująca taki interfejs będzie w swoim zasięgu „widziała” wszystkie te zmienne jako stałe. (Przypomina to tworzenie stałych dyrektywą `#define` lub deklaracją `const` w językach C lub C++). Jeśli interfejs nie zawiera żadnych metod, wtedy implementująca go klasa nie musi deklarować żadnych dodatkowych metod. Zmienne występujące w interfejsie są widziane przez klasę korzystającą z interfejsu jako zmienne typu `final`. Kolejny przykład używa interfejsu do zaimplementowania automatu podejmującego decyzje.

```

import java.util.Random;

interface SharedConstants {
    int NO = 0;
    int YES = 1;
    int MAYBE = 2;
    int LATER = 3;
    int SOON = 4;
    int NEVER = 5;
}

class Question implements SharedConstants {
    Random rand = new Random();
    int ask() {
        int prob = (int) (100 * rand.nextDouble());
        if (prob < 30)
            return NO;           // 30%
        else if (prob < 60)
            return YES;          // 30%
        else if (prob < 70)
            return MAYBE;        // 10%
    }
}

```



```

        else if (prob < 85)
            return LATER;           // 15%
        else if (prob < 98)
            return SOON;            // 13%
        else
            return NEVER;           // 2%
    }
}

class AskMe implements SharedConstants {
    static void answer(int 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());
}
}

```

Zauważ, że program korzysta z jednej ze standardowych klas Javy — klasy `Random`. Klasa generuje liczby pseudolosowe. Zawiera kilka metod zwracających wylosowane liczby w formacie najbardziej odpowiednim dla danego programu. W tym przykładzie stosujemy metodę `nextDouble()`, która zwraca losową wartość zmiennoprzecinkową w przedziale od 0.0 do 1.0.

W przykładowym programie dwie klasy, `Question` i `AskMe`, implementują interfejs `SharedConstants` definiujący stałe `YES`, `NO`, `MAYBE`, `SOON`, `LATER` i `NEVER`. Kod wewnątrz klas korzysta z tych zmiennych w taki sposób, jakby były one zadeklarowane w tej klasie lub odziedziczone po innej klasie. Zauważ, że wyniki działania programu są różne po każdym jego uruchomieniu.

```

Wkrótce
Później
Tak
Nie

```

Interfejsy można rozszerzać

Jeden interfejs może dziedziczyć po innym interfejsie. Wystarczy w tym celu użyć słowa kluczowego `extends`. Składnia jest dokładnie taka sama jak w przypadku dziedziczenia po klasie. Gdy klasa implementuje interfejs dziedziczący po innym interfejsie, musi zapewnić wszystkie metody zdefiniowane w całym łańcuchu dziedziczenia. Oto przykład.

```
//Jeden interfejs rozszerza inny.
interface A {
    void meth1();
    void meth2();
}

//teraz B zawiera meth1() i meth2() oraz dodaje własne meth3()
interface B extends A {
    void meth3();
}

//klasa musi zaimplementować wszystkie metody zdefiniowane w interfejsach A i B
class MyClass implements B {
    public void meth1() {
        System.out.println("Implementacja meth1().");
    }

    public void meth2() {
        System.out.println("Implementacja meth2().");
    }

    public void meth3() {
        System.out.println("Implementacja meth3().");
    }
}

class IFExtend {
    public static void main(String arg[]) {
        MyClass ob = new MyClass();

        ob.meth1();
        ob.meth2();
        ob.meth3();
    }
}
```

Jako eksperyment można spróbować usunąć implementację metody `meth1()` z klasy `MyClass`. Spowoduje to zgłoszenie błędu kompilacji. Dowolna klasa implementująca dany interfejs musi zdefiniować kod wszystkich metod interfejsu, także tych, które interfejs odziedziczył po innym interfejsie.

Choć przykłady przedstawione w książce niezbyt często korzystają z pakietów i interfejsów, w rzeczywistych programach Javy oba elementy są niezwykle ważne. W zasadzie wszystkie poważne programy Javy umieszcza się w pakietach. Większość z tych programów definiuje własne interfejsy. Warto więc dobrze zapoznać się z tymi kwestiami.

Skorowidz

A

- abstrakcja, 48
- adnotacja, 308, 321
 - @Deprecated, 320
 - @Documented, 319
 - @Inherited, 320
 - @Override, 320
 - @Retention, 319
 - @SafeVarargs, 320
 - @SuppressWarnings, 321
 - @Target, 319
 - jednoelementowa, 318
 - wbudowana, 319
 - znacznikowa, 317
- adnotacje ze strategią
 - CLASS, 309
 - RUNTIME, 309
 - SOURCE, 309
- adres, 712
- adres 127.0.0.1, 991
- adres IP, 712
- adres URL, 1067
- akcja, 51
- algorytm harmonogramowania wątków, 272
- algorytmy, 488
- algorytmy w klasie Collections, 530
- analyzer leksykalny, 559
- Android, 46
- animacja poklatkowa, 916
- anonimowa klasa wewnętrzna, 785
- ANSI, 33
- aplet
 - RegPay, 1124
 - RemBal, 1120
 - SimpleApplet, 343
 - z paskiem reklamowym, 744, 750
- aplety, 37, 341, 716, 733, 794, 1026, 1095
- architektura, 735
 - składniki, 740
 - szkielet, 738
- aplikacja typu klient-serwer, 987
 - instalacja plików, 990
 - kod źródłowy, 988
 - rejestr RMI, 990
 - uruchomienie klienta, 991
 - uruchomienie serwera, 990
- appletviewer, 734
- architektura
 - apletu, 735
 - model-delegacja, 1014
 - model-widok-kontroler, 1014
 - MVC, 1014
 - z odrębnym modelem, 1014
- argument, 152
 - append, 653
 - array, 654
 - fileObj, 653
 - filePath, 653
 - formatString, 993
 - locale, 992
 - outputStream, 660
 - style, 992
 - wieloznaczny, 374
 - wieloznaczny ograniczony, 375, 377
- argumenty wiersza poleceń, 186
- ARM, automatic resource management, 338, 484
- ASCII, 409
- asercja, 351
 - włączanie, 354
 - wyłączanie, 354
- asynchroniczne wykonywanie zadań, 968
- autoanaliza programu, 983
- automatyczna konwersja typu, 81

automatyczne opakowywanie, 490
 typów logicznych, 306
 typów prostych, 303
 typów znakowych, 306
 w wyrażeniach, 304
automatyczne rozpakowywanie, 304, 305
automatyczne rozszerzanie typów, 83
automatyczne wydobywanie typów prostych, 303
automatyczne zarządzanie zasobami, 338, 484
AWT, Abstract Window Toolkit, 342, 787, 1012

B

bardziej znaczący surogat, 451
barwa, 809
barwa-nasylenie-jasność, 809
bezpieczeństwo, 37, 351, 387, 464, 696
bezpieczeństwo kolekcji, 540
biblioteka API Javy, 45
biblioteka AWT, 342, 733
biblioteka Swing, 342, 733, 1012, 1015, 1065
biblioteki klas Javy, 65
blok kodu, 61
blok synchronizujący, 276
blokada, 951
blokada wielobieźna, 952
blokada wzajemna, deadlock, 282
błąd niejednoznaczności, 400
błędy systemowe, 286
BMP, basic multilingual plane, 451
buforowanie operacji wejścia-wyjścia, 640
bufory, 672

C

CGI, Common Gateway Interface, 39, 1067
ciało metody, 57
cykl życia serwletu, 1068
czcionki, 812, 817
 nazwa, 815
 rozmiar, 815
 styl, 815
 terminy, 818
czcionki dostępne, 813

D

dane, 142
dane sterujące dostępem do kodu, 48
dane wejściowe, 619
dane wyjściowe, 594, 619, 754
datagramy, 728, 730
definicja klasy, 141
definiowanie interfejsu, 228
definiowanie ograniczenia, 373
definiowanie pakietu, 220

deklaracja metody kodu rdzennego, 348
deklaracja tablicy, 91
deklaracja zmiennej sterującej, 126
deklarowanie obiektów, 145
dekoder, 675
dekrementacja, 96
delegacja interfejsu użytkownika, 1014
delegowanie zdarzeń, 756, 775
deserializacja, 664
długość łańcucha, 410
dodanie metody, 148, 151
domena, 714
domyślny poziom równoległości, 972
dopasowywanie do wzorca, 977, 983
doprecyzowane zgłaszanie dalej, 257
dostęp
 asynchroniczny, 277
 do kolekcji, 510
 do składowych, 195
 do składowych klasy, 223
 domyślny, 175
 prywatny, 175
 publiczny, 175
 wielowątkowy, 276
dowiązanie późne, 216
dowiązanie wczesne, 216
drzewo, 1059
drzewo katalogów, 701
dwa parametry typu, 370
dynamiczne przydzielanie metod, 209
dynamiczny polimorfizm, 211
dziedziczenie, inheritance, 50, 193, 206
dzieli i zwyciężaj, 959

E

egzekutor, 923, 944
egzemplarz klasy, 49, 141
elipsa, 804
emerytura, 1116
etykieta, 136, 829

F

fałszywe budzenie, 278
FIFO, 496
filtr, 625
 Blur, 913
 Contrast, 910
 Grayscale, 909
 Invert, 909
 konwolucyjny, 911
 Sharpen, 916
filtrowanie obrazu, 906
finalizacja, 158

- format
 - GIF, 888
 - JPEG, 888
 - PNG, 888
- formatowanie
 - daty i czasu, 994
 - daty i godziny, 589
 - liczb, 588
 - łańcuchów i znaków, 588
 - tekstu, 973, 991
- framework Collections, 361, 488, 516, 923
- framework Fork/Join, 45, 921, 955
- framework NIO, 45
- FTP, File Transfer Protocol, 1130
- funkcja matches(), 976
- funkcja mieszająca, 505
- funkcja randomColor(), 1008
- funkcje trygonometryczne, 469
- funkcje wirtualne, 211
- funkcje wykładnicze, 470
- funkcje zaokrąglenia, 471

G

- gniazda klientów TCP/IP, 716
- gniazda serwerów TCP/IP, 727
- gniazdo, socket, 711, 716
- gniazdo Berkeley, 711
- graficzny interfejs użytkownika, 733, 755
- grupa wątków, 264
- grupowe rozgłaszanie, 757
- GUI, 733, 755

H

- hasła języka Java, 40
- hermetyzacja, encapsulation, 49
- hierarchia dziedziczenia, 203
- hierarchia klas, 51
- hierarchia klas sparametryzowanych, 388
- hierarchia klas wyjątków, 241
- histogram, 901
- HSB, Hue-Saturation-Brightness, 809
- HTTP, HyperText Transfer Protocol, 1130

I

- identyfikator klasy, 55
- identyfikatory, 63
- implementacja interfejsu, 229
- import pakietów, 226
- import statyczny, 354, 356
- indeks argumentu, 597
- indeks względny, 598
- informacje o ścieżce, 696

- inkrementacja, 96
- instrukcja
 - break, 136
 - continue, 138
 - do-while, 122
 - for, 125
 - for-each, 128
 - goto, 136
 - if, 113
 - if-else-if, 115
 - package, 220
 - pusta, 122
 - return, 139
 - switch, 116
 - throw, 247
 - while, 121
- instrukcje iteracyjne
 - do-while, 122
 - for, 125
 - for-each, 128
 - pętle zagnieżdżone, 133
 - while, 121
 - zmienna sterująca pętlą, 125
- instrukcje skoku
 - break, 134
 - continue, 138
 - return, 139
- instrukcje wyboru
 - if, 113
 - if-else-if, 115
 - switch, 116
 - zagnieżdżone if, 115
 - zagnieżdżone switch, 120
- interfejs
 - ActionListener, 772, 1024
 - AdjustmentListener, 773, 844
 - AnnotatedElement, 315
 - API, 922
 - Appendable, 483
 - AppletContext, 752
 - AppletStub, 754
 - AudioClip, 754
 - AutoCloseable, 333, 484, 599, 607, 627, 634
 - BeanInfo, 1002
 - Callable, 947
 - CGI, 1067
 - CharSequence, 483
 - Cloneable, 464
 - Closeable, 627, 634
 - Collection, 491
 - Comparable, 483
 - Comparator, 526
 - ComponentListener, 773
 - Concurrent API, 921

interfejsy

- ContainerListener, 773
- Customizer, 1004
- Deque, 496
- Enumeration, 542
- Executor, 923
- Externalizable, 664
- FilenameFilter, 625
- Flushable, 627, 634
- FocusListener, 773
- Future, 947
- HttpServletRequest, 1079, 1090
- HttpServletResponse, 1079
- HttpSession, 1080
- HttpSessionBindingListener, 1081
- ImageConsumer, 900, 902
- ImageObserver, 891
- ImageProducer, 898, 902
- ItemListener, 773, 839
- Iterable, 484
- Iterator, 510
- Java Beans API, 1004
- java.io.Serializable, 1004
- KeyListener, 774, 779
- LayoutManager, 851
- List, 493
- ListSelectionModel, 1054
- Map, 517
- Map.Entry, 519
- MinMax, 383
- MouseListener, 774
- MouseMotionListener, 774
- MouseWheelListener, 774, 776
- NavigableMap, 519
- NavigableSet, 495
- ObjectInput, 665
- ObjectOutput, 664
- Observer, 577
- Path, 621, 676
- PluginFilter.java, 907
- Queue, 496
- RandomAccess, 516
- Readable, 484, 600
- ReadableByteChannel, 600
- ReadWriteLock, 954
- Runnable, 265, 473
- Serializable, 664
- Servlet, 1074
- Servlet API, 1073
- ServletConfig, 1074
- ServletContext, 1074
- ServletRequest, 1075, 1077
- ServletResponse, 1075
- Set, 494

- SortedMap, 518
- SortedSet, 495
- TextListener, 775
- Thread.UncaughtExceptionHandler, 485
- TreeNode, 1059
- TreeSelectionListener, 1059
- WindowFocusListener, 775
- WindowListener, 775

interfejsy, 228

- dziedziczenie, 238
- implementacja, 229
- implementacja częściowa, 231
- zmienne, 236

interfejsy

- atributów plików, 680
- menedżera pobierania, 1131
- kolekcji, 490
- map, 517
- nasłuchujące, 782
- nasłuchujące zdarzeń, 772
- pakietu java.beans, 1005
- pakietu java.lang, 435
- pakietu java.util, 616
- pakietu javax.servlet, 1073
- pakietu javax.servlet.http, 1079
- publiczne klasy, 49
- sparametryzowane, 383
- składowe, 232
- sparametryzowane, 362
- stosu, 233
- strumieni Javy, 669
- zagnieżdżone, 232

interpreter kodu bajtowego, 38

introspekcja, 486, 1000

IP, Internet Protocol, 712

iterator, 484, 489, 510, 551

iterator Iterator, 512

J

J2SE 5, 43

jasność, 809

java Beans, 999

Java SE, 44

Java SE 7, 44

javac, 54

jądro konwolucji, 912

JCP, Java Community Process, 46

JDK 6, 44

jednostka kompilacji, 54

język

- BASIC, 32

- BCPL, 32

- C, 32

C#, 36
C++, 33, 1129
COBOL, 32
FORTRAN, 32
Java, 34
JFC, Java Foundation Classes, 1012
JNI, Java Native Interface, 348
JNLP, Java Network Launch Protocol, 734
JVM, Java Virtual Machine, 38

K

kanal, 673, 684, 688
kanal w trybie odczytu i zapisu, 691
katalog, 624
katalog bin, 1070
katalog MyPackage, 220
klasa, 49
 AbstractButton, 1039
 AbstractList, 542
 ActionEvent, 759, 1024
 AdjustmentEvent, 761
 Applet, 733, 735
 ArrayDeque, 509
 ArrayList, 500
 Arrays, 534
 AtomicInteger, 954
 AtomicLong, 954
 AWTEvent, 879
 bazowa, 193
 BeingWatched, 579
 BitSet, 561
 Blur.java, 913
 Boolean, 301, 451
 BorderLayout, 854
 BufferedInputStream, 640
 BufferedOutputStream, 642
 BufferedReader, 657
 BufferedWriter, 658
 Byte, 440
 ByteArrayInputStream, 636
 ByteArrayOutputStream, 638
 ByteBuffer, 674, 684
 Calendar, 566
 Canvas, 792
 CardLayout, 858
 Character, 301, 448, 450
 CharArrayReader, 654
 CharArrayWriter, 655
 Checkbox, 834
 CheckboxGroup, 836
 CheckboxMenuItem, 866
 Choice, 838
 Class, 466
 ClassLoader, 468
 ClassValue, 482
 Color, 741, 808, 810
 Compiler, 472
 Component, 790, 888, 1028
 ComponentEvent, 762
 Console, 661
 Container, 791
 ContainerEvent, 762
 Contrast.java, 910
 Convolver, 912
 Convolver.java, 911
 Cookie, 1081, 1082
 CookieHandler, 727
 CookieManager, 727
 Coords, 376
 CountDownLatch, 930
 CropImageFilter, 903
 Currency, 582
 CyclicBarrier, 931
 DatagramPacket, 729
 DatagramSocket, 728
 DataInputStream, 649
 DataOutputStream, 648
 Date, 564
 DateFormat, 991
 Dictionary, 548
 Double, 436
 Download, 1131
 konstruktor, 1135
 metoda download(), 1135
 metoda run(), 1135
 metoda stateChanged(), 1138
 metody akcesorów i działań, 1139
 zmienne, 1135
 DownloadManager, 1130, 1144
 konstruktor, 1149
 metoda tableSelectionChanged(), 1150
 metoda updateButtons(), 1151
 metoda verifyUrl(), 1150
 obsługa zdarzeń akcji, 1152
 zmienne, 1149
 DownloadsTableModel, 1140
 metoda addDownload(), 1142
 metoda clearDownload(), 1142
 metoda getColumnClass(), 1143
 metoda getValueAt(), 1143
 metoda update(), 1143
 Enum, 482
 EnumMap, 526
 EnumSet, 510
 Error, 240
 EventSetDescriptor, 1006
 Exchanger, 934
 File, 621

klasa

- FileChannel, 675, 684
- FileDialog, 877
- FileInputStream, 332, 333, 632, 705
- FileNotFoundException, 628
- FileOutputStream, 332, 336, 634
- FileReader, 600, 651
- Files, 678
- FileStore, 682
- FileSystem, 682
- FileSystems, 682
- FileWriter, 653
- Float, 437
- FlowLayout, 852
- FocusEvent, 763
- FontMetrics, 818
- ForkJoinPool, 958, 968, 971
- ForkJoinTask, 956, 970, 972
- Formatter, 583, 599
 - formatowanie, 585
 - konstruktor klasy, 584
 - zamykanie obiektu, 599
 - znaczniki formatów, 594
- Frame, 790, 792
- Gen, 362, 366
- GenDemo, 362
- GenericServlet, 1072, 1075
- Graphics, 802, 892
- Grayscale.java, 909
- GregorianCalendar, 569
- GridBagConstraints, 862
- GridBagLayout, 861
- GridLayout, 857
- HashMap, 522
- HashSet, 505
- Hashtable, 549
- HttpCookie, 727
- HttpServlet, 1083, 1085
- HttpSession, 1090
- HttpSessionBindingEvent, 1084
- HttpSessionEvent, 1083
- URLConnection, 724
- IdentityHashMap, 526
- Image, 887
- ImageFilter, 902
- ImageFilterDemo.java, 906
- ImageIcon, 1036
- Inet4Address, 716
- Inet6Address, 716
- InetAddress, 714
 - metody fabryczne, 714
- InheritableThreadLocal, 479
- Inner, 182
- InputEvent, 764
- InputStream, 630
- Integer, 440
- Introspector, 1004
- Invert.java, 909
- ItemEvent, 765
- java.lang.Enum., 297
- JButton, 1040
- JCheckBox, 1045
- JComboBox, 1056
- JComponent, 1015
- JFrame, 1020
- JLabel, 1036
- JList, 1053
- JRadioButton, 1047
- JScrollPane, 1051
- JTabbedPane, 1049
- JTable, 1062
- JTextField, 1038
- JToggleButton, 1042
- KeyEvent, 765
- Label, 829
- LinkedHashMap, 525
- LinkedHashSet, 506
- LinkedList, 504, 515
- List, 840
- ListResourceBundle, 614
- ListSelectionEvent, 1054
- LoadedImage.java, 907
- Locale, 572
- Long, 440
- Matcher, 976
- Math, 354, 469
 - funkcje trygonometryczne, 469
 - funkcje wykładnicze, 469
 - funkcje zaokrągleń, 470
- MediaTracker, 895, 896
- MemoryImageSource, 898, 899
- Menu, 872
- MenuItem, 866
- MethodDescriptor, 1007
- MouseEvent, 766
- MouseEvents, 778
- MouseMotionAdapter, 782
- MouseWheelEvent, 768
- MyFileVisitor, 703
- MyGenClass, 400
- NewThread, 287
- Number, 301, 436
- Object, 217, 463
- ObjectInputStream, 665
- ObjectOutputStream, 664
- Observable, 576
- OutputStream, 330, 630, 631
- Package, 479

- Panel, 791
- Paths, 679
- Pattern, 975
- Phaser, 936
- PixelGrabber, 900
- PopupMenu, 872
- PrintStream, 645
- PrintWriter, 331, 599, 660
- PriorityQueue, 508
- Process, 452
- ProcessBuilder, 457
- ProgressRenderer, 1139
- Properties, 552
- PropertyDescriptor, 1006
- PropertyResourceBundle, 614
- PushbackInputStream, 642
- PushbackReader, 659
- Random, 574
- RandomAccessFile, 650, 708
- Reader, 327, 651
- RecursiveAction, 957, 962
- RecursiveTask, 958, 966
- RegPayS, 1124
- ResourceBundle, 611
- RGBImageFilter, 905
- Runtime, 454
 - zarządzanie pamięcią, 454
- Scanner, 600, 604, 607, 610
 - konstruktor klasy, 600
 - skanowanie, 600
 - ustawianie separatorów, 609
- Scrollbar, 843
- SecurityManager, 481
- Semaphore, 924
- SequenceInputStream, 644
- ServerSocket, 727
- ServletInputStream, 1075
- ServletOutputStream, 1077
- Sharpen.java, 915
- Short, 440
- SimpleDateFormat, 993
- SimpleTimeZone, 571
- Socket, 716
- Stack, 176, 546
- StackTraceElement, 481
- Stats, 371
- StrictMath, 471
- String, 183, 425
 - długość łańcucha, 185
 - klasa pomocnicza StringBuffer, 184
 - konkatenacja łańcuchów, 184
 - niezmiennosc obiektów, 184
 - porównywanie łańcuchów, 185
 - stała tekstowa, 184
 - tworzenie obiektów, 184
 - znak łańcucha, 185
- StringBuffer, 184, 407, 425
- StringBuilder, 407, 433
- StringTokenizer, 559
- SwingUtilities, 1022
- System, 326, 459
- TComp, 529
- TextArea, 849
- TextComponent, 849
- TextEvent, 769
- TextField, 847
- Thread, 262, 267, 286, 473
- ThreadGroup, 473
- ThreadLocal, 479
- Throwable, 240, 243, 480
- Timer, 580
- TimerTask, 580
- TimeZone, 570
- TreeMap, 523
- TreePath, 1059
- TreeSet, 507
- TwoGen, 369
- URI, 727
- URLConnection, 722
- Vector, 542
- Void, 451
- Window, 792
- WindowEvent, 770
- Writer, 651
- klasy, 141, 222
 - abstrakcyjne, 213
 - adapterów, 782
 - AWT, 788
 - dodatkowe pakietu java.util, 616
 - frameworku Fork/Join, 956
 - implementujące interfejsy map, 522
 - kolekcji, 500
 - map, 519
 - nadrzędne, 51, 193
 - pakietu java.awt.event, 760
 - pakietu java.awt.image, 887
 - pakietu java.beans, 1005
 - pakietu java.io, 620
 - pakietu java.lang, 435
 - pakietu java.lang.reflect, 984
 - pakietu java.net, 713
 - pakietu java.util, 487, 542
 - pakietu java.util.concurrent, 923
 - pakietu javax.servlet, 1073
 - pakietu javax.servlet.http, 1079
 - pakietu javax.swing, 1015, 1035
 - potomne, 193
 - sparametryzowane, 362, 369, 377, 394

- klasy
 - strumieni, 630
 - strumieni bajtów, 324
 - strumieni znaków, 325
 - systemu NIO, 671
 - wejścia-wyjścia, 620
 - wewnętrzne, 181, 784
 - wewnętrzne anonimowe, 183
 - zagnieżdżone, 181
 - niestatyczne, 181
 - statyczne, 181
 - zdarzeń, 758
 - zewnętrzne, 181
- klauzula catch, 244
- klauzula extends, 377, 388
- klauzula throws, 249
- klonowanie, 464
- kod asemblerowy, 32
- kod bajtowy, 38
- kod działający na danych, 47
- kod klasy, 141
- kod kraju, 612
- kod źródłowy serwletu, 1071, 1088
- koder, 675
- kody języków, 612
- kolejka FIFO, 496
- kolejka LIFO, 159
- kolejkowanie, 277
- kolejność wykonywania operatorów, 111
- kolekcje, 361, 488
- kolekcje modyfikowalne, 491
- kolekcje niemodyfikowalne, 491
- kolekcje współbieżne, 950
- kolory, 741
- koło, 804
- komentarz, 55, 64
 - dokumentujący, 55, 1155, 1160
 - jednowierszowy, 55
 - wielowierszowy, 55
- komparator, 526
- kompilator, 51, 54
- kompilator javac, 54
- komponent, 1015
- komponent Java Bean, 1000, 1007
- komponent wizualny, 1014
- komponenty biblioteki Swing, 1015, 1035
- komunikacja międzyprocesowa, 278
- komunikacja międzywątkowa, 262, 277
- konkatenacja łańcuchów, 411
- konstrukcja <?>, 310
- konstrukcja Stats<?>, 374
- konstrukcja try, 245
- konstruktor, 154
- konstruktor BoxWeight(), 199
- konstruktor klasy, 146
- konstruktor klasy bazowej, 199
- konstruktor klasy String, 408, 410
- konstruktor sparametryzowany, 155, 382
- konstruktory klasy ServerSocket, 728
- konstruktory klasy Socket, 717
- konstruktory klasy StringBuffer, 425
- kontekst graficzny, 802
- kontener, 1016
- kontener (serwer) serwletów, 1069
- kontener Glassfish, 1069
- kontener najwyższego poziomu, 1015
- kontener Tomcat, 1069
- kontrola dostępu, 173
- kontroler, 1014
- kontrolka
 - Button, 879
 - Checkbox, 880
 - Choice, 882
 - edycji, 846
 - JTextField, 1038
 - List, 883
 - Scrollbar, 884
 - TextArea, 849
 - TextField, 848
- kontrolki, 827
 - etykiety, 828
 - listy, 828
 - listy rozwijane, 828
 - menu, 868
 - paski przewijania, 828
 - polą tekstowe, 828
 - polą wyboru, 828
 - przyciski, 828
 - typy, 828
- konwersja
 - danych, 423
 - formatów, 583
 - liczb, 446
 - łańcuchów, 412
 - rozszerzająca, 81
 - typów, 81
 - zawężająca, 82
- kopiowanie pliku, 692
- kredyt, 1120
- krzywa Gaussa., 575
- kwantyfikator, 979

L

- LayoutManager, 908
- lewy ukośnik (\), 621
- liczby pseudolosowe, 574
- LIFO, 496
- lista, 840, 1053

lista kombinowana, 1056
lista rozwijana, 838
literał, 977
literał tekstowy, 410

L

łańcuch, 92
łańcuch polecenia akcji, 1040
łańcuch wyjątków, 255
łańcuch zapytania, query string, 1086
łączenie łańcuchów, 184, 411
łączenie łańcuchów z innymi typami danych, 411
łuk, 805

M

mapa, 489, 516
maszyna wirtualna Javy, 38
mechanizm odzyskiwania pamięci, 157
menedżer pobierania plików, 1129, 1153
menedżer układu graficznego, 827, 852

- BorderLayout, 854, 1032
- CardLayout, 858
- FlowLayout, 852
- GridBagLayout, 861, 866
- GridLayout, 857

menu, 866

metadane, 308

metoda

- accept(), 727
- actionPerformed(), 906, 1047
- addDownload(), 1142
- addFirst(), 504
- addImage(), 895
- addLast(), 504
- addMouseListener(), 778
- addMouseMotionListener(), 778
- allocate(), 684
- append(), 429
- arraycopy(), 462
- arriveAndAwaitAdvance(), 937
- asList(), 534
- average(), 371
- bar(), 282
- binarySearch(), 534
- call(), 274, 947
- callme(), 178
- cancel(), 968
- capacity(), 427
- charAt(), 185, 413, 428
- checkID(), 895
- clearDownload(), 1142
- clone(), 464
- close(), 332, 336, 599, 611, 628

- compare(), 527
- compareTo(), 297, 418
- compareToIgnoreCase(), 419
- compile(), 975
- compute(), 957
- concat(), 422
- convolve(), 912
- copyOf(), 535
- copyOfRange(), 535
- counter(), 578
- createImage(), 889
- currentThread(), 263
- currentTimeMillis(), 461
- delete(), 430
- deleteCharAt(), 431
- destroy(), 740, 1068
- dispose(), 873
- download(), 1135
- drawArc(), 805
- drawImage(), 889
- drawLine(), 802
- drawPolygon(), 806
- drawRect(), 803
- drawString(), 826
- empty(), 547
- enableEvents(), 879
- endsWith(), 416
- ensureCapacity(), 427
- equals(), 185, 297, 415, 417, 536
- equalsIgnoreCase(), 415
- exec(), 456
- execute(), 968
- exists(), 622
- exitValue(), 456
- Files.newByteChannel(), 684
- Files.newInputStream(), 693
- fill(), 536
- fillArc(), 805
- fillOval(), 804
- fillPolygon(), 806
- fillRect(), 803
- filterRGB(), 909
- finalize(), 158
- find(), 976
- findInLine(), 610
- findWithinHorizon(), 611
- foo(), 282
- forDigit(), 450
- format(), 585
- freeMemory(), 454
- get(), 281
- getActionCommand(), 1040
- getAdjustable(), 761
- getAdjustmentType(), 845

metoda

getAnnotation(), 311, 315
getAnnotations(), 314, 315
getAscent(), 820
getBytes(), 414
getCause(), 255
getChannel(), 705
getChars(), 414, 428
getCodeBase(), 751
getColumnClass(), 1143
getContentPane(), 1021
getDateInstance(), 991
getDirectionality(), 450
getDocumentBase(), 751
getEchoChar(), 847
GetFieldID(), 350
getFirst(), 504
getFont(), 817
getHeaderFields(), 722
getImage(), 889
getInputStream(), 457, 717
getInsets(), 856
getItem(), 868
getItemCount(), 838, 841
getItemSelectable(), 842
getLabel(), 867
getLast(), 504
getLocalHost(), 714
getName(), 622
getob(), 364
GetObjectClass(), 350
getOuputStream(), 717
getOutputStream(), 457
getParameter(), 748
getParent(), 622
getPriority(), 273
getProperty(), 553
getPropertyDescriptors(), 1010
getRectangular(), 1008
getRGB(), 810
getSelectedItem(), 838, 841
getSelectedText(), 847
getSession(), 1090
getState(), 289, 834
getTimeInstance(), 992
getValueAt(), 1143
getWriter(), 1072
grabPixel(), 901
hashCode(), 450, 549
imageUpdate(), 891
indexOf(), 419
init(), 739
initCause(), 255
insert(), 429
invoke(), 959
invokeAll(), 957
isAbsolute(), 623
isAlive(), 270
isAnnotationPresent(), 317
isCancelled(), 969
isCompletedAbnormally(), 969
isDirectory(), 624
isEditable(), 847
isEnabled(), 867
isFile(), 622
isInfinite(), 440
isNaN(), 440
isSelected(), 1043
itemStateChanged(), 835
join(), 270, 957
keyPressed(), 779
keyTyped(), 780
lastIndexOf(), 419
length(), 185, 427
listFiles(), 626
load(), 555
main(), 56, 174, 939
makeGUI(), 1028
map(), 675
mark(), 637
mkdir(), 626
mkdirs(), 627
mouseClicked(), 782
myMeth(), 313
MyMouseAdapter(), 785
myresume(), 287
newByteChannel(), 684, 686
newDirectoryStream(), 698
newFileSystem(), 682
newOutputStream(), 695
nextDouble(), 237, 605
nextGaussian(), 575
notify(), 278
notifyAll(), 278
notifyObservers(), 576
offerFirst(), 504
offerLast(), 504
onAdvance(), 940, 943
openConnection(), 724
ordinal(), 297, 298
paint(), 342, 739, 885, 1028
parseInt(), 446
Paths.get(), 683
peekFirst(), 504
peekLast(), 504
pollFirst(), 504
pollLast(), 504
pop(), 159, 235

print(), 59
printf(), 187, 599, 647
println(), 59, 264
processItemEvent(), 882
push(), 159, 235
put(), 281
radix(), 611
read(), 327
readLine(), 328
readObject(), 665
readPassword(), 662
regionMatches(), 416
register(), 937
reinitialize(), 969
removeFirst(), 504
removeLast(), 504
repaint(), 743
replace(), 422, 431
replaceAll(), 981
reset(), 637
resume(), 284, 286
reverse(), 430
reverseOrder(), 533
run(), 287, 1135
Runtime.exec(), 457
search(), 547
send(), 729
server(), 1072
service(), 1068
setBounds(), 851
setChanged(), 576
setCharAt(), 428
setColor(), 810
setContentTypes(), 1072
setDefault(), 573
setDefaultCloseOperation(), 1020
setEchoChar(), 847
setEditable(), 847
setEnabled(), 867
SetIntField(), 350
setLayout(), 851
setLength(), 427
setRectangular(), 1008
setState(), 834
setVisible(), 1021
showDocument(), 751
showStatus(), 784
showType(), 364
skip(), 611
sleep(), 264, 577, 950
sort(), 537
split(), 982
start(), 266, 739
startsWith(), 416
stateChanged(), 1138
stop(), 286, 740
store(), 555
substring(), 421, 431
suspend(), 284, 286
System.out.println(), 734
tableSelectionChanged(), 1150
toArray(), 502
toBinaryString(), 447
toCharArray(), 415
toHexString(), 447
toLowerCase(), 424
toOctalString(), 447
toPath(), 696
toString(), 243, 412
totalMemory(), 454
trim(), 422
type(), 459
update(), 740, 1143
updateButtons(), 1151
useDelimiter(), 609
useRadix(), 611
valueOf(), 294, 423
values(), 294
varargs, 187, 191
vaTest(), 188
verifyUrl(), 1150
visitFile(), 703
wait(), 278
waitFor(), 456
walkFileTree(), 701
write(), 325, 330, 336
metody, 142, 148, 150
abstrakcyjne, 213
abstrakcyjne klasy Dictionary, 548
checked, 533
dodatkowe klasy String, 426
dodatkowe klasy StringBuffer, 432
fabryczne klasy InetAddress, 714
finalne, 216
get() i put(), 674
getRed(), getGreen() i getBlue(), 809
graficzne, 802
hasNext, 602
interfejsu AppletContext, 753
interfejsu BasicFileAttributes, 681
interfejsu CharSequence, 483
interfejsu Collection, 492
interfejsu Deque, 498
interfejsu DosFileAttributes, 681
interfejsu FileVisitor, 702
interfejsu HttpServletRequest, 1080
interfejsu HttpServletResponse, 1081
interfejsu HttpSession, 1082

metoda

- interfejsu Iterator, 511
 - interfejsu List, 494
 - interfejsu ListIterator, 512
 - interfejsu Map, 518
 - interfejsu Map.Entry, 521
 - interfejsu NavigableMap, 520
 - interfejsu NavigableSet, 497
 - interfejsu ObjectInput, 667
 - interfejsu ObjectOutput, 665
 - interfejsu Path, 677
 - interfejsu PosixFileAttributes, 682
 - interfejsu Servlet, 1074
 - interfejsu ServletConfig, 1074
 - interfejsu ServletContext, 1075
 - interfejsu ServletRequest, 1076
 - interfejsu SortedMap, 519
 - interfejsu SortedSet, 495
 - klasy Applet, 736
 - klasy BitSet, 561
 - klasy Boolean, 453
 - klasy Buffer, 673
 - klasy Calendar, 566
 - klasy Character, 449
 - klasy Class, 466
 - klasy Color, 809
 - klasy Cookie, 1083
 - klasy Console, 662
 - klasy Currency, 583
 - klasy DatagramPacket, 730
 - klasy DatagramSocket, 729
 - klasy Date, 565
 - klasy Enum, 482
 - klasy EnumSet, 511
 - klasy File, 623
 - klasy Files, 678
 - klasy Font, 814
 - klasy FontMetrics, 819
 - klasy ForkJoinTask, 970
 - klasy Formatter, 585
 - klasy Hashtable, 550
 - klasy HttpServlet, 1084
 - klasy HttpURLConnection, 725
 - klasy InetAddress, 715
 - klasy InputStream, 631
 - klasy Math, 470
 - klasy Object, 217, 463
 - klasy ObjectInputStream, 668
 - klasy ObjectOutputStream, 666
 - klasy Observable, 577
 - klasy OutputStream, 631
 - klasy Package, 480
 - klasy Process, 453
 - klasy ProcessBuilder, 458
 - klasy Properties, 553
 - klasy Random, 575
 - klasy Reader, 652
 - klasy ResourceBundle, 612
 - klasy Runtime, 455
 - klasy Scanner, 603
 - klasy Socket, 717
 - klasy StackTraceElement, 481
 - klasy StringTokenizer, 560
 - klasy System, 460
 - klasy Thread, 263, 474
 - klasy ThreadGroup, 476
 - klasy Timer, 581
 - klasy TimerTask, 580
 - klasy TimeZone, 571
 - klasy URLConnection, 723
 - klasy Vector, 544
 - klasy Writer, 653
 - mostu, 398
 - obiektu Modifier, 986
 - prywatne, 49
 - przetwarzające zdarzenia, 879
 - publiczne, 49
 - rdzenne, 351
 - read() i write(), 675
 - rekurencyjne, 171
 - składowe, 49
 - sparametryzowane, 362
 - statyczne, 177
 - unmodifiable, 533
 - w Javie, 49
 - w klasie
 - Byte, 441
 - Double, 438
 - Float, 437
 - Integer, 442
 - Long, 444
 - Short, 442
 - Stack, 547
 - Throwable, 253
 - zmiennieargumentowe, 187
 - związane z blokadami, 952
 - zwracające strumienie, 717
- MIME, Multipurpose Internet Mail Extensions, 1067
- mniej znaczący surogat, 451
- model, 1014
- model delegowania zdarzeń, 1023
- model zorientowany na procesy, 47
- modyfikator
 - dostępu, 56, 173
 - private, 174
 - protected, 174
 - public, 174
 - native, 348

strictfp, 347
transient, 344
volatile, 344
modyfikowanie łańcucha, 420
monitor, 262, 273
muteks, 273
MVC, Model-View-Controller, 1014
mysz, 776

N

narzędzia AWT, 740, 758, 787, 836, 1012
narzędzia Java EE SDK, 1069
narzędzie javadoc, 1155
nasylenie, 809
native, 348
nawiasy okrągłe, 112
nazwa logiczna, 813
nazwa pliku, 54
nazwa rodzaju, 813
nazwa rodziny, 611, 813
nazwa wątku, 264
nazwa zmiennej, 58
niejednoznaczności, 191
niezaufany kod, 454
NIO, New I/O, 45, 671
NIO.2, 45, 683
notacja naukowa, 75
notacja standardowa, 75
NTSC, National Television Standards Committee, 909

O

obiekt, 141
 ArrayList, 502, 540
 Comparator, 526
 Iterator, 512
 jako parametr, 166
 klasy String, 184
 nasłuchujący, 756, 758
 Properties, 554
 String, 407
 TimeUnit, 949
 typu Process, 456
 typu String, 184
 typu wyliczeniowego, 295
obiektowa reprezentacja typów prostych, 308
obliczanie
 czasu wykonywania programu, 461
 maksymalnej emerytury, 1116
 początkowej wartości inwestycji, 1112
 przyszłej wartości inwestycji, 1104
 raty pożyczki, 1096
 metoda actionPerformed(), 1102
 metoda compute(), 1103
 metoda init(), 1100

 metoda makeGUI(), 1100
 pola apletu, 1099
 wkładu początkowego, 1108
 pozostałej kwoty do spłaty kredytu, 1120
obramowanie, 855
obraz, 887
 ładowanie, 889
 tworzenie, 888
 wyświetlanie, 889
obserwator obrazu, 889
obsługa
 metadanych, 695
 obrazów graficznych, 887
 rysowania, 1032
 wielu wątków, 921
 wyjątków, 258
 catch, 243, 244
 finally, 250
 throw, 247
 throws, 249
 try, 242
 wyświetlenie opisu, 243
 zdarzeń, 756, 796
 zdarzeń list, 841
 zdarzeń list rozwijanych, 839
 zdarzeń menu, 868
 zdarzeń pasków przewijania, 844
 zdarzeń pól tekstowych, 848
 zdarzeń pól wyboru, 834
 zdarzeń przycisków, 831
 znaków Unicode, 452
 żądań GET, 1085
 żądań POST, 1086
ochrona dostępu, 222
odcinek, 802
odczytywanie
 adnotacji, 314
 danych z konsoli, 327
 łańcuchów, 328
 pliku, 333, 684, 704
 znaków, 327
odpowiedź HTTP, 1072, 1085
ograniczenia typów sparametryzowanych, 401
ograniczenie parametru typu V przez T, 382
okna, 790, 792
 obsługa zdarzeń, 796
 tytuł, 793
 ukrywanie, 793
 wymiar, 793
 wyświetlanie, 793
 zamykanie, 793
okno dialogowe, 872
 modalne, 872
 niemodalne, 872
 otwierania pliku, 878

okno najwyższego poziomu, 792
 okrag, 804
 OOP, object-oriented programming, 34
 opakowania typów, 300
 opakowania typów numerycznych, 301
 opakowania typów prostych, 436
 opakowywanie, 302
 operacje atomowe, 954
 operacje wejścia-wyjścia, 619
 operator
 diamentu \diamond , 395
 instanceof, 345, 391
 kropki, 143
 new, 146, 395
 trójargumentowy, 110
 operatory
 arytmetyczne, 94
 arytmetyczne z przypisaniem, 95
 bitowe, 98
 &, 98
 &=, 98
 ^, 98
 ^=, 98
 |, 98
 |=, 98
 ~, 98
 <<, 98
 <<=, 98
 >>, 98
 >>=, 98
 >>>, 98
 >>>=, 98
 bitowe logiczne, 99
 bitowe z przypisaniem, 105
 logiczne, 107
 !, 108
 !=, 108
 &, 108
 &&, 108
 &=, 108
 ?., 108
 ^, 108
 ^=, 108
 |, 108
 ||, 108
 |=, 108
 ==, 108
 logiczne ze skracaniem, 109
 relacji, 106
 !=, 106
 <, 59, 106
 <=, 106
 ==, 59, 106, 417
 >, 59, 106
 >=, 106

otwieranie plików, 680
 APPEND, 680
 CREATE, 680
 CREATE_NEW, 680
 DELETE_ON_CLOSE, 680
 DSYNC, 680
 READ, 680
 SPARSE, 680
 SYNC, 680
 TRUNCATE_EXISTING, 680
 WRITE, 680

P

pakiety, 173, 219
 Concurrency Utilities, 972
 Java EE, 1073
 Java SE, 1073
 java.awt, 1024
 java.awt.event, 759, 1024
 java.awt.image, 887, 903, 919
 java.io, 324, 619
 java.lang, 252, 407, 435
 java.text, 973
 java.util, 487, 541, 580, 611
 java.util.concurrent, 922, 949
 java.util.concurrent.atomic, 923, 954
 java.util.concurrent.locks, 924
 java.util.regex, 973
 javax.imageio, 919
 javax.servlet, 1073
 javax.servlet.http, 1073, 1079
 javax.swing, 1015, 1035
 javax.swing.event, 1023
 javax.swing.tree, 1059
 RMI, 990
 wyrażen regularnych, 973
 zasobów, 611
 pakiety
 biblioteki Swing, 1017
 głównego interfejsu API, 974
 systemu NIO, 671
 pamięć, 454
 panel podzielony na zakładki, 1049
 panel szklany, 1017
 panel treści, 1017
 panel wielowarstwowy, 1017
 para klucz-wartość, 549
 parametr, 56, 148, 152
 parametr typu, 364
 parametr wieloargumentowy, 192
 pasek menu, 827, 866
 pasek przewijania, 843
 pasek stanu, 746

pętla

- do-while, 122
- for, 60, 125
- for-each, 490, 514
- rozszerzona for, 131
- usprawniona for, 129
- while, 121
- zagnieżdżona, 133
- zdarzeń z odpytywaniem, 260

PLAF, pluggable look and feel, 1013

plik

- AddClient.java, 989
- AddCookie.html, 1088
- AddCookieServlet.java, 1088
- AddServer.java, 988
- AddServerImpl.java, 988
- AddServerIntf.java, 988
- ColorGet.html, 1085
- ColorGetServlet.java, 1085
- ColorPost.html, 1086
- ColorPostServlet.java, 1086
- GetCookiesServlet.java, 1088
- jni.h, 350
- NativeDemo.c, 350
- NativeDemo.h, 350
- PostParameters.html, 1077
- PostParametersServlet.java, 1077
- RunApp.html, 343
- servlet-api.jar, 1070
- shutdown.bat, 1070
- startup.bat, 1070
- web.xml, 1070, 1086

pliki

- .class, 144
- cookie, 727, 1082
- dll, 350
- kopiowanie, 692
- odczytywanie, 684, 704
- opcje otwierania, 680
- otwieranie, 693
- zapisywanie, 688

pobieranie plików z internetu, 1130

pobieranie priorytetu wątku, 273

początkowa wartość inwestycji, 1112

podklasa, 51, 193

podklasa sparametryzowana, 390

podpakiet

- java.lang.annotation, 486
- java.lang.instrument, 486
- java.lang.invoke, 486
- java.lang.management, 486
- java.lang.ref, 486
- java.lang.reflect, 486
- java.util.jar, 617
- java.util.logging, 617

java.util.prefs, 617

java.util.regex, 617

java.util.spi, 617

java.util.zip, 617

podpakiety pakietu java.util, 616

podstawowa płaszczyzna wielojęzyczna, 451

podwójne buforowanie, 892

pole tekstowe, 846

pole wyboru, 834, 1045

polecenie package, 220

polimorfizm, 51, 164, 211

pop, 159

porównywanie łańcuchów, 415

port, 711

poziom równoległości, 962

prawy ukośnik (/), 621

priorytet operatorów, 111

priorytet wątku, 261, 272

private, 174

problem producenta i konsumenta, 278

procedura pośrednicząca, 989

proces, 259

progi przetwarzania sekwencyjnego, 971

program AddClient, 991

program CopyFile, 340

program obsługujący interfejs CGI, 1068

program pobierania plików, 1130

klasa Download, 1130

klasa DownloadManager, 1130

klasa DownloadsTableModel, 1130

klasa ProgressRenderer, 1130

kompilacja, 1152

uruchamianie, 1152

program rmregistry, 990

program ShowFile, 335

program SwingDemo, 1019

program TreeMap, 528

program współbieżny, 921

program wykorzystujący okna, 800

programowanie

obiektowe, 34, 47

równoległe, parallel programming, 45, 261, 955

strukturalne, 33

wielowątkowe, 259

współbieżne, 921

z pętlą zdarzeń, 277

projekt Coin., 44

prostokąt, 803

protected, 174

protokół

FTP, 712, 1130

HTTP, 712, 1090, 1130

wersja niewznawialna (HTTP 1.0), 1130

wersja wznawialna (HTTP 1.1), 1130

protokół
 IPv4, 712
 JNLP, 747
 TCP/IP, 712, 728
 Telnet, 712
przechowywanie obiektów, 515
przeciążanie konstruktorów, 164, 357
przeciążanie metod, 161, 163, 190
przecinek, 126
przeglądarka apletów, 734
przekazywanie parametrów, 748
przekazywanie przez referencję, 168
przekazywanie przez wartość, 168
przekazywanie sterowania do wątku, 275
przekształcanie apletu do serwletu, 1124
przełączenie kontekstu, 261
przełącznik, 1042
przenośność, 38, 351
przesłanie, 211
przesłanie metod, 207, 211
przesłanie nazw, 202
przesunięcie w lewo, 101
przesunięcie w prawo, 103
przesunięcie w prawo bez znaku, 104
przetwarzanie, 559
przetwarzanie sekwencyjne, 962
przewijanie komponentu, 1051
przycisk, 830, 1040
 JButton, 1039
 JCheckBox, 1039
 JRadioButton, 1039
 JToggleButton, 1039
przycisk przełącznika, toggle button, 1042
przyciski biblioteki Swing, 1039
przyciski opcji, 1046
przyciski z ikonami, 1042
przykrywanie metod, 394
przykrywanie metody paint(), 885
public, 142, 174
punkt kodowy, 451
push, 159
pusta instrukcja, 122

R

rata pożyczki, 1096
referencja do obiektu, 145
referencja do typu sparametryzowanego, 387
refleksja, reflection, 310, 313, 486, 983
rekurencja, 171, 959
RGB, Red-Green-Blue, 809
RMI, Remote Method Invocation, 42, 663, 987
rozdzielczość czasu, 949
rozgłaszanie zdarzenia, 1002
rozmiary obiektów graficznych, 807

rozpakowywanie, 302
rozszerzanie
 komponentów AWT, 878
 komponentu grupy pól wyboru, 881
 kontrolki Button, 879
 kontrolki Checkbox, 880
 kontrolki Choice, 882
 kontrolki List, 883
 kontrolki Scrollbar, 884
 typu, 84
rozwijanie w miejscu wywołania, 216
rysowanie, 1028
 biblioteka Swing, 1029
 obszar rysowania, 1029
rysowanie wykresu, 1127
rzutowanie, 393
rzutowanie na błędny typ, 540
rzutowanie niezgodnych typów, 82
rzutowanie typów, 81

S

selektor, 676
semafor, 273, 972
separator, 64, 559, 609
serializacja, 663, 666, 1004
serwer InterNIC, 718
serwlet, 39, 1067, 1095
 kod źródłowy, 1088
 serwlet RegPayS, 1124
sesja, 1090
sieć, 711
skaner, 559, 611
składowa
 System.err, 326
 System.in, 326
 System.out, 326
składowe klasy, 49, 142, 196
składowe statyczne, 177
skrót, 505
słowo kluczowe, 64
 abstract, 213
 assert, 351
 catch, 242
 enum, 292
 final, 178, 216, 257
 finally, 250
 interface, 228
 static, 176, 354
 super, 177, 199
 synchronized, 274, 276
 this, 156, 177
 throw, 239, 247
 throws, 249
 try, 239, 242

- sparametryzowana klasa bazowa, 388
- sparametryzowane kolekcje, 539
- specyfikator
 - %%, 591
 - %d, 588
 - %e, 588
 - %f, 588
 - %g, 588
 - %n, 591
 - %s, 593
 - %tM, 589
 - konwersji formatu, 586
 - minimalnej szerokości pola, 591
 - precyzji, 593
- specyfikatory formatów, 586, 587
- specyfikatory formatów pisane wielką literą, 597
- sprawdzanie zgodności typów, 391
- stała
 - klasowa, 312
 - tekstowa, 184
 - własnego typu, 292
 - wyliczeniowa, 292
- stałe, 63
 - całkowitoliczbowe, 74
 - klas Byte, Short, Integer i Double, 441
 - klas Float i Double, 437
 - klasy AdjustmentEvent, 761
 - klasy Calendar, 567
 - klasy Character, 448
 - klasy ComponentEvent, 762
 - klasy ItemEvent, 765
 - klasy Locale, 573
 - klasy MouseEvent, 767
 - klasy MouseWheelEvent, 768
 - klasy WindowEvent, 770
 - logiczne, 76
 - łańcuchowe, 77
 - pola klasy GridBagConstraints, 863
 - znakowe, 76
- stan wątku, 287, 289
- static, 142
- stos, 159, 481
- stos rozszerzający, 234
- stos wywołań, 241
- strategia zachowywania adnotacji, 309
- strona, party, 936
- strumień buforowany, 639
- strumień filtrowany, 639
- strumień znaków, 651
- strumień, 324, 619
 - bajtów, 324
 - błędów, 326
 - fin, 340
 - fout, 340

- katalogu, 698
- wejściowy, 326
- wyjściowy, 326
- znaków, 324
- sufiksy formatów daty i czasu, 590
- suwak, 843
- synchronizacja, 273, 924
- synchronizacja metod, 274
- synchronizacja międzyprocesowa, 262
- synchronizator, 922
- system NIO, 619, 672
 - bufory, 672
 - kanały, 673, 684
 - selektor, 676
 - strumień, 693
 - system plików, 695
 - ścieżki, 695
 - zestaw znaków, 675
- System.in, 327
- System.out, 330
- szablon obiektu, 141
- szeregowanie dostępu do metody, 275
- szkielet apletu, 738

Ś

- ścieżka CLASSPATH, 220
- śledzenie sesji, 1090
- środowisko
 - Eclipse, 1069
 - Java, 462
 - NetBeans, 1069
 - Tomcat, 1069

T

- tablica, 56, 85, 179
 - dynamiczna, 501, 542
 - jednowymiarowa, 85
 - mieszająca, 522, 549
 - newimgpixels, 915
 - tablic, 131
 - wielowymiarowa, 87, 131
- TCP, Transmission Control Protocol, 712
- technologia CGI, 1068
- technologia Java Beans, 973
- tekst
 - wyrównywanie, 822
 - wyśrodkowanie, 821
 - wyświetlanie, 819
- tekst ASCII, 333
- termin ważności *cookie*, 1090
- testowanie apletów, 343
- token, 559, 600
- transient, 344

trwałość, 1003
tryb rysowania, 811
try-with-resources, 256, 338, 599, 607, 629, 686, 707
trzy kropki (...), 188
tworzenie
 apletu, 1026
 czcionek, 815
 drzewa, 1060
 katalogów, 626
 klasy sparametryzowanej, 370
 komponentu JTable, 1063
 metody sparametryzowanej, 380
 obiektu obrazu, 888
 okna typu Frame, 794
 panelu z zakładkami, 1049
 podklasy wyjątków, 252
 serwletów, 1069
 serwletów finansowych, 1123
 wątku, 265, 267
 wielu wątków, 269
typ
 automatyczna konwersja, 81
 automatyczne rozszerzanie, 83
 konwersja rozszerzająca, 81
 konwersja zawężająca, 82
 rzutowanie, 82
 zasady rozszerzania, 84
typ
 boolean, 68
 byte, 67, 69
 Canvas, 792
 char, 67
 double, 68, 71
 float, 68, 71
 int, 67, 69
 long, 67, 69
 typ określony w metodzie, 151
 short, 69
 text/html, 1067
 text/plain, 1067
 Type, 459
typy
 całkowitoliczbowe, 68
 logiczne, 73
 numeryczne, 371
 ograniczone, 370
 opakowujące, wrapper type, 291
 proste, 68
 sparametryzowane, 218, 361, 366, 489, 541
 bezpieczeństwo, 366
 zalety, 367
 surowe, 385
 wyliczeniowe, 291, 299
 zmiennoprzecinkowe, 70
 znakowe, 71
 zwracanych danych, 151

U

UDP, User Datagram Protocol, 712
UI delegate, 1014
układ graficzny komponentów, 850
Unicode, 71, 409, 450
URI, Uniform Resource Identifier, 727
URL, Uniform Resource Locator, 720, 1067
uzupełnianie do 2, 98
uzyskiwanie stanu wątku, 289

V

varargs, 187
volatile, 344

W

wartości domyślne, 315
wartość inwestycji, 1104
wartość kolejności, 297
wartość przekazana jako argument, 148
wątek, 259, 479
 główny, 263
 posiadający monitor, 273
 potomny, 268
 przerwany, 261
 rozdzielający zdarzenia, 1021
w stanie
 działania, 261
 oczekiwania, 261
 zablokowania, 261
 zawieszenia, 261
 wznowiony, 261
wdrażanie serwletu, 1070
widok, viewport, 1014, 1051
widok kolekcji, 489
wielkie litery, 424, 597
wielokąt, 806
wielokrotne łapanie, 257
wielowątkowość, 260, 277, 290, 922
wielozadaniowość, 259
wielozadaniowość z wywłaszczaniem, 261
wirtualne kody klawiszy, 766
wkład początkowy, 1108
właściwość, 1000
właściwość indeksowana, 1001
właściwość ograniczona, 1003
 bound, 1003
 constrained, 1003
właściwość prosta, 1001
wnioskowanie typów, 396
wskaźnik, 92
wskaźnik pliku, 650
współbieżność, 950

wyciek pamięci, memory leaks, 332
wydajność, 962
wydobywanie, 302
wyjątek
 ArithmeticException, 242
 ArrayIndexOutOfBoundsException, 536, 537
 ArrayStoreException, 535, 536
 AssertionError, 351
 ClassCastException, 533, 535, 537
 FileNotFoundException, 332, 628
 HeadlessException, 829
 IllegalArgumentException, 536, 537
 IOException, 336, 628, 635
 NegativeArraySizeException, 535
 NoSuchMethodException, 311
 NullPointerException, 535, 536, 550
 NumberFormatException, 302
 SecurityException, 332, 628
 ServletException, 1077
 UnavailableException, 1077
 UnsupportedOperationException, 533
wyjątki, 239
 final rethrow, 256
 finalne zgłoszenie dalej, 256
 multi-catch, 256
 nieprzechwycone, 241
 nieweryfikowane, 252
 try-with-resources, 256, 338, 599, 629, 686
 typy, 240
 wbudowane, 251
 wielokrotne łapanie, 256
wykładnik binarny, 76
wykradanie zadań, work-stealing, 959
wyliczenia, 292
wymuszanie zgodności typów, 382
wyodrębnianie znaków, 413
wyrażenia regularne, 973, 977, 983
wyrażenie catch, 628
wyrażenie try, 338
wyrównywanie tekstu, 822
wysyłanie jednostkowe, 757
wyszukiwanie znaku, 419
wyścig, 275
wyśrodkowanie tekstu, 821
wyświetlanie, 742, 754
wyświetlanie tekstu, 819
wywołanie super(), 202, 268, 359
wywołanie this(), 357, 359
wywoływanie konstruktorów, 206
wznawianie pobierania, 1129, 1153
wznawianie wątku, 284
wzorce projektowe, 1002
wzorce projektowe dla zdarzeń, 1002
wzorzec, 699, 973, 976, 981

Z

zachowanie oszczędne (leniwe), 980
zachowanie zachłanne, 980
zadania
 anulowanie, 968
 asynchroniczność, 968
 określanie statusu, 969
 ponowne uruchamianie, 969
 wydajność, 962
zakleszczenie, 282, 1022
zamykanie strumieni, 628
zapis do pliku, 336
zapisywanie pliku, 688
zarządzanie pamięcią, 454
zatrzymanie wątku, 286
zawartość katalogu, 698
zawieszanie wątku, 284
zbiór siatek, 861
zdalne wywoływanie metod, 663, 987
zdarzenie, 757
 AdjustmentEvent, 845
 generowane przez klawiaturę, 779
 generowane przez mysz, 776
 PropertyChangeEvent, 1003
zmienna, 57, 77
 czas życia, 79
 deklaracja, 78
 inicjalizacja dynamiczna, 78
 zasięg, 79
zmienna
 FILE_NEW, 179
 FILE_OPEN, 179
 flicker, 894
 globalna, 177
 lokalna, 157
 raw, 387
 referencyjna, 147, 198
 referencyjna interfejsu, 230
 referencyjna obRef, 464
 składowa, 49, 142, 157
 składowa length, 179
 sterująca pętlą, 125
 suspendFlag, 287
 środowiskowa CLASSPATH, 990, 1070
 środowiskowa JAVA_HOME, 1070
zmienna liczba argumentów, 187
zmienne członkowskie, 49
zmienne w interfejsach, 236
znacznik
 #, 596
 @author, 1156
 @deprecated, 1157
 @exception, 1157

znacznik

- @param, 1158
- @return, 1158
- @see, 1158
- @serial, 1159
- @serialData, 1159
- @serialField, 1159
- @since, 1159
- @throws, 1159
- @version, 1160
- {@code}, 1156
- {@docRoot}, 1157
- {@inheritDoc}, 1157
- {@link}, 1157
- {@linkplain}, 1157
- {@literal}, 1158
- {@value}, 1159
- , 887
- APPLET, 735, 747
- CODE, 747
- IMG, 748
- nawiasów, 595
- plusa, 595
- przecinka, 596
- spacji, 595
- zera, 595

znaczniki

- formatów, 594
- komentarzy, 1155
- komentarzy dokumentacyjnych, 1156
- przyłączane, 1155
- samodzielne, 1155

znak

- }, 57
- znak biały, 63
- średnika (;), 57
- zapytania (?), 374
- zastępczy, 979

znaki uzupełniające, supplemental characters, 451

znoszenie, 365, 397

zwracanie, pushback, 642

zwracanie obiektów, 170

Ż

źródła zdarzeń, 757, 771

Ż

żądanie HTTP

- GET, 1068

- POST, 1077, 1086

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



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

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

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

GRUPA WYDAWNICZA

 **Helion SA**

Java

Kompendium programisty

Wydanie VIII

Podobnie jak wcześniejsze popularne języki programowania komputerów, Java to mieszanka najlepszych elementów swoich poprzedników. Jej dodatkową zaletą jest innowacyjna koncepcja samego języka, wynikająca z unikatowej misji. Obecnie Java jest bardzo chętnie wykorzystywana we wszelkich projektach informatycznych. Silne typowanie, jasno określona, przejrzysta składnia oraz ogromna liczba bibliotek rozwiązujących najbardziej wymyślne problemy zjednują jej wielu zwolenników.

Książka, którą trzymasz w ręku, jest unikalną pozycją, gdyż jako jedna z pierwszych omawia nowości z ostatniego wydania języka Java, oznaczonego numerem 7. Znajdziesz tu informacje na temat nowego API do obsługi operacji wejścia-wyjścia, wykorzystania łańcuchów znaków w wyrażeniach `switch` oraz wychwytywania wielu wyjątków w ramach jednego bloku `catch`. Poza nowościami autor omawia konstrukcje obecne w języku od lat. To idealna propozycja dla każdego programisty Javy, który treści zawarte w książce może traktować wybiórczo i sięgać po nią, gdy będzie miał wątpliwości co do sposobu rozwiązania konkretnego problemu. Natomiast dla osób, które chcą poznać język Java i wkroczyć w świat zaawansowanych technologii, będzie ona pasjonującym przewodnikiem.

W trakcie lektury:

- ▶ poznasz nowości w najnowszym wydaniu Javy
- ▶ zaznajomisz się z historią tego języka
- ▶ nauczysz się podstaw składni
- ▶ stworzysz aplikację internetową
- ▶ poznasz mocne i słabe strony Javy



Kompletne źródło informacji na temat języka Java!

helion.pl
księgarnia
internetowa

Nr katalogowy: 8403



Księgarnia internetowa:
<http://helion.pl>



Zamówienia telefoniczne:
0 801 339900



0 601 339900

**Helion**

Sprawdź najnowsze promocje:

① <http://helion.pl/promocje>

Książki najchętniej czytane:

② <http://helion.pl/bestsellery>

Zamów informacje o nowościach:

③ <http://helion.pl/nowosci>**Helion SA**

ul. Kosciuszki 1c, 44-103 Gliwice

tel.: 32 239 98 63

e-mail: helion@helion.pl<http://helion.pl>

ISBN 978-83-246-3767-6



Cena 179,00 zł

Informatyka w najlepszym wydaniu

9 788324 637676