

JAVA[®]

Podstawy

WYDANIE X



CAY S. HORSTMANN

Tytuł oryginału: Core Java Volume I – Fundamentals (10th Edition)

Tłumaczenie: Łukasz Piwko

ISBN: 978-83-283-2480-0

Authorized translation from the English language edition, entitled CORE JAVA VOLUME I – FUNDAMENTALS, Tenth Edition; ISBN 0134177304; by Cay S. Horstmann; published by Pearson Education, Inc, publishing as Prentice Hall.
Copyright © 2016 Oracle and/or its affiliates. All rights reserved.
Portions © Cay S. Horstmann

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 Pearson Education Inc.
Polish language edition published by HELION S.A. Copyright © 2016.

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

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

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

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

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
<ftp://ftp.helion.pl/przyklady/javp10.zip>

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
<http://helion.pl/user/opinie/javp10>
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

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

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

Spis treści

Wstęp	13
Podziękowania	19
1. Wprowadzenie do Javy	21
1.1. Java jako platforma programistyczna	21
1.2. Słowa klucze białej księgi Javy	22
1.2.1. Prostota	23
1.2.2. Obiektowość	23
1.2.3. Sieciowość	24
1.2.4. Niezawodność	24
1.2.5. Bezpieczeństwo	24
1.2.6. Niezależność od architektury	25
1.2.7. Przenośność	26
1.2.8. Interpretacja	26
1.2.9. Wysoka wydajność	27
1.2.10. Wielowątkowość	27
1.2.11. Dynamiczność	27
1.3. Aplety Javy i internet	28
1.4. Krótka historia Javy	29
1.5. Główne nieporozumienia dotyczące Javy	31
2. Środowisko programistyczne Javy	35
2.1. Instalacja oprogramowania Java Development Kit	36
2.1.1. Pobieranie pakietu JDK	36
2.1.2. Instalacja pakietu JDK	38
2.1.3. Instalacja plików źródłowych i dokumentacji	40
2.2. Używanie narzędzi wiersza poleceń	41
2.3. Praca w zintegrowanym środowisku programistycznym	43
2.4. Uruchamianie aplikacji graficznej	46
2.5. Tworzenie i uruchamianie apletów	48

3. Podstawowe elementy języka Java	55
3.1. Prosty program w Javie	56
3.2. Komentarze	59
3.3. Typy danych	60
3.3.1. Typy całkowite	60
3.3.2. Typy zmiennoprzecinkowe	61
3.3.3. Typ char	62
3.3.4. Unicode i typ char	63
3.3.5. Typ boolean	64
3.4. Zmienne	65
3.4.1. Inicjalizacja zmiennych	66
3.4.2. Stałe	67
3.5. Operatory	67
3.5.1. Funkcje i stałe matematyczne	68
3.5.2. Konwersja typów numerycznych	70
3.5.3. Rzutowanie	71
3.5.4. Łączenie przypisania z innymi operatorami	72
3.5.5. Operatory inkrementacji i dekrementacji	72
3.5.6. Operatory relacyjne i logiczne	73
3.5.7. Operatory bitowe	73
3.5.8. Nawiasy i priorytety operatorów	74
3.5.9. Typ wyliczeniowy	75
3.6. Łańcuchy	76
3.6.1. Podłańcuchy	76
3.6.2. Konkatenacja	76
3.6.3. Łańcuchów nie można modyfikować	77
3.6.4. Porównywanie łańcuchów	78
3.6.5. Łańcuchy puste i łańcuchy null	79
3.6.6. Współrzędne kodowe znaków i jednostki kodowe	80
3.6.7. API String	81
3.6.8. Dokumentacja API w internecie	84
3.6.9. Składanie łańcuchów	85
3.7. Wejście i wyjście	88
3.7.1. Odbieranie danych wejściowych	88
3.7.2. Formatowanie danych wyjściowych	91
3.7.3. Zapis i odczyt plików	95
3.8. Sterowanie wykonywaniem programu	97
3.8.1. Zasięg blokowy	97
3.8.2. Instrukcje warunkowe	98
3.8.3. Pętle	100
3.8.4. Pętle o określonej liczbie powtórzeń	104
3.8.5. Wybór wielokierunkowy — instrukcja switch	108
3.8.6. Instrukcje przerywające przepływ sterowania	110
3.9. Wielkie liczby	113
3.10. Tablice	115
3.10.1. Pętla typu for each	116
3.10.2. Inicjowanie tablic i tworzenie tablic anonimowych	117
3.10.3. Kopiowanie tablicy	118
3.10.4. Parametry wiersza poleceń	119
3.10.5. Sortowanie tablicy	120
3.10.6. Tablice wielowymiarowe	123
3.10.7. Tablice postrzępione	126

4. Obiekty i klasy	129
4.1. Wstęp do programowania obiektowego	130
4.1.1. Klasy	131
4.1.2. Obiekty	131
4.1.3. Identyfikacja klas	132
4.1.4. Relacje między klasami	133
4.2. Używanie klas predefiniowanych	134
4.2.1. Obiekty i zmienne obiektów	135
4.2.2. Klasa <code>LocalDate</code>	137
4.2.3. Metody udostępniające i zmieniające wartość elementu	139
4.3. Definiowanie własnych klas	142
4.3.1. Klasa <code>Employee</code>	143
4.3.2. Używanie wielu plików źródłowych	145
4.3.3. Analiza klasy <code>Employee</code>	146
4.3.4. Pierwsze kroki w tworzeniu konstruktorów	147
4.3.5. Parametry jawne i niejawne	148
4.3.6. Korzyści z hermetyzacji	149
4.3.7. Przywileje klasowe	151
4.3.8. Metody prywatne	152
4.3.9. Stałe jako pola klasy	152
4.4. Pola i metody statyczne	153
4.4.1. Pola statyczne	153
4.4.2. Stałe statyczne	154
4.4.3. Metody statyczne	155
4.4.4. Metody fabryczne	156
4.4.5. Metoda <code>main</code>	156
4.5. Parametry metod	159
4.6. Konstruowanie obiektów	165
4.6.1. Przeciążanie	165
4.6.2. Domyślna inicjalizacja pól	166
4.6.3. Konstruktor bezargumentowy	167
4.6.4. Jawna inicjalizacja pól	167
4.6.5. Nazywanie parametrów	168
4.6.6. Wywoływanie innego konstruktora	169
4.6.7. Bloki inicjalizujące	170
4.6.8. Niszczanie obiektów i metoda <code>finalize</code>	174
4.7. Pakiety	174
4.7.1. Importowanie klas	175
4.7.2. Importowanie statyczne	177
4.7.3. Dodawanie klasy do pakietu	177
4.7.4. Zasięg pakietów	180
4.8. Ścieżka klas	181
4.8.1. Ustawianie ścieżki klas	184
4.9. Komentarze dokumentacyjne	184
4.9.1. Wstawianie komentarzy	185
4.9.2. Komentarze do klas	186
4.9.3. Komentarze do metod	186
4.9.4. Komentarze do pól	187
4.9.5. Komentarze ogólne	187
4.9.6. Komentarze do pakietów i ogólne	188
4.9.7. Generowanie dokumentacji	189
4.10. Porady dotyczące projektowania klas	190

5. Dziedziczenie	193
5.1. Klasy, nadklasy i podklasy	194
5.1.1. Definiowanie podklas	194
5.1.2. Przesłanianie metod	195
5.1.3. Konstruktory podklas	197
5.1.4. Hierarchia dziedziczenia	201
5.1.5. Polimorfizm	201
5.1.6. Zasady wywoływania metod	203
5.1.7. Wyłączanie dziedziczenia — klasy i metody finalne	205
5.1.8. Rzutowanie	206
5.1.9. Klasy abstrakcyjne	209
5.1.10. Ograniczanie dostępu	214
5.2. Kosmiczna klasa wszystkich klas — Object	215
5.2.1. Metoda equals	215
5.2.2. Porównywanie a dziedziczenie	217
5.2.3. Metoda hashCode	220
5.2.4. Metoda toString	222
5.3. Generyczne listy tablicowe	228
5.3.1. Dostęp do elementów listy tablicowej	231
5.3.2. Zgodność pomiędzy typowanymi a surowymi listami tablicowymi	234
5.4. Opakowania obiektów i automatyczne pakowanie	235
5.5. Metody ze zmienną liczbą parametrów	238
5.6. Klasy wyliczeniowe	240
5.7. Refleksja	242
5.7.1. Klasa Class	242
5.7.2. Podstawy przechwytywania wyjątków	244
5.7.3. Zastosowanie refleksji w analizie funkcjonalności klasy	246
5.7.4. Refleksja w analizie obiektów w czasie działania programu	251
5.7.5. Zastosowanie refleksji w generycznym kodzie tablicowym	255
5.7.6. Wywoływanie dowolnych metod	258
5.8. Porady projektowe dotyczące dziedziczenia	261
6. Interfejsy, wyrażenia lambda i klasy wewnętrzne	265
6.1. Interfejsy	266
6.1.1. Koncepcja interfejsu	266
6.1.2. Własności interfejsów	272
6.1.3. Interfejsy a klasy abstrakcyjne	273
6.1.4. Metody statyczne	274
6.1.5. Metody domyślne	275
6.1.6. Wybieranie między metodami domyślnymi	276
6.2. Przykłady interfejsów	278
6.2.1. Interfejsy i wywołania zwrotne	278
6.2.2. Interfejs Comparator	281
6.2.3. Klonowanie obiektów	282
6.3. Wyrażenia lambda	288
6.3.1. Po co w ogóle są lambdy	288
6.3.2. Składnia wyrażeń lambda	289
6.3.3. Interfejsy funkcyjne	292
6.3.4. Referencje do metod	293
6.3.5. Referencje do konstruktorów	295
6.3.6. Zakres dostępności zmiennych	295

6.3.7. Przetwarzanie wyrażeń lambda	298
6.3.8. Poszerzenie wiadomości o komparatorach	301
6.4. Klasy wewnętrzne	302
6.4.1. Dostęp do stanu obiektu w klasie wewnętrznej	303
6.4.2. Specjalne reguły składniowe dotyczące klas wewnętrznych	306
6.4.3. Czy klasy wewnętrzne są potrzebne i bezpieczne?	307
6.4.4. Lokalne klasy wewnętrzne	310
6.4.5. Dostęp do zmiennych finalnych z metod zewnętrznych	310
6.4.6. Anonimowe klasy wewnętrzne	313
6.4.7. Statyczne klasy wewnętrzne	316
6.5. Klasy pośredniczące	319
6.5.1. Kiedy używać klas pośredniczących	319
6.5.2. Tworzenie obiektów pośredniczących	320
6.5.3. Właściwości klas pośredniczących	324
7. Wyjątki, asercje i dzienniki	327
7.1. Obsługa błędów	328
7.1.1. Klasyfikacja wyjątków	329
7.1.2. Deklarowanie wyjątków kontrolowanych	331
7.1.3. Zgłaszanie wyjątków	333
7.1.4. Tworzenie klas wyjątków	334
7.2. Przechwytywanie wyjątków	335
7.2.1. Przechwytywanie wyjątku	335
7.2.2. Przechwytywanie wielu typów wyjątków	337
7.2.3. Powtórne generowanie wyjątków i budowanie łańcuchów wyjątków	339
7.2.4. Klauzula finally	340
7.2.5. Instrukcja try z zasobami	343
7.2.6. Analiza danych ze śledzenia stosu	345
7.3. Wskazówki dotyczące stosowania wyjątków	348
7.4. Asercje	350
7.4.1. Koncepcja asercji	351
7.4.2. Włączanie i wyłączanie asercji	352
7.4.3. Zastosowanie asercji do sprawdzania parametrów	352
7.4.4. Zastosowanie asercji do dokumentowania założeń	353
7.5. Dzienniki	355
7.5.1. Podstawy zapisu do dziennika	355
7.5.2. Zaawansowane techniki zapisu do dziennika	356
7.5.3. Zmiana konfiguracji menedżera dzienników	358
7.5.4. Lokalizacja	359
7.5.5. Obiekty typu Handler	360
7.5.6. Filtry	363
7.5.7. Formatery	364
7.5.8. Przepis na dziennik	364
7.6. Wskazówki dotyczące debugowania	372
8. Programowanie generyczne	379
8.1. Dlaczego programowanie generyczne	380
8.1.1. Zalety parametrów typów	380
8.1.2. Dla kogo programowanie generyczne	381
8.2. Definicja prostej klasy generycznej	382
8.3. Metody generyczne	384

8.4. Ograniczenia zmiennych typowych	385
8.5. Kod generyczny a maszyna wirtualna	387
8.5.1. Wymazywanie typów	388
8.5.2. Translacja wyrażeń generycznych	389
8.5.3. Translacja metod generycznych	389
8.5.4. Używanie starego kodu	392
8.6. Ograniczenia i braki	393
8.6.1. Nie można podawać typów prostych jako parametrów typowych	393
8.6.2. Sprawdzanie typów w czasie działania programu jest możliwe tylko dla typów surowych	393
8.6.3. Nie można tworzyć tablic typów generycznych	394
8.6.4. Ostrzeżenia dotyczące zmiennej liczby argumentów	394
8.6.5. Nie wolno tworzyć egzemplarzy zmiennych typowych	395
8.6.6. Nie można utworzyć egzemplarza generycznej tablicy	396
8.6.7. Zmiennych typowych nie można używać w statycznych kontekstach klas generycznych	398
8.6.8. Obiektów klasy generycznej nie można generować ani przechwytywać	398
8.6.9. Można wyłączyć sprawdzanie wyjątków kontrolowanych	399
8.6.10. Uważaj na konflikty, które mogą powstać po wymazaniu typów	401
8.7. Zasady dziedziczenia dla typów generycznych	402
8.8. Typy wieloznaczne	404
8.8.1. Koncepcja typu wieloznacznego	404
8.8.2. Ograniczenia nadtypów typów wieloznacznych	405
8.8.3. Typy wieloznaczne bez ograniczeń	408
8.8.4. Chwytywanie typu wieloznacznego	408
8.9. Refleksja a typy generyczne	411
8.9.1. Generyczna klasa Class	411
8.9.2. Zastosowanie parametrów Class<T> do dopasowywania typów	412
8.9.3. Informacje o typach generycznych w maszynie wirtualnej	412
9. Kolekcje	419
9.1. Architektura kolekcji Javy	419
9.1.1. Oddzielenie warstwy interfejsów od warstwy klas konkretnych	420
9.1.2. Interfejs Collection	422
9.1.3. Iteratory	423
9.1.4. Generyczne metody użytkowe	425
9.1.5. Interfejsy w systemie kolekcji Javy	428
9.2. Konkretnie klasy kolekcyjne	430
9.2.1. Listy powiązane	431
9.2.2. Listy tablicowe	440
9.2.3. Zbiór HashSet	440
9.2.4. Zbiór TreeSet	444
9.2.5. Kolejki Queue i Deque	448
9.2.6. Kolejki priorytetowe	450
9.3. Słowniki	451
9.3.1. Podstawowe operacje słownikowe	451
9.3.2. Modyfikowanie wpisów w słowniku	454
9.3.3. Widoki słowników	456
9.3.4. Klasa WeakHashMap	457
9.3.5. Klasy LinkedHashSet i LinkedHashMap	458
9.3.6. Klasy EnumSet i EnumMap	459
9.3.7. Klasa IdentityHashMap	460

9.4. Widoki i opakowania	462
9.4.1. Lekkie obiekty opakowujące kolekcje	462
9.4.2. Przedziały	463
9.4.3. Widoki niemodyfikowalne	464
9.4.4. Widoki synchronizowane	465
9.4.5. Widoki kontrolowane	465
9.4.6. Uwagi dotyczące operacji opcjonalnych	466
9.5. Algorytmy	469
9.5.1. Sortowanie i tasowanie	470
9.5.2. Wyszukiwanie binarne	473
9.5.3. Proste algorytmy	474
9.5.4. Operacje zbiorowe	476
9.5.5. Konwersja pomiędzy kolekcjami a tablicami	477
9.5.6. Pisanie własnych algorytmów	478
9.6. Stare kolekcje	479
9.6.1. Klasa Hashtable	479
9.6.2. Wyliczenia	480
9.6.3. Słowniki własności	481
9.6.4. Stosy	482
9.6.5. Zbiory bitów	482
10. Grafika	487
10.1. Wprowadzenie do pakietu Swing	488
10.2. Tworzenie ramki	492
10.3. Pozycjonowanie ramki	494
10.3.1. Własności ramek	496
10.3.2. Określanie rozmiaru ramki	497
10.4. Wyświetlanie informacji w komponencie	500
10.5. Figury 2D	505
10.6. Kolory	513
10.7. Czcionki	517
10.8. Wyświetlanie obrazów	525
11. Obsługa zdarzeń	529
11.1. Podstawy obsługi zdarzeń	529
11.1.1. Przykład — obsługa kliknięcia przycisku	531
11.1.2. Zwięzłe definiowanie procedur nasłuchowych	536
11.1.3. Przykład — zmiana stylu	538
11.1.4. Klasy adaptacyjne	542
11.2. Akcje	546
11.3. Zdarzenia generowane przez mysz	553
11.4. Hierarchia zdarzeń w bibliotece AWT	560
11.4.1. Zdarzenia semantyczne i niskiego poziomu	561
12. Komponenty Swing interfejsu użytkownika	565
12.1. Swing a wzorzec projektowy Model-View-Controller	566
12.1.1. Wzorce projektowe	566
12.1.2. Wzorzec Model-View-Controller	568
12.1.3. Analiza MVC przycisków Swing	571
12.2. Wprowadzenie do zarządzania rozkładem	572
12.2.1. Rozkład brzegowy	574
12.2.2. Rozkład siatkowy	576

12.3. Wprowadzanie tekstu	580
12.3.1. Pola tekstowe	580
12.3.2. Etykiety komponentów	582
12.3.3. Pola haseł	584
12.3.4. Obszary tekstowe	584
12.3.5. Panele przewijane	585
12.4. Komponenty umożliwiające wybór opcji	587
12.4.1. Pola wyboru	587
12.4.2. Przełączniki	590
12.4.3. Obramowanie	593
12.4.4. Listy rozwijalne	597
12.4.5. Suwaki	600
12.5. Menu	606
12.5.1. Tworzenie menu	606
12.5.2. Ikony w elementach menu	609
12.5.3. Pola wyboru i przełączniki jako elementy menu	610
12.5.4. Menu podręczne	611
12.5.5. Mnemoniki i akceleratory	612
12.5.6. Aktywowanie i dezaktywowanie elementów menu	614
12.5.7. Paski narzędzi	618
12.5.8. Dymki	620
12.6. Zaawansowane techniki zarządzania rozkładem	623
12.6.1. Rozkład GridBagLayout	624
12.6.2. Rozkład grupowy	634
12.6.3. Rezygnacja z zarządców rozkładu	643
12.6.4. Niestandardowi zarządcy rozkładu	643
12.6.5. Kolejka dostępu	647
12.7. Okna dialogowe	648
12.7.1. Okna dialogowe opcji	649
12.7.2. Tworzenie okien dialogowych	659
12.7.3. Wymiana danych	663
12.7.4. Okna dialogowe wyboru plików	669
12.7.5. Okna dialogowe wyboru kolorów	679
12.8. Rozwiązywanie problemów z programami z graficznym interfejsem użytkownika	684
12.8.1. Wskazówki dotyczące debugowania	684
12.8.2. Zaprzęganie robota AWT do pracy	686

13. Przygotowywanie apletów i aplikacji do użytku 693

13.1. Pliki JAR	694
13.1.1. Tworzenie plików JAR	694
13.1.2. Manifest	695
13.1.3. Wykonywalne pliki JAR	696
13.1.4. Zasoby	697
13.1.5. Pieczętowanie pakietów	700
13.2. Zapisywanie preferencji użytkownika	701
13.2.1. Słowniki własności	701
13.2.2. API Preferences	706
13.3. Moduły ładowania usług	712
13.4. Aplety	714
13.4.1. Prosty aplet	715
13.4.2. Znacznik applet i jego atrybuty	718

13.4.3. Parametry przekazujące informacje do apletów	720
13.4.4. Dostęp do obrazów i plików audio	725
13.4.5. Środowisko działania apletu	726
13.4.6. Komunikacja pomiędzy apletami	727
13.4.7. Wyświetlanie elementów w przeglądarce	727
13.4.8. Piaskownica	729
13.4.9. Podpisywanie kodu	730
13.5. Java Web Start	732
13.5.1. Wdrażanie aplikacji Java Web Start	732
13.5.2. API JNLP	735
14. Współbieżność	745
14.1. Czym są wątki	746
14.1.1. Wykonywanie zadań w osobnych wątkach	751
14.2. Przerywanie wątków	755
14.3. Stany wątków	758
14.3.1. Wątki NEW	758
14.3.2. Wątki RUNNABLE	758
14.3.3. Wątki BLOCKED i WAITING	759
14.3.4. Zamykanie wątków	759
14.4. Własności wątków	761
14.4.1. Priorytety wątków	761
14.4.2. Wątki demony	762
14.4.3. Procedury obsługi nieprzechwyconych wyjątków	762
14.5. Synchronizacja	764
14.5.1. Przykład sytuacji powodującej wyścig	764
14.5.2. Wyścigi	768
14.5.3. Obiekty klasy Lock	769
14.5.4. Warunki	772
14.5.5. Słowo kluczowe synchronized	777
14.5.6. Bloki synchronizowane	781
14.5.7. Monitor	783
14.5.8. Pola ulotne	783
14.5.9. Zmienne finalne	785
14.5.10. Zmienne atomowe	785
14.5.11. Zakleszczenia	787
14.5.12. Zmienne lokalne wątków	790
14.5.13. Testowanie blokad i odmierzanie czasu	791
14.5.14. Blokady odczytu-zapisu	793
14.5.15. Dlaczego metody stop i suspend są wycofywane	794
14.6. Kolejki blokujące	795
14.7. Kolekcje bezpieczne wątkowo	802
14.7.1. Szybkie słowniki, zbiory i kolejki	802
14.7.2. Atomowe modyfikowanie elementów słowników	804
14.7.3. Operacje masowe na współbieżnych słownikach skrótów	806
14.7.4. Współbieżne widoki zbiorów	808
14.7.5. Tablice kopiowane przy zapisie	808
14.7.6. Równoległe algorytmy tablicowe	808
14.7.7. Starsze kolekcje bezpieczne wątkowo	809
14.8. Interfejsy Callable i Future	810

14.9. Klasa Executors	815
14.9.1. Pule wątków	815
14.9.2. Planowanie wykonywania	820
14.9.3. Kontrolowanie grup zadań	821
14.9.4. Szkielet rozgałęzienie-złączenie	822
14.9.5. Klasa CompletableFuture	824
14.10. Synchronizatory	827
14.10.1. Semaforey	827
14.10.2. Klasa CountdownLatch	828
14.10.3. Bariery	829
14.10.4. Klasa Exchanger	830
14.10.5. Kolejki synchroniczne	830
14.11. Wątki a biblioteka Swing	830
14.11.1. Uruchamianie czasochłonnych zadań	831
14.11.2. Klasa SwingWorker	835
14.11.3. Zasada jednego wątku	841
A. Słowa kluczowe Javy	843
Skorowidz	845

13

Przygotowywanie apletów i aplikacji do użytku

W tym rozdziale:

- 13.1. Pliki JAR
- 13.2. Zapisywanie preferencji użytkownika
- 13.3. Moduły ładowania usług
- 13.4. Aplety
- 13.5. Java Web Start

W tej chwili powinniśmy swobodnie posługiwać się większością funkcji języka Java. Mamy też solidne podstawy programowania interfejsów graficznych. Skoro potrafimy tworzyć aplikacje użytkowe, musimy poznać techniki przygotowywania ich do użytku na komputerze użytkownika. W kwestii tej wybór często pada na **aplety** (ang. *applet*), które były powodem ogromnego zainteresowania Javą na początku jej istnienia. Aplet to specjalny rodzaj programu w Javie, który może zostać pobrany przez przeglądarkę z internetu i uruchomiony. Miał on uwolnić użytkowników od problemów związanych z instalacją oprogramowania, które byłoby pobierane na dowolne urządzenie lub komputer obsługujący Javę i podłączony do internetu.

Aplety nie spełniły pokładanych w nich oczekiwań z wielu powodów. Dlatego rozdział ten zaczynamy od technik pakowania aplikacji. Następnie przedstawiamy sposoby zapisywania przez aplikacje informacji konfiguracyjnych i preferencji użytkownika. Dodatkowo opisujemy techniki wczytywania wtyczek do aplikacji za pomocą klasy `ServiceLoader`.

Później podpowiadamy, co trzeba wiedzieć na temat apletów, na wypadek gdyby ktoś musiał jeden utworzyć albo obsługiwać. Opisujemy też mechanizm **Java Web Start**, będący alternatywnym rozwiązaniem w zakresie dostarczania aplikacji za pomocą internetu, który pod wieloma względami przypomina aplety, tylko lepiej nadaje się do wdrażania aplikacji działających poza stroną internetową.

13.1. Pliki JAR

Pakowanie aplikacji ma na celu utworzenie jednego pliku do wykorzystania przez użytkownika zamiast całej struktury katalogów pełnych plików klas. Do tego celu służą poddawane kompresji ZIP pliki Java Archive (JAR). Mogą one zawierać nie tylko pliki klas, ale również obrazy i pliki dźwiękowe.



W Javie dostępny jest też alternatywny algorytm kompresji o nazwie pack200, który został zoptymalizowany pod kątem bardziej efektywnego zmniejszania rozmiarów plików klas w porównaniu do zwykłego algorytmu ZIP. Według zapewnień firmy Oracle współczynnik kompresji plików klas sięga aż 90%. Więcej informacji na ten temat znajduje się pod adresem <http://docs.oracle.com/javase/1.5.0/docs/guide/deployment/deployment-guide/pack200.html>.

13.1.1. Tworzenie plików JAR

Do tworzenia plików JAR służy narzędzie o nazwie `jar` (w standardowej instalacji JDK znajduje się w katalogu `jdk/bin`). Najczęściej stosowane polecenie tworzące plik JAR ma następującą składnię:

```
jar cvf JARNazwaPliku Plik1 Plik2 . . .
```

Na przykład:

```
jar cvf CalculatorClasses.jar *.class icon.gif
```

OGólny format polecenia `jar` jest następujący:

```
jar opcje Plik1 Plik2 . . .
```

Tabela 13.1 przedstawia wszystkie opcje narzędzia `jar`. Są one podobne do opcji polecenia `tar` w systemie Unix.

Tabela 13.1. Opcje narzędzia `jar`

Opcja	Opis
c	Tworzy puste archiwum i dodaje do niego pliki. Katalogi są przetwarzane rekurencyjnie.
C	Zmienia tymczasowo lokalizację. Na przykład polecenie <code>cvf JARFileName.jar -C classes *.class</code> przechodzi do katalogu <code>classes</code> w celu dodania klas.
e	Tworzy punkt startowy w manifeście (zobacz podrozdział 13.1.3, „Wykonywalne pliki JAR”).
f	Określa plik JAR o danej nazwie jako drugi argument wiersza poleceń. Jeśli tego parametru brakuje, <code>jar</code> wyśle wynik do standardowego wyjścia (przy tworzeniu pliku JAR) lub wczyta go ze standardowego wejścia (przy rozpakowywaniu lub tabulacji pliku JAR).
i	Tworzy plik indeksowy (przyspieszający wyszukiwanie w dużych archiwach).
m	Dodaje manifest do pliku JAR. Manifest jest opisem zawartości i pochodzenia pliku archiwum. Każde archiwum ma domyślny manifest, ale można utworzyć własny, który uwierzytelnia zawartość archiwum.

Tabela 13.1. Opcje narzędzia jar (ciąg dalszy)

Opcja	Opis
M	Blokuje tworzenie domyślnego pliku manifestu.
t	Wyświetla spis treści.
u	Aktualizuje istniejący plik JAR.
v	Generuje obszerne dane wyjściowe.
x	Wypakowuje pliki. Jeśli podanych zostanie kilka nazw plików, zostaną wypakowane tylko one. W przeciwnym przypadku program wypakuje wszystkie pliki.
0	Wyłącza kompresję ZIP.

W plikach JAR można pakować aplikacje, komponenty programów (tak zwane *beans*, o których mowa w rozdziale 11. drugiego tomu) i biblioteki kodu. Na przykład biblioteka wykonawcza JDK jest zawarta w bardzo dużym pliku o nazwie *rt.jar*.

13.1.2. Manifest

Poza klasami, obrazami i innymi plikami źródłowymi każdy plik JAR zawiera plik **manifestu**, który określa specjalne własności archiwum.

Wspomniany plik manifestu ma nazwę *MANIFEST.MF*, a jego lokalizacja to specjalny podkatalog *META-INF* w pliku JAR. Minimalna zawartość takiego pliku nie jest zbyt interesująca:

```
Manifest-Version: 1.0
```

Złożone pliki tego typu mogą zawierać znacznie więcej wpisów pogrupowanych w sekcjach. Pierwsza sekcja nosi nazwę **sekcji głównej** (ang. *main section*) i ma zastosowanie do całego pliku JAR. Kolejne sekcje określają własności różnych elementów mających nazwy, jak konkretne pliki, pakiety czy adresy URL. Każda z nich musi się zaczynać od słowa *Name*. Sekcje są rozdzielane pustą linią. Na przykład:

```
Manifest-Version: 1.0
opis całego archiwum
```

```
Name: Woozle.class
opis jednego pliku
```

```
Name: com/mycompany/mypkg/
opis pakietu
```

Aby zmienić zawartość pliku manifestu, należy dokonać niezbędnych zmian i wydać poniższe polecenie:

```
jar cfm NazwaPlikuJAR NazwaPlikuManifest . . .
```

Na przykład poniższe polecenie tworzy nowy plik JAR z plikiem manifestu:

```
jar cfm MyArchive.jar manifest.mf com/mycompany/mypkg/*.class
```

Aby zaktualizować plik manifestu istniejącego pliku JAR, należy umieścić w pliku tekstowym wpisy, które mają być dodane, i wydać następujące polecenie:

```
jar ufm MyArchive.jar manifest-additions.mf
```



Więcej informacji na temat plików JAR i manifestu można znaleźć na stronie <http://docs.oracle.com/javase/8/docs/technotes/guides/jar>.

13.1.3. Wykonywalne pliki JAR

Istnieje możliwość określenia **punktu startowego** programu, czyli klasy, od której zaczyna się działanie programu, za pomocą opcji `e` narzędzia `jar`:

```
jar cvfe MyProgram.jar com.mycompany.mypkg.MainAppClass pliki do dodania
```

Klasę główną programu można też określić w manifestcie. W tym celu należy do niego dodać instrukcję o następującej postaci:

```
Main-Class: com.mycompany.mypkg.MainAppClass
```

Nie dodawaj rozszerzenia `.class` do nazwy klasy głównej.



Na końcu ostatniego wiersza w pliku manifestu musi się znajdować znak nowego wiersza. W przeciwnym przypadku plik zostanie odczytany nieprawidłowo. Błąd polegający na utworzeniu pliku tekstowego zawierającego tylko wiersz `Main-Class` bez znaku końca wiersza jest często spotykany.

W obu wymienionych przypadkach program można uruchomić przy użyciu następującego polecenia:

```
java -jar MyProgram.jar
```

W zależności od konfiguracji systemu operacyjnego może być możliwe uruchomienie aplikacji za pomocą dwukrotnego kliknięcia pliku JAR. Poniżej znajduje się opis zachowania różnych systemów w takiej sytuacji:

- W systemie Windows instalator aplikacji Java tworzy dowiązanie dla plików o rozszerzeniu `.jar`, które uruchamia te pliki za pomocą polecenia `javaw -jar` (polecenie `javaw`, w przeciwieństwie do `java`, nie otwiera okna wiersza poleceń).
- System Solaris rozpoznaje „magiczną liczbę” pliku JAR i uruchamia ją za pomocą polecenia `java -jar`.
- System Mac OS X rozpoznaje rozszerzenie `.jar` i uruchamia programy w Javie w wyniku dwukrotnego kliknięcia pliku JAR.

Jednak programy Javy w plikach JAR to nie to samo co aplikacje rodzime. W systemie Windows można skorzystać z narzędzi innych producentów służących do zamieniania plików JAR na pliki wykonywalne tego systemu. Plik JAR jest opakowywany w plik o rozszerzeniu `.exe`, który lokalizuje i uruchamia maszynę wirtualną Javy (JVM) lub informuje użytkownika, co powinien zrobić, jeśli JVM nie ma. Istnieje kilka komercyjnych i darmowych narzędzi tego typu, np.: `Launch4J` (<http://launch4j.sourceforge.net>) i `IzPack` (<http://izpack.org>).

W komputerach z systemem Mac OS X sytuacja wygląda nieco lepiej. W skład środowiska programistycznego XCode wchodzi narzędzie o nazwie Jar Bundler służące do zamieniania plików JAR na aplikacje Mac.

13.1.4. Zasoby

Klasy używane zarówno w apletach, jak i aplikacjach często wykorzystują pliki danych tego samego typu:

- pliki obrazów i zawierające dźwięk;
- pliki tekstowe zawierające łańcuchy komunikatów i etykiety przycisków;
- pliki z danymi binarnymi, na przykład opisującymi rozkład mapy.

W Javie takie pliki nazywane są **zasobami** (ang. *resources*).



W systemie Windows termin „zasób” (ang. *resource*) ma węższe znaczenie. Zasoby w tym systemie także mogą być plikami obrazów, etykietami przycisków itd., ale są związane z plikami wykonywalnymi z dostępem za pośrednictwem standardowego interfejsu programistycznego. Natomiast pliki zasobów Javy są przechowywane osobno, a nie jako części plików klas. Dostęp do zasobów i ich interpretacja zależy od programu.

Weźmy na przykład klasę o nazwie `AboutPanel`, która wyświetla komunikat widoczny na rysunku 13.1.

Rysunek 13.1.
Wyświetlanie
zasobu
z pliku JAR



Wiadomo, że tytuł i rok wydania zostaną zmienione w kolejnym wydaniu książki. Aby ułatwić zmianę, ten tekst należy umieścić w pliku tekstowym, a nie bezpośrednio w kodzie programu.

Powstaje jednak pytanie, gdzie umieścić taki plik jak *about.txt*. Oczywiście najlepiej byłoby, aby znajdował się on razem z pozostałymi plikami programu w pliku JAR.

Program ładujący klasy potrafi znaleźć pliki klas, jeśli znajdują się gdzieś na ścieżce klas, w archiwum lub na serwerze sieciowym. Mechanizm zasobów oferuje podobną funkcjonalność dla plików, które nie są klasami. Poniżej znajduje się spis wymaganych czynności:

1. Utwórz obiekt `Class` klasy, która ma zasób, na przykład *AboutPanel.class*.
2. Jeśli zasobem jest obraz lub plik audio, wywołaj metodę `getResource(filename)` w celu uzyskania lokalizacji zasobu w postaci adresu URL. Następnie odczytaj go za pomocą metody `getImage` lub `getAudioClip`.
3. W przypadku innych zasobów niż obrazy i pliki audio dane z pliku należy wczytywać za pomocą metody `getResourceAsStream`.

Chodzi o to, aby program ładujący klasy potrafił znaleźć klasę i odszukać związane z nią zasoby w tej samej lokalizacji.

Na przykład poniższy fragment kodu tworzy ikonę z pliku *about.gif*:

```
URL url = ResourceTest.class.getResource("about.gif");
Image img = new ImageIcon(url).getImage();
```

Powyższy kod można odczytać następująco: „znajdź plik *about.gif* w tej samej lokalizacji, w której znajduje się klasa *ResourceTest*”.

Poniższe instrukcje wczytują plik *about.txt*:

```
InputStream stream = ResourceTest.class.getResourceAsStream("about.txt");
Scanner in = new Scanner(stream, "UTF-8");
```

Plik zasobu nie musi się znajdować w tym samym katalogu co klasa — może być w jakimś podkatalogu. Można zastosować hierarchiczną nazwę zasobu, jak poniższa:

```
data/text/about.txt
```

Jest to względna nazwa zasobu. Jest ona interpretowana względem pakietu klasy, która ładuje dany zasób. Należy pamiętać, że zawsze trzeba używać separatora `/`, bez względu na separator katalogów stosowany w systemie, w którym są przechowywane pliki zasobów. Na przykład w systemie plików systemu Windows separatory `\` są automatycznie zamieniane na `/`.

Nazwa zasobu zaczynająca się od znaku `/` jest bezwzględną nazwą zasobu. Jest ona lokalizowana w taki sam sposób jak klasa wewnątrz pakietu. Na przykład zasób:

```
/corejava/title.txt
```

znajduje się w katalogu *corejava* (który może być podkatalogiem ścieżki klas wewnątrz pliku JAR lub, w przypadku apletów, na serwerze sieciowym).

Jedynym przeznaczeniem funkcji ładowania zasobów jest ładowanie plików. Nie istnieją żadne standardowe metody interpretujące zawartość pliku zasobów. Każdy program musi interpretować zawartość swoich plików zasobów na swój własny sposób.

Innym często spotykanym zastosowaniem zasobów jest międzynarodowa lokalizacja programów. W plikach zasobów przechowuje się łańcuchy, które zmieniają się w zależności od języka, czyli komunikaty i etykiety interfejsu użytkownika. Dla każdego języka tworzony jest osobny plik. **API internacjonalizacji**, które zostało opisane w rozdziale 5. drugiego tomu, udostępnia standardową metodę służącą do organizacji i dostępu do plików lokalizacyjnych.

Listing 13.1 przedstawia kod programu demonstrującego ładowanie zasobów. Poniższe polecenia kompilują go, tworzą plik JAR i uruchamiają go:

```
javac resource/ResourceTest.java
jar cvfm ResourceTest.jar resource/ResourceTest.mf resource/*.class resource/*.gif
↳ resource/*.txt
java -jar ResourceTest.jar
```

Listing 13.1. resource/ResourceTest.java

```
package resource;

import java.awt.*;
import java.io.*;
import java.net.*;
import java.util.*;
import javax.swing.*;

/**
 * @version 1.41 2015-06-12
 * @author Cay Horstmann
 */
public class ResourceTest
{
    public static void main(String[] args)
    {
        EventQueue.invokeLater(() -> {
            JFrame frame = new ResourceTestFrame();
            frame.setTitle("ResourceTest");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setVisible(true);
        });
    }
}

/**
 * Ramka ładująca zasoby graficzne i tekstowe
 */
class ResourceTestFrame extends JFrame
{
    private static final int DEFAULT_WIDTH = 300;
    private static final int DEFAULT_HEIGHT = 300;

    public ResourceTestFrame()
    {
        setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
        URL aboutURL = getClass().getResource("about.gif");
        Image img = new ImageIcon(aboutURL).getImage();
        setIconImage(img);

        JTextArea textArea = new JTextArea();
        InputStream stream = getClass().getResourceAsStream("about.txt");
        try (Scanner in = new Scanner(stream, "UTF-8"))
        {
            while (in.hasNext())
                textArea.append(in.nextLine() + "\n");
        }
    }
}
```

```
    }  
    add(textArea);  
  }  
}
```

Aby przekonać się, że program pobiera pliki zasobów z archiwum JAR, a nie bieżącego katalogu, można przenieść ten program do innego folderu.

`java.lang.Class` **1.0**

- `URL getResource(String name)` **1.1**
- `InputStream getResourceAsStream(String name)` **1.1**

Znajduje zasób w tym samym katalogu, w którym jest umieszczona klasa, i zwraca adres URL lub strumień wejściowy, za pomocą którego można ten zasób załadować. Zwraca wartość `null`, jeśli zasób nie istnieje, dzięki czemu nie powoduje wyjątku dla błędu wejścia-wyjścia.

13.1.5. Pieczętowanie pakietów

W rozdziale 4. wspomnieliśmy o możliwości **pieczętowania** (ang. *seal*) pakietów Javy w celu uniemożliwienia dodawania do nich kolejnych klas. Może być to konieczne w przypadku używania klas, metod i pól o zasięgu pakietowym. Gdyby nie pieczętowanie, inne klasy mogłyby być umieszczane w tym samym pakiecie i dzięki temu uzyskiwać dostęp do elementów pakietowych.

Jeśli na przykład pakiet `com.mycompany.util` zostanie zapieczętowany, żadna klasa spoza tego zapieczętowanego archiwum nie może być zdefiniowana za pomocą poniższej instrukcji:

```
package com.mycompany.util;
```

W tym celu wszystkie klasy pakietu należy umieścić w pliku JAR. Domyślnie pakiety w pliku JAR nie są zapieczętowane. Można zmienić to domyślne globalne ustawienie, wstawiając wiersz

```
Sealed: true
```

w głównej sekcji pliku manifestu. Aby zapieczętować tylko wybrane pakiety, należy do pliku manifestu w pliku JAR wstawić dodatkowe sekcje:

```
Name: com/mycompany/util/  
Sealed: true
```

```
Name: com/mycompany/misc/  
Sealed: false
```

Aby zapieczętować pakiet, należy utworzyć plik tekstowy z instrukcjami manifestu. Następnie należy uruchomić narzędzie `jar` w zwykły sposób:

```
jar cvfm MyArchive.jar manifest.mf pliki do dodania
```

13.2. Zapisywanie preferencji użytkownika

Użytkownicy oczekują, że wszystkie dokonane przez nich ustawienia zostaną zapisane i zastosowane przy ponownym uruchamianiu aplikacji. Najpierw zajmiemy się prostą techniką opartą na zapisywaniu informacji konfiguracyjnych w plikach własności, które były kiedyś stosowane w Javie. Następnie przejdziemy do opisu niezwykle funkcjonalnego API zarządzania preferencjami.

13.2.1. Słowniki własności

Słowniki własności (ang. *property map*) to struktury danych przechowujące pary klucz – wartość, które często znajdują zastosowanie jako przechowalnice danych konfiguracyjnych aplikacji. Każdy taki słownik ma trzy cechy:

- Klucze i wartości są łańcuchami.
- Można ją łatwo zapisać w pliku i załadować z niego.
- Istnieje druga tabela przechowująca wartości domyślne.

Klasa odpowiedzialna za implementację słowników własności nosi nazwę `Properties`.

Jak wiadomo, słowniki własności znajdują zastosowanie w określaniu opcji konfiguracyjnych programów. Na przykład:

```
Properties settings = new Properties();
settings.put("width", "200");
settings.put("title", "Witaj, świecie!");
```

Do zapisania takiej listy własności w pliku należy użyć metody `store`. My zapiszemy nasz słownik w pliku o nazwie *program.properties*. Drugi argument metody `store` to komentarz, który jest dodawany do pliku.

```
FileOutputStream out = new FileOutputStream("program.properties");
settings.store(out, "Ustawienia programu");
```

W pliku zostaną zapisane następujące dane:

```
#Ustawienia programu
#Mon Apr 30 07:22:52 2007
width=200
title=Witaj, świecie!
```

Do ładowania plików ustawień służą następujące instrukcje:

```
FileInputStream in = new FileInputStream("program.properties");
settings.load(in);
```

Istnieje zwyczaj przechowywania ustawień programu w podkatalogu głównego katalogu użytkownika. Nazwa tego katalogu zazwyczaj zaczyna się od kropki — w systemie Unix konwencja taka oznacza, że katalog jest katalogiem systemowym ukrytym przed użytkownikiem. W naszym przykładowym programie stosujemy się do tej konwencji.

Do sprawdzenia katalogu głównego użytkownika można wykorzystać metodę `System.getProperties`, która — tak się składa — wykorzystuje obiekt typu `Properties` do zapisu danych systemowych. Katalog główny ma klucz `user.home`. Istnieje także metoda pozwalająca odczytać pojedynczy klucz:

```
String userDir = System.getProperty("user.home");
```

Dobrze jest na wszelki wypadek dostarczyć zestaw ustawień domyślnych dla programu. Klasa `Properties` dysponuje dwoma mechanizmami pozwalającymi określić ustawienia domyślne. Po pierwsze, można utworzyć łańcuch, który będzie stosowany domyślnie za każdym razem, kiedy dany klucz nie zostanie znaleziony.

```
String title = settings.getProperty("title", "Domyślny tytuł");
```

Jeśli w słowniku własności znajduje się własność `title`, parametr `title` zostanie ustawiony na jej łańcuch. W przeciwnym przypadku parametr ten przyjmie wartość `Domyślny tytuł`.

Po drugie, jeśli wpisywanie wartości domyślnej w każdym wywołaniu metody `getProperty` okaże się zbyt żmudne, wszystkie ustawienia domyślne można umieścić w drugorzędnym słowniku własności dostarczonym następnie w konstruktorze słownika głównego.

```
Properties defaultSettings = new Properties();
defaultSettings.put("width", "300");
defaultSettings.put("height", "200");
defaultSettings.put("title", "Domyślny tytuł");
...
Properties settings = new Properties(defaultSettings);
```

Można nawet dostarczyć domyślne ustawienia dla ustawień domyślnych. Wystarczy tylko utworzyć kolejny słownik własności i przekazać go do konstruktora `defaultSettings`. Nie jest to jednak często spotykane rozwiązanie.

Listing 13.2 przedstawia program zapisujący i ładujący ustawienia programu. Zapamiętywane są położenie, rozmiar i tytuł ramki. Wygląd programu można dostosować *według własnego uznania*, edytując plik o nazwie `.corejava/program.properties` znajdujący się w katalogu głównym.

Listing 13.2. `properties/PropertiesTest.java`

```
package properties;

import java.awt.EventQueue;
import java.awt.event.*;
import java.io.*;
import java.util.Properties;

import javax.swing.*;

/**
 * Program testujący mechanizm własności. Ten program zapamiętuje położenie, rozmiar i tytuł ramki.
 * @version 1.01 2015-06-16
 * @author Cay Horstmann
 */
public class PropertiesTest
{
```

```

public static void main(String[] args)
{
    EventQueue.invokeLater(() -> {
        PropertiesFrame frame = new PropertiesFrame();
        frame.setVisible(true);
    });
}

/**
 * Ramka pobierająca dane dotyczące położenia i rozmiaru z pliku własności oraz aktualizująca ten plik
 * w momencie zamykania programu
 */
class PropertiesFrame extends JFrame
{
    private static final int DEFAULT_WIDTH = 300;
    private static final int DEFAULT_HEIGHT = 200;

    private File propertiesFile;
    private Properties settings;

    public PropertiesFrame()
    {
        // Pobranie informacji o położeniu, rozmiarze i tytule z pliku własności

        String userDir = System.getProperty("user.home");
        File propertiesDir = new File(userDir, ".corejava");
        if (!propertiesDir.exists()) propertiesDir.mkdir();
        propertiesFile = new File(propertiesDir, "program.properties");

        Properties defaultSettings = new Properties();
        defaultSettings.put("left", "0");
        defaultSettings.put("top", "0");
        defaultSettings.put("width", "" + DEFAULT_WIDTH);
        defaultSettings.put("height", "" + DEFAULT_HEIGHT);
        defaultSettings.put("title", "");

        settings = new Properties(defaultSettings);

        if (propertiesFile.exists())
            try (InputStream in = new FileInputStream(propertiesFile))
            {
                settings.load(in);
            }
            catch (IOException ex)
            {
                ex.printStackTrace();
            }

        int left = Integer.parseInt(settings.getProperty("left"));
        int top = Integer.parseInt(settings.getProperty("top"));
        int width = Integer.parseInt(settings.getProperty("width"));
        int height = Integer.parseInt(settings.getProperty("height"));
        setBounds(left, top, width, height);

        // Jeśli nie ma tytułu, użytkownik zostanie poproszony o jego podanie

        String title = settings.getProperty("title");

```

```

        if (title.equals("")) title = JOptionPane.showInputDialog("Wpisz tytuł  

        ↳ramki:");
        if (title == null) title = "";
        setTitle(title);

        addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent event)
            {
                settings.put("left", "" + getX());
                settings.put("top", "" + getY());
                settings.put("width", "" + getWidth());
                settings.put("height", "" + getHeight());
                settings.put("title", getTitle());

                try (OutputStream out = new FileOutputStream(propertiesFile))
                {
                    settings.store(out, "Program Properties");
                }
                catch (IOException ex)
                {
                    ex.printStackTrace();
                }
                System.exit(0);
            }
        });
    }
}

```



Klasa `Properties` ze względów historycznych implementuje interfejs `Map<Object, Object>`, a więc można używać metod `get` i `put` interfejsu `Map`. Metoda `get` zwraca obiekt typu `Object`, a metoda `put` pozwala na wstawienie do struktury dowolnego obiektu. Najlepiej cały czas posługiwać się metodami `getProperty` i `setProperty`, które działają na łańcuchach, a nie obiektach.



Obiekty typu `Properties` są zwykłymi tablicami pozbawionymi struktury hierarchicznej. Programiści często tworzą namiastkę hierarchii, odpowiednio nazywając klucze, np. `window.main.color`, `window.main.title` itd. Jednak klasa `Properties` nie zawiera żadnych metod wspomagających organizację takich hierarchii. Do przechowywania skomplikowanych informacji konfiguracyjnych lepiej używać klasy `Preferences`, która została opisana w kolejnym podrozdziale.

`java.util.Properties` **1.0**

■ `Properties()`

Tworzy pusty słownik własności.

■ `Properties(Properties defaults)`

Tworzy słownik własności z zestawem ustawień domyślnych.

Parametr: `defaults` Wartości domyślne

■ `String getProperty(String key)`

Pobiera słownik własności. Zwraca łańcuch skojarzony z kluczem `key` lub łańcuch skojarzony z kluczem `key` w słowniku domyślnym, jeśli nie ma go w aktualnym słowniku, albo wartość `null`, jeśli nie zostanie on znaleziony także w tym drugim słowniku.

Parametr: `key` Klucz, którego wartość ma zostać pobrana.

■ `String getProperty(String key, String defaultValue)`

Pobiera własność z domyślną wartością, jeśli klucz `key` nie zostanie znaleziony. Zwraca łańcuch skojarzony z kluczem `key` lub łańcuch domyślny, jeśli nie ma go w tablicy.

Parametry: `key` Klucz, którego wartość ma zostać pobrana.
 `defaultValue` Wartość zwracana, jeśli dany klucz nie istnieje.

■ `Object setProperty(String key, String value)`

Parametry: `key` Klucz, z którym jest związany łańcuch do ustawienia.
 `value` Wartość, która ma zostać skojarzona z kluczem.

■ `void load(InputStream in) throws IOException`

Ładuje słownik własności ze strumienia wejściowego.

Parametr: `in` Strumień wejściowy

■ `void store(OutputStream out, String header) 1.2`

Zapisuje słownik własności w strumieniu wyjściowym.

Parametry: `out` Strumień wyjściowy
 `header` Nagłówek umieszczany w pierwszym wierszu zapisywanego pliku

`java.lang.System 1.0`

■ `Properties.getProperties()`

Pobiera wszystkie właściwości systemowe. Jeśli aplikacja nie ma uprawnień do pobierania wszystkich właściwości systemowych, generowany jest wyjątek zabezpieczeń.

■ `String getProperty(String key)`

Pobiera właściwość systemową opatrzoną podanym kluczem. Jeśli aplikacja nie ma uprawnień do pobierania tej właściwości systemowej, generowany jest wyjątek zabezpieczeń. Poniższe właściwości można zawsze pobierać:

```
java.version
java.vendor
java.vendor.url
java.class.version
os.name
```

```

os.version
os.arch
file.separator
path.separator
line.separator
java.specification.version
java.vm.specification.version
java.vm.specification.vendor
java.vm.specification.name
java.vm.version
java.vm.vendor
java.vm.name

```



Nazwy wolno dostępnych właściwości systemowych można znaleźć w pliku *security/java.policy* znajdującym się w katalogu środowiska uruchomieniowego Javy.

13.2.2. API Preferences

Klasa `Properties`, mimo iż umożliwia zapisywanie i odczytywanie danych konfiguracyjnych w prosty sposób, ma kilka wad:

- Nie zawsze mamy możliwość zapisania plików konfiguracyjnych w katalogu głównym użytkownika, ponieważ niektóre systemy operacyjne nie znają koncepcji katalogu głównego.
- Nie istnieje standardowa konwencja nazywania plików konfiguracyjnych, co wiąże się z ryzykiem wystąpienia konfliktów nazw, jeśli użytkownik zainstaluje kilka aplikacji Java.

Niektóre systemy operacyjne mają centralne repozytorium, w którym przechowują dane konfiguracyjne. Najlepszym przykładem takiego repozytorium jest rejestr w systemie Microsoft Windows. Klasa `Preferences` pozwala utworzyć takie repozytorium niezależnie od platformy. W systemie Windows klasa ta wykorzystuje rejestr. W systemie Linux informacje te są zapisywane w lokalnym systemie plików. Oczywiście implementacja repozytorium nie ma tajemnic dla programisty używającego klasy `Preferences`.

Repozytorium `Preferences` ma strukturę drzewiastą z nazwami ścieżek do węzłów typu `/com/mycompany/myapp`. Podobnie jak w przypadku pakietów, konfliktów nazw ścieżek unika się, stosując odwrócone nazwy domen. Projektanci tego API zalecają nawet, aby ścieżkom do węzłów konfiguracyjnych nadawać takie same nazwy jak pakietom używanym w programie.

Każdy węzeł w repozytorium ma oddzielną tablicę par klucz – wartość, w której można przechowywać łańcuchy, liczby i tablice bajtów. Nie ma możliwości zapisywania obiektów serializowanych, ponieważ projektanci uznali, że format ten jest zbyt ulotny do długoterminowego przechowywania. Oczywiście ci, którzy się z tym nie zgadzają, mogą zapisywać serializowane obiekty w tablicach bajtów.

Większą elastyczność zapewniają dodatkowe równoległe drzewa. Każdy użytkownik programu ma własne drzewo i dodatkowe drzewo systemowe, które przechowuje ustawienia wspólne wszystkich użytkowników. Odpowiednie drzewo ustawień jest pobierane przez klasę `Preferences` przy użyciu pojęcia bieżącego użytkownika, rozumianego zgodnie z systemem operacyjnym.

Aby uzyskać dostęp do węzła drzewa, należy zacząć od użytkownika root lub katalogu systemowego root:

```
Preferences root = Preferences.userRoot();
```

lub

```
Preferences root = Preferences.systemRoot();
```

Następnie uzyskujemy dostęp do węzła. Można ograniczyć się do podania jego ścieżki:

```
Preferences node = root.node("/com/mycompany/myapp");
```

Za pomocą wygodnego skrótu można pobrać węzeł, którego ścieżka jest taka sama jak nazwa pakietu klasy. Wystarczy pobrać obiekt tej klasy i zastosować poniższe wywołanie:

```
Preferences node = Preferences.userNodeForPackage(obj.getClass());
```

lub

```
Preferences node = Preferences.systemNodeForPackage(obj.getClass());
```

Zazwyczaj obj jest referencją this.

Mając węzeł, można uzyskać dostęp do jego tablicy par klucz – wartość za pomocą następujących metod:

```
String get(String key, String defval)
int getInt(String key, int defval)
long getLong(String key, long defval)
float getFloat(String key, float defval)
double getDouble(String key, double defval)
boolean getBoolean(String key, boolean defval)
byte[] getByteArray(String key, byte[] defval)
```

Należy pamiętać, że przy odczytywaniu informacji trzeba dostarczyć wartość domyślną, na wypadek gdyby w repozytorium brakowało jakichś danych. Wartości domyślne są wymagane z kilku powodów. Danych może brakować, ponieważ użytkownik nigdy nie ustawił danej opcji. Niektóre ograniczone platformy mogą nie mieć repozytorium, a urządzenia przenośne mogą być tymczasowo odłączone od repozytorium.

Do zapisu danych w repozytorium służą metody put:

```
put(String key, String value)
putInt(String key, int value)
```

i tak dalej.

Listę wszystkich kluczy zapisanych w węźle można uzyskać za pomocą metody `String[] keys`. Nie ma jednak obecnie sposobu na sprawdzenie typu wartości określonego klucza.

Centralne repozytoria, takie jak rejestr w systemie Windows, zazwyczaj cierpią z dwóch powodów:

- Z czasem zamieniają się w śmietniko pełne przestarzałych informacji.
- Dane konfiguracyjne plączą się w repozytorium, przez co trudno jest je przenieść na inną platformę.

Klasa Preferences ma rozwiązanie drugiego problemu. Można wyeksportować ustawienia poddrzewa (lub — rzadziej — jednego węzła) za pomocą poniższych metod:

```
void exportSubtree(OutputStream out)
void exportNode(OutputStream out)
```

Dane są zapisywane w formacie XML. Można je zaimportować do innego repozytorium za pomocą następującego wywołania:

```
void importPreferences(InputStream in)
```

Poniżej znajduje się zawartość przykładowego pliku:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE preferences SYSTEM "http://java.sun.com/dtd/preferences.dtd">
<preferences EXTERNAL_XML_VERSION="1.0">
  <root type="user">
    <map/>
    <node name="com">
      <map/>
      <node name="horstmann">
        <map/>
        <node name="corejava">
          <map>
            <entry key="left" value="11"/>
            <entry key="top" value="9"/>
            <entry key="width" value="453"/>
            <entry key="height" value="365"/>
            <entry key="title" value="Witaj, świecie!"/>
          </map>
        </node>
      </node>
    </node>
  </root>
</preferences>
```

Jeśli program wykorzystuje klasę Preferences, należy umożliwić użytkownikowi import i eksport ustawień, co ułatwia przenoszenie ustawień na inny komputer. Program na listingu 13.3 demonstruje omówioną technikę. Zapisuje on położenie, rozmiar i tytuł głównego okna. Po zamknięciu i ponownym uruchomieniu programu okno będzie wyglądało dokładnie tak samo jak przed zamknięciem programu.

Listing 13.3. preferences/PreferencesTest.java

```
package preferences;

import java.awt.*;
import java.io.*;
import java.util.prefs.*;

import javax.swing.*;
import javax.swing.filechooser.*;

/**
 * Program testujący ustawianie preferencji. Zapamiętuje położenie, rozmiar i tytuł ramki.
 * @version 1.03 2015-06-12
 * @author Cay Horstmann
 */
```

```

public class PreferencesTest
{
    public static void main(String[] args)
    {
        EventQueue.invokeLater(() -> {
            PreferencesFrame frame = new PreferencesFrame();
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setVisible(true);
        });
    }
}

/**
 * Ramka pobierająca dane dotyczące położenia i rozmiaru z preferencji użytkownika oraz aktualizująca
 * preferencje w momencie zamykania programu
 */
class PreferencesFrame extends JFrame
{
    private static final int DEFAULT_WIDTH = 300;
    private static final int DEFAULT_HEIGHT = 200;
    private Preferences root = Preferences.userRoot();
    private Preferences node = root.node("/com/horstmann/corejava");

    public PreferencesFrame()
    {
        // Sprawdzanie położenia, rozmiaru i tytułu w preferencjach

        int left = node.getInt("left", 0);
        int top = node.getInt("top", 0);
        int width = node.getInt("width", DEFAULT_WIDTH);
        int height = node.getInt("height", DEFAULT_HEIGHT);
        setBounds(left, top, width, height);

        // Jeśli nie ma tytułu, użytkownik zostanie poproszony o jego podanie

        String title = node.get("title", "");
        if (title.equals(""))
        {
            title = JOptionPane.showInputDialog("Podaj tytuł ramki:");
            if (title == null) title = "";
            setTitle(title);
        }

        // Utworzenie okna wyboru plików wyświetlającego pliki XML

        final JFileChooser chooser = new JFileChooser();
        chooser.setCurrentDirectory(new File("."));
        chooser.setFileFilter(new FileNameExtensionFilter("Pliki XML", "xml"));

        // utworzenie menu

        JMenuBar menuBar = new JMenuBar();
        setJMenuBar(menuBar);
        JMenu menu = new JMenu("Plik");
        menuBar.add(menu);

        JMenuItem exportItem = new JMenuItem("Eksportuj preferencje");
        menu.add(exportItem);
        exportItem
            .addActionListener(event -> {

```

```
        if (chooser.showSaveDialog(PreferencesFrame.this) ==
            JFileChooser.APPROVE_OPTION)
        {
            try
            {
                savePreferences();
                OutputStream out = new FileOutputStream(chooser
                    .getSelectedFile());
                node.exportSubtree(out);
                out.close();
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
        }
    }
});

JMenuItem importItem = new JMenuItem("Importuj preferencje");
menu.add(importItem);
importItem
    .addActionListener(event -> {
        if (chooser.showOpenDialog(PreferencesFrame.this) ==
            JFileChooser.APPROVE_OPTION)
        {
            try
            {
                InputStream in = new FileInputStream(chooser
                    .getSelectedFile());
                Preferences.importPreferences(in);
                in.close();
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
        }
    })

JMenuItem exitItem = new JMenuItem("Zamknij");
menu.add(exitItem);
exitItem.addActionListener(event -> {
    savePreferences();
    System.exit(0);
});
}

public void savePreferences()
{
    node.putInt("left", getX());
    node.putInt("top", getY());
    node.putInt("width", getWidth());
    node.putInt("height", getHeight());
    node.put("title", getTitle());
}
}
```

`java.util.prefs.Preferences 1.4`

- `Preferences userRoot()`

Zwraca węzeł preferencji root użytkownika programu.

- `Preferences systemRoot()`

Zwraca węzeł preferencji root systemu.

- `Preferences node(String path)`

Zwraca węzeł, do którego można uzyskać dostęp z bieżącego węzła za pośrednictwem podanej ścieżki `path`. Jeśli ścieżka jest bezwzględna (czyli zaczyna się od znaku `/`), szukanie węzła zaczyna się od korzenia drzewa zawierającego ten węzeł preferencji. Jeśli węzeł w podanej ścieżce nie istnieje, zostanie utworzony.

- `Preferences userNodeForPackage(Class c1)`

- `Preferences systemNodeForPackage(Class c1)`

Zwraca węzeł w bieżącym drzewie użytkownika lub drzewo systemowe, którego ścieżka bezwzględna węzła odpowiada nazwie pakietu klasy `c1`.

- `String[] keys()`

Zwraca wszystkie klucze należące do węzła.

- `String get(String key, String defval)`

- `int getInt(String key, int defval)`

- `long getLong(String key, long defval)`

- `float getFloat(String key, float defval)`

- `double getDouble(String key, double defval)`

- `boolean getBoolean(String key, boolean defval)`

- `byte[] getByteArray(String key, byte[] defval)`

Zwraca wartość skojarzoną z danym kluczem lub podaną wartość domyślną, jeśli z kluczem nie jest skojarzona żadna wartość lub skojarzona wartość jest nieprawidłowego typu, lub magazyn preferencji nie jest dostępny.

- `void put(String key, String value)`

- `void putInt(String key, int value)`

- `void putLong(String key, long value)`

- `void putFloat(String key, float value)`

- `void putDouble(String key, double value)`

- `void putBoolean(String key, boolean value)`

- `void putByteArray(String key, byte[] value)`

Zapisuje parę klucz – wartość w węźle.

- `void exportSubtree(OutputStream out)`

Zapisuje preferencje węzła i jego potomków w określonym strumieniu.

- `void exportNode(OutputStream out)`

Zapisuje preferencje węzła (ale nie jego potomków) w określonym strumieniu.

- `void importPreferences(InputStream in)`

Importuje preferencje zawarte w określonym strumieniu.

13.3. Moduły ładowania usług

Niektóre aplikacje specjalnie buduje się tak, by było łatwo rozbudowywać ich funkcjonalność za pomocą wtyczek. Istnieją nawet platformy ułatwiające to zadanie, np. OSGi (<http://osgi.org>), które wykorzystuje się w środowiskach programistycznych, serwerach aplikacji i innych złożonych programach. Opis tych platform wykracza daleko poza temat tej książki, ale warto wiedzieć, że JDK również zawiera prosty mechanizm do ładowania wtyczek. Jego opis zamieszczamy poniżej.

Twórcy programów obsługujących wtyczki często pozostawiają programistom tych dodatków pewien zakres wolności pod względem sposobu implementacji swoich produktów. Dobrze jest też mieć do wyboru kilka implementacji. Klasa `ServiceLoader` sprawia, że ładowanie wtyczek spełniających wymagania określonego wspólnego interfejsu jest bardzo łatwe.

Należy zdefiniować interfejs (lub, jeśli ktoś woli, nadklasę) z metodami, które muszą być zdefiniowane przez każdy egzemplarz usługi. Powiedzmy na przykład, że stworzyliśmy usługę szyfrowania:

```
package serviceLoader;

public interface Cipher
{
    byte[] encrypt(byte[] source, byte[] key);
    byte[] decrypt(byte[] source, byte[] key);
    int strength();
}
```

Dostawca usług zapewnia przynajmniej jedną klasę implementującą tę usługę, np.:

```
package serviceLoader.impl;

public class CaesarCipher implements Cipher
{
    public byte[] encrypt(byte[] source, byte[] key)
    {
        byte[] result = new byte[source.length];
        for (int i = 0; i < source.length; i++)
            result[i] = (byte)(source[i] + key[0]);
        return result;
    }
}
```



```

public byte[] decrypt(byte[] source, byte[] key)
{
    return encrypt(source, new byte[] { (byte) -key[0] });
}

public int strength() { return 1; }
}

```

Klasy implementacyjne mogą należeć do dowolnego pakietu, tzn. nie muszą się znajdować w tym samym pakiecie co interfejs usługi. Każda z nich musi natomiast mieć konstruktor bezargumentowy.

Teraz dodamy nazwy klas do pliku tekstowego UTF-8 znajdującego się w katalogu *META-INF/services*. Nazwa tego pliku musi się zgadzać z pełną nazwą klasy. W naszym przypadku oznacza to, że plik *META-INF/services/serviceLoader.Cipher* będzie zawierał następujący wiersz:

```
serviceLoader.impl.CaesarCipher
```

Teraz dostarczyliśmy tylko jedną klasę implementacyjną, ale w razie potrzeby możemy dodać ich więcej i wybierać tę, która w danym przypadku jest najodpowiedniejsza.

Po tych przygotowaniach program inicjalizuje moduł ładujący usługi w następujący sposób:

```
public static ServiceLoader<Cipher> cipherLoader = ServiceLoader.load(Cipher.class);
```

Wystarczy to zrobić tylko raz w programie.

Metoda iterator modułu ładowania usług zwraca iterator do przeglądania wszystkich dostarczonych implementacji usługi. (Szerzej na temat iteratorów piszemy w rozdziale 9.). Aby je przejrzeć, najłatwiej posłużyć się rozszerzoną pętlą *for*, w której można wybrać odpowiedni obiekt do świadczenia usługi.

```

public static Cipher getCipher(int minStrength)
{
    for (Cipher cipher : cipherLoader) //niejawnie wywołuje cipherLoader.iterator()
    {
        if (cipher.strength() >= minStrength) return cipher;
    }
    return null;
}

```

java.util.ServiceLoader<S> 1.6

- `static <S> ServiceLoader<S> load(Class<S> service)`

Tworzy moduł ładowania usług służący do wczytywania klas implementujących określony interfejs usługowy.

- `Iterator<S> iterator()`

Zwraca iterator, który leniwie ładuje klasy usług, tzn. załadowanie klasy następuje po przesunięciu iteratora do przodu.

13.4. Aplety

Aplety to programy w języku Java dołączane do stron HTML. Strona HTML musi poinformować przeglądarkę, które aplety ma załadować oraz gdzie mają one być rozmieszczone. Jak nietrudno się domyślić, znacznik służący do wstawiania apletów musi dostarczać informacje dotyczące lokalizacji plików klas oraz samego apletu (jego rozmiaru, lokalizacji itd.). Przeglądarka pobiera pliki klas z internetu (lub katalogu na urządzeniu użytkownika) i automatycznie uruchamia aplet.

Na początku istnienia apletów jedyną przeglądarką, która je obsługiwała, była HotJava firmy Sun. Oczywiście znalazło się niewiele osób, które były skłonne używać oddzielnej przeglądarki dla jednej dodatkowej funkcji. Aplety zyskały prawdziwą popularność z chwilą dołączenia przez firmę Netscape maszyny wirtualnej Javy do przeglądarki Navigator. Niedługo później to samo zrobiła firma Microsoft w przeglądarce Internet Explorer. Niestety Microsoft podciął skrzydła Netscape, opornie dodając obsługę starych wersji Javy w Internet Explorerze, aż w końcu całkiem tego zaniechał.

Problem ten rozwiązuje narzędzie Java Plug-in. Integruje się ono z przeglądarkami jako rozszerzenie, co umożliwia uruchamianie apletów przy wykorzystaniu zewnętrznego środowiska uruchomieniowego Javy.

Przez wiele lat było to dobre rozwiązanie, a jako aplety powszechnie implementowało się materiały edukacyjne, aplikacje biznesowe oraz niektóre gry. Niestety firma Sun, a po jej zniknięciu firma Oracle niespiesznie łączyła luki w zabezpieczeniach maszyny wirtualnej Javy, które od czasu do czasu ktoś wykrywał i wykorzystywał w niecnym zamiarach. Słabo zabezpieczona maszyna wirtualna narażała użytkowników na spore ryzyko, więc producenci przeglądarek postanowili ograniczyć możliwość korzystania z Javy w swoich produktach. Niektórzy zezwalali na używanie tylko najnowszej wersji wtyczki Java, a inni porzucili wsparcie dla architektury wtyczkowej. Niestety reakcja firmy Oracle na to, co się dzieje, była rozczarowująca. Firma zaczęła wymagać, aby wszystkie aplety były podpisane cyfrowo (zobacz sekcję 13.4.9, „Podpisywanie kodu”).

Z tych powodów dziś wdrożenie apletu Java jest sporym wyzwaniem dla programisty, a dla użytkownika — jego uruchomienie. Dlatego też dalsza część tego rozdziału zainteresuje głównie tych czytelników, którzy spotykają się w pracy ze starymi apletami.

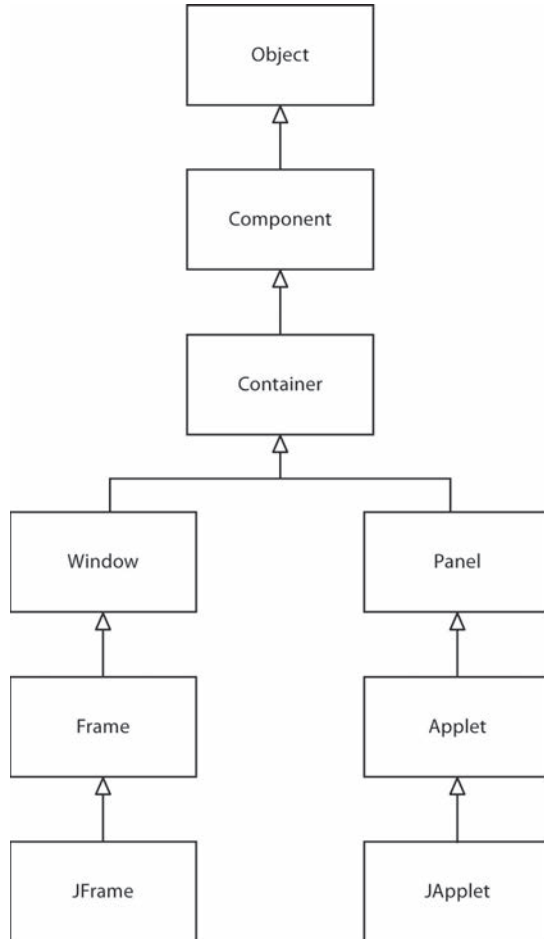


Aby móc uruchamiać aplety opisywane w tym rozdziale, należy zainstalować najnowszą wersję narzędzia Java Plug-in i upewnić się, że przeglądarka jest połączona z wtyczką. Dodatkowo, aby móc testować aplety, należy tak skonfigurować wtyczkę, by ufała plikom lokalnym. Piszemy o tym w podrozdziale 2.5, „Tworzenie i uruchamianie apletów”.

13.4.1. Prosty aplet

Aby tradycji stało się zadość, przerobimy program *NotHelloWorld* na aplet. Aplet to zwykła klasa Javy rozszerzająca klasę `java.applet.Applet`. Do implementacji apletów użyjemy Swinga. Wszystkie nasze aplety będą rozszerzały klasę `JApplet`, która jest nadklasą apletów Swing. Jak widać na rysunku 13.2, klasa `JApplet` jest bezpośrednią podklasą klasy `Applet`.

Rysunek 13.2.
Diagram
dziedziczenia
klasy `Applet`



Jeśli aplet zawiera składniki Swing, należy rozszerzyć klasę `JApplet`. Komponenty Swing w zwykłym kontenerze `Applet` nie są poprawnie rysowane.

Na listingu 13.4 przedstawiona jest apletowa wersja programu „nie powitalnego”.

Zwróć uwagę na duże podobieństwo do wersji z rozdziału 10. Ponieważ aplet działa w oknie przeglądarki, nie trzeba było definiować metody zamykającej.

Listing 13.4. applet/NotHelloWorld.java

```

package applet;

import java.awt.*;
import javax.swing.*;

/**
 * @version 1.24 2015-06-12
 * @author Cay Horstmann
 */
public class NotHelloWorld extends JApplet
{
    public void init()
    {
        EventQueue.invokeLater(() -> {
            JLabel label = new JLabel("To nie jest aplet Witaj, świecie",
                SwingConstants.CENTER);
            add(label);
        });
    }
}

```

Uruchomienie powyższego apletu wymaga wykonania trzech czynności:

1. Kompilacji plików źródłowych na pliki klas.
2. Spakowania klas do pliku JAR (zobacz sekcję 13.1.1, „Tworzenie plików JAR”).
3. Utworzenia pliku HTML zawierającego informacje o lokalizacji plików klas oraz rozmiarze apletu.

Poniżej znajduje się jego zawartość:

```

<applet code="applet/NotHelloWorld.class" width="300" height="300">
</applet>

```

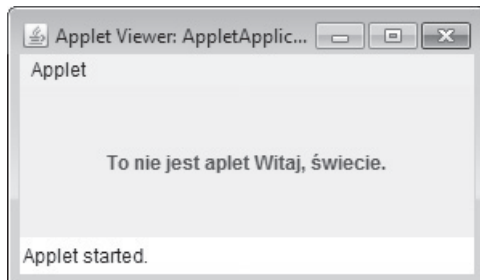
Dobrym pomysłem jest przetestowanie apletu we wchodzącej w skład pakietu JDK **przeglądarce apletów** (ang. *applet viewer*) przed otwarciem go w przeglądarce internetowej. Poniższe polecenie wiersza poleceń otwiera nasz aplet we wspomnianej przeglądarce:

```
appletviewer NotHelloWorldApplet.html
```

Argumentem narzędzia appletviewer jest nazwa pliku HTML, nie pliku klasy. Rysunek 13.3 przedstawia nasz aplet w przeglądarce apletów.

Rysunek 13.3.

Aplet
w przeglądarce
apletów

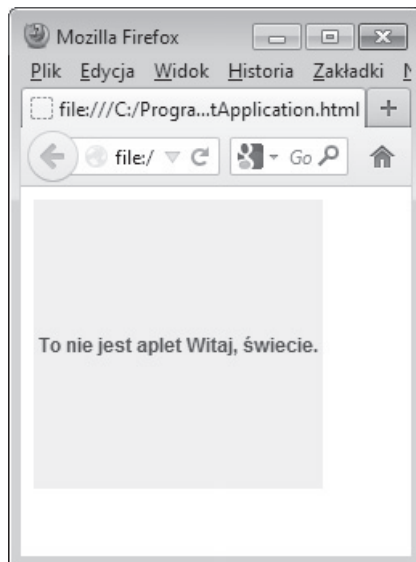


Przeglądarka apletów dobrze sprawdza się jako pierwszy etap testowania. Trzeba jednak w końcu uruchomić aplet w przeglądarce, aby sprawdzić, jak będzie się prezentował użytkownikowi. Przeglądarka apletów pokazuje sam aplet bez otaczającego go kodu HTML. Jeśli strona HTML zawiera kilka znaczników `applet`, przeglądarka otworzy kilka okien.

Aby obejrzeć swój aplet w przeglądarce, wystarczy załadować w niej odpowiednią stronę HTML (rysunek 13.4). Jeśli aplet nie pojawia się, należy zainstalować narzędzie Java Plug-in i zezwolić mu na ładowanie niepodpisanych apletów lokalnych (zobacz podrozdział 2.5, „Tworzenie i uruchamianie apletów”).

Rysunek 13.4.

Aplet
w przeglądarce



Jeśli w aplecie zostaną wprowadzone jakieś zmiany i zostanie on raz jeszcze poddany kompilacji, konieczne jest ponowne uruchomienie przeglądarki, aby załadowała nowe pliki klas. Samo odświeżenie strony nie spowoduje załadowania nowej wersji apletu. Bywa to kłopotliwe przy szukaniu błędów. Można uniknąć ponownego uruchamiania przeglądarki dzięki użyciu **konsoli Javy**. Należy uruchomić tę konsolę i wydać polecenie `x`, które czyści pamięć programu ładującego klasy. Wtedy po odświeżeniu strony zostanie załadowana nowa wersja apletu. W systemie Windows należy otworzyć panel kontrolny Java Plug-in znajdujący się w *Panelu sterowania*. W systemie Linux należy użyć polecenia `jcontrol` i zażądać wyświetlenia panelu kontrolnego Javy. Konsola będzie się pojawiać za każdym razem, kiedy ładowany jest aplet.

Graficzne aplikacje w Javie można z łatwością przekonwertować na aplety. Cały kod dotyczący interfejsu użytkownika pozostaje bez zmian. Oto lista niezbędnych czynności:

1. Utwórz stronę HTML z odpowiednim znacznikiem wstawiającym aplet.
2. Utwórz podklasę klasy `JApplet`. Zdefiniuj tę klasę jako publiczną. W przeciwnym razie apletu nie będzie można załadować.
3. Usuń z aplikacji metodę `main`. Nie twórz ramki dla aplikacji, ponieważ będzie ona wyświetlana w oknie przeglądarki.

4. Przenieś kod inicjalizujący z konstruktora ramki do metody `init` apletu. Nie trzeba jawnie konstruować obiektu apletu — przeglądarka robi to automatycznie i wywołuje metodę `init`.
5. Usuń wywołanie metody `setSize`. W apletach za rozmiary odpowiadają parametry HTML `width` i `height`.
6. Usuń wywołanie metody `setDefaultCloseOperation`. Apletu nie można zamknąć — jego działanie kończy się w chwili zamknięcia przeglądarki.
7. Jeśli w programie znajduje się wywołanie metody `setTitle`, należy je usunąć. Aplety nie mają pasków tytułu (można oczywiście nadać tytuł samej stronie HTML za pomocą znacznika `title`).
8. Nie wywołuj metody `setVisible(true)`. Aplet jest wyświetlany automatycznie.

java.applet.Applet **1.0**

■ `void init()`

Jest wywoływana przy pierwszym ładowaniu apletu. Metodę tę należy przesłonić i umieścić w niej cały kod inicjalizujący.

■ `void start()`

Należy przesłonić tę metodę i umieścić w niej kod, który ma być wykonywany *za każdym razem*, gdy użytkownik odwiedza stronę zawierającą ten aplet. Do typowych działań należy tu reaktywacja wątku.

■ `void stop()`

Należy przesłonić tę metodę i umieścić w niej kod, który ma być wykonywany *za każdym razem*, gdy użytkownik opuszcza stronę zawierającą ten aplet. Do typowych działań należy tu dezaktywacja wątku.

■ `void destroy()`

Metodę tę należy przededefiniować, wstawiając do niej kod wykonywany w momencie zamknięcia przeglądarki.

■ `void resize(int width, int height)`

Wymusza zmianę rozmiaru apletu. Byłaby to doskonała metoda, gdyby działała na stronach internetowych. Niestety obecnie nie działa w przeglądarkach, ponieważ zakłóca ich mechanizm rozkładu elementów na stronie.

13.4.2. Znacznik applet i jego atrybuty

Znacznik applet w najprostszej postaci może wyglądać następująco:

```
<applet code="NotHelloWorld.class" archive="NotHelloWorld.jar" width="300"
height="100"></applet>
```

Znacznik applet ma następujące atrybuty:

■ width, height

Atrybuty te są wymagane i określają szerokość i wysokość apletu w pikselach. W przeglądarce apletów wymiary te są traktowane jako początkowe, ale rozmiar każdego okna tej przeglądarki można zmienić. W przeglądarce internetowej *nie ma możliwości* zmiany rozmiaru apletu. Optymalny rozmiar apletu, tak aby wyglądał dobrze u każdego użytkownika, należy określić metodą prób i błędów.

■ align

Określa wyrównanie apletu. Wartości tego atrybutu są takie same jak atrybutu align znacznika img.

■ vspace, hspace

Określają liczbę pikseli nad i pod apletem (vspace) oraz po jego obu stronach (hspace).

■ code

Określa nazwę pliku klasy apletu.

Ścieżka musi się zgadzać z pakietem, do którego należy klasa. Jeśli na przykład klasa apletu należy do pakietu `com.mycompany`, atrybut wygląda następująco `code="com/mycompany/MyApplet.class"`. Można też stosować alternatywny zapis w postaci `code="com.mycompany.MyApplet.class"`.

Atrybut `code` określa tylko nazwę klasy apletu. Oczywiście sam aplet może zawierać także inne pliki klas. Kiedy przeglądarka załaduje klasę zawierającą aplet, zorientuje się, że potrzebne są dodatkowe klasy, i je również załaduje.

■ archive

Określa listę plików JAR lub plików zawierających klasy i inne zasoby apletu, które są pobierane z serwera przed jego załadowaniem. Pliki JAR na liście są rozdzielane przecinkami:

```
<applet code="MyApplet.class"
  archive="MyClasses.jar,corejava/CoreJavaClasses.jar"
  width="100" height="150">
```

■ codebase

Określa adres URL, pod którym znajdują się pliki JAR (a kiedyś także pliki klas) do załadowania.

■ object

Przestarzały atrybut określający nazwę pliku zawierającego poddany *serializacji* obiekt apletu, który miał być wykorzystany w celu przechowania stanu apletu. Funkcja ta jest już bezużyteczna, ponieważ nie ma możliwości podpisania pliku poddanego serializacji.

■ alt

W atrybucie tym można zdefiniować tekst do wyświetlenia, gdy przeglądarka nie obsługuje Javy.

Jeśli przeglądarka w ogóle nie rozpoznaje appletów, czyli pochodzi z czasów prehistorycznych, ignoruje znaczniki `applet` i `param`. W takiej sytuacji zostanie wyświetlony tekst znajdujący się pomiędzy znacznikami `<applet>` i `</applet>`. Natomiast przeglądarki obsługujące applety nie wyświetlają tego tekstu. Na przykład:

```
<applet ... alt="Gdyby Twoja przeglądarka obsługiwała Javę, w tym miejscu
↳byłby widoczny mój applet.">
    Gdyby Twoja przeglądarka obsługiwała Javę, w tym miejscu byłby widoczny mój applet.
</applet>
```

■ name

Twórcy skryptów wykorzystują ten atrybut do odwoływania się do appletu w swoich skryptach. Przeglądarki Netscape i Internet Explorer zezwalają na wywołanie metod appletów na stronie za pośrednictwem JavaScriptu.

Aby uzyskać dostęp do appletu z poziomu JavaScriptu, najpierw należy nadać mu nazwę:

```
<applet ... name="mine"></applet>
```

Dzięki temu można się do niego odwoływać za pomocą zapisu `document.applets.nazwaappletu`. Na przykład:

```
var myApplet = document.applets.mine;
```

Można wywoływać metody appletu:

```
myApplet.init();
```

Atrybut `name` ma także kluczowe znaczenie w sytuacjach, kiedy dwa applety znajdujące się na tej samej stronie mają się ze sobą bezpośrednio komunikować. Należy nadać nazwę każdemu appletowi. Łańcuch ten należy przekazać do metody `getApplet` z klasy `AppletContext`. Mechanizm ten, o nazwie **kommunikacja między appletami** (ang. *inter-applet communication*), został opisany nieco dalej w tym rozdziale.



Na stronie <http://www.javaworld.com/javatips/jw-javatip80.html> Francis Lu wykorzystuje mechanizm komunikacji Javy z JavaScriptem do rozwiązania odwiecznego problemu zmiany rozmiaru appletu, na stałe ustawionego przez atrybuty `width` i `height`. Jest to dobry przykład integracji tych dwóch języków.

13.4.3. Parametry przekazujące informacje do appletów

Podobnie jak aplikacje wykorzystują informacje podawane za pośrednictwem wiersza poleceń, applety używają parametrów podawanych w plikach HTML. Służy do tego znacznik `param` i jego atrybuty. Wyobraźmy sobie, że chcemy, aby na stronie internetowej można było ustawić krój czcionki używanej w appiecie. Można do tego użyć następującego kodu HTML:

```
<applet code="FontParamApplet.class" width="200" height="200">
  <param name="font" value="Helvetica"/>
</applet>
```


Następnie wartość parametru pobieramy za pomocą metody `getParameter` z klasy `Applet`:

```
public class FontParamApplet extends JApplet
{
    public void init()
    {
        String fontName = getParameter("font");
        . . .
    }
    . . .
}
```



Metodę `getParameter` można wywoływać tylko w metodzie `init` apletu, nie w konstruktorze, ponieważ w chwili jego wykonywania parametry nie są jeszcze gotowe. Ponieważ układ większości bardziej rozbudowanych apletów jest zdeterminowany przez parametry, zalecamy zaniechanie używania konstruktorów w apletach. Cały kod inicjalizujący można umieścić w metodzie `init`.

Parametry są zawsze zwracane jako łańcuchy. Jeśli wymagana jest liczba, łańcuch trzeba przekonwertować na typ liczbowy. Służą do tego standardowe metody, takie jak `parseInt` z klasy `Integer`.

Na przykład kod HTML zawierający parametr `size`, który określa rozmiar czcionki, mógłby wyglądać następująco:

```
<applet code="FontParamApplet.class" ...>
  <param name="font" value="Helvetica"/>
  <param name="size" value="24"/>
</applet>
```

Poniższy fragment kodu demonstruje sposób odczytu parametru liczbowego:

```
public class FontParamApplet extends JApplet
{
    public void init()
    {
        String fontName = getParameter("font");
        int fontSize = Integer.parseInt(getParameter("size"));
        . . .
    }
}
```



Przy porównywaniu wartości atrybutu `name` ze znacznika `param` z argumentem metody `getParameter` nie jest rozpoznawana wielkość liter.

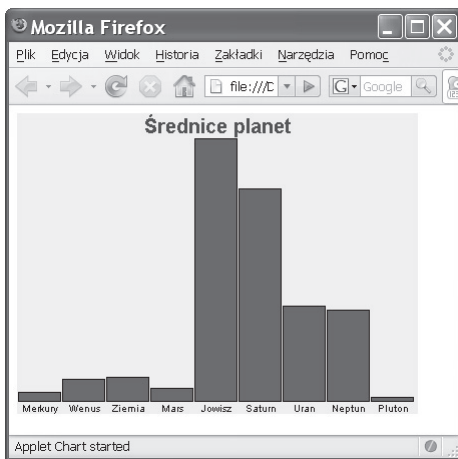
Poza upewnieniem się, że parametry w kodzie pasują, należy sprawdzić, czy parametr `size` został ustawiony, czy nie. Służy do tego prosty test na obecność wartości `null`. Na przykład:

```
int fontSize;
String sizeString = getParameter("size");
if (sizeString == null) fontSize = 12;
else fontSize = Integer.parseInt(sizeString);
```

Rysunek 13.5 przedstawia aplet rysujący wykres słupkowy, który w dużym stopniu wykorzystuje parametry.

Rysunek 13.5.

Aplet
wyświetlający
wykres



Aplet ten pobiera etykiety i wysokości słupków z wartości atrybutów znacznika param. Poniżej znajduje się kod HTML strony widocznej na rysunku 13.5.

```
<applet code="Chart.class" width="400" height="300">
  <param name="title" value="Średnice planet"/>
  <param name="values" value="9"/>
  <param name="name.1" value="Merkury"/>
  <param name="name.2" value="Wenus"/>
  <param name="name.3" value="Ziemia"/>
  <param name="name.4" value="Mars"/>
  <param name="name.5" value="Jowisz"/>
  <param name="name.6" value="Saturn"/>
  <param name="name.7" value="Uran"/>
  <param name="name.8" value="Neptun"/>
  <param name="name.9" value="Pluton"/>
  <param name="value.1" value="3100"/>
  <param name="value.2" value="7500"/>
  <param name="value.3" value="8000"/>
  <param name="value.4" value="4200"/>
  <param name="value.5" value="88000"/>
  <param name="value.6" value="71000"/>
  <param name="value.7" value="32000"/>
  <param name="value.8" value="30600"/>
  <param name="value.9" value="1430"/>
</applet>
```

Można było utworzyć w aplocie tablicę łańcuchów i tablicę liczb, ale wykorzystanie mechanizmu parametrów ma dwie zalety. Po pierwsze, na jednej stronie można wyświetlić kilka kopii tego samego apletu, pokazujących różne wykresy — trzeba zastosować kilka znaczników applet z różnymi zestawami parametrów. Po drugie, można zmieniać dane przedstawione na wykresie. Wprawdzie średnice planet jeszcze przez jakiś czas się nie zmieniają, ale wyobraźmy sobie, że na stronie przedstawiamy tygodniowy wykres sprzedaży. Stronę internetową łatwo się aktualizuje, ponieważ jest to czysty tekst. Edytowanie i kompilowanie plików Javy wymaga znacznie więcej wysiłku.

Istnieją nawet komercyjne komponenty JavaBean (tak zwane beany), które tworzą o wiele atrakcyjniejsze wykresy. Podając parametry takiemu zakupionemu komponentowi, nie trzeba nawet wiedzieć nic na temat tworzenia wykresów.

Listing 13.5 przedstawia kod źródłowy omawianego apletu rysującego wykres. Należy zauważyć, że metoda `init` pobiera parametry, a metoda `paintComponent` rysuje wykres.

Listing 13.5. chart/Chart.java

```
package chart;

import java.awt.*;
import java.awt.font.*;
import java.awt.geom.*;
import javax.swing.*;

/**
 * @version 1.34 2015-06-12
 * @author Cay Horstmann
 */
public class Chart extends JApplet
{
    public void init()
    {
        EventQueue.invokeLater(() -> {
            String v = getParameter("values");
            if (v == null) return;
            int n = Integer.parseInt(v);
            double[] values = new double[n];
            String[] names = new String[n];
            for (int i = 0; i < n; i++)
            {
                values[i] = Double.parseDouble(getParameter("value." + (i + 1)));
                names[i] = getParameter("name." + (i + 1));
            }

            add(new ChartComponent(values, names, getParameter("title")));
        });
    }
}

/**
 * Komponent rysujący wykres słupkowy.
 */
class ChartComponent extends JComponent
{
    private double[] values;
    private String[] names;
    private String title;

    /**
     * Tworzy obiekt typu ChartComponent.
     * @param v tablica wartości wykresu
     * @param n tablica nazw wartości
     * @param t tytuł wykresu
     */
    public ChartComponent(double[] v, String[] n, String t)
    {

```

```
        values = v;
        names = n;
        title = t;
    }

    public void paintComponent(Graphics g)
    {
        Graphics2D g2 = (Graphics2D) g;

        // Obliczanie wartości minimalnej i maksymalnej.
        if (values == null) return;
        double minValue = 0;
        double maxValue = 0;
        for (double v : values)
        {
            if (minValue > v) minValue = v;
            if (maxValue < v) maxValue = v;
        }
        if (maxValue == minValue) return;

        int panelWidth = getWidth();
        int panelHeight = getHeight();

        Font titleFont = new Font("SansSerif", Font.BOLD, 20);
        Font labelFont = new Font("SansSerif", Font.PLAIN, 10);

        // Obliczanie szerokości tytułu.
        FontRenderContext context = g2.getFontRenderContext();
        Rectangle2D titleBounds = titleFont.getStringBounds(title, context);
        double titleWidth = titleBounds.getWidth();
        double top = titleBounds.getHeight();

        // Rysowanie tytułu.
        double y = -titleBounds.getY(); // wysokość
        double x = (panelWidth - titleWidth) / 2;
        g2.setFont(titleFont);
        g2.drawString(title, (float) x, (float) y);

        // Obliczanie szerokości etykiet słupków.
        LineMetrics labelMetrics = labelFont.getLineMetrics("", context);
        double bottom = labelMetrics.getHeight();

        y = panelHeight - labelMetrics.getDescent();
        g2.setFont(labelFont);

        // Obliczanie współczynnika skali i szerokości słupków.
        double scale = (panelHeight - top - bottom) / (maxValue - minValue);
        int barWidth = panelWidth / values.length;

        // Rysowanie słupków.
        for (int i = 0; i < values.length; i++)
        {
            // Uzyskanie współrzędnych prostokąta tworzącego słupek.
            double x1 = i * barWidth + 1;
            double y1 = top;
            double height = values[i] * scale;
            if (values[i] >= 0) y1 += (maxValue - values[i]) * scale;
            else
            {

```

```

        y1 += maxValue * scale;
        height = -height;
    }

    // Wypełnienie słupka i rysowanie jego obrysu.
    Rectangle2D rect = new Rectangle2D.Double(x1, y1, barWidth - 2, height);
    g2.setPaint(Color.RED);
    g2.fill(rect);
    g2.setPaint(Color.BLACK);
    g2.draw(rect);

    // Rysowanie etykiety na środku pod słupkiem.
    Rectangle2D labelBounds = labelFont.getStringBounds(names[i], context);

    double labelWidth = labelBounds.getWidth();
    x = x1 + (barWidth - labelWidth) / 2;
    g2.drawString(names[i], (float) x, (float) y);
    }
}

```

```
java.applet.Applet 1.0
```

■ `public String getParameter(String name)`

Pobiera wartość parametru zdefiniowanego w znaczniku `param` na stronie internetowej ładującej aplet. W łańcuchu `name` rozpoznawane są małe i wielkie litery.

■ `public String getAppletInfo()`

Tę metodę wielu programistów przeddefiniowuje, aby zwracała łańcuch zawierający informacje o autorze, wersji i prawach autorskich do apletu. Informacje te należy podawać poprzez przesłonięcie tej metody w klasie apletu.

■ `public String[][] getParameterInfo()`

Tę metodę można przeddefiniować, aby zwracała tablicę opcji znacznika `param`, obsługiwanych przez aplet. Każdy wiersz zawiera trzy pozycje: nazwę, typ i opis parametru. Na przykład:

```

"fps", "1-10", "ramek na sekundę"
"repeat", "boolean", "powtórzyć pętlę obrazu?"
"images", "url", "katalog zawierający obrazy"

```

13.4.4. Dostęp do obrazów i plików audio

W apletach można używać obrazów i plików audio. W chwili pisania tego tekstu obsługiwane były następujące formaty plików graficznych: GIF, PNG i JPEG, oraz plików audio: AU, AIFF, WAV i MIDI. Animowane gify są także poprawnie obsługiwane.

Lokalizację obrazów i plików audio określa się za pomocą względnych adresów URL. Bazowy adres URL zazwyczaj sprawdza się za pomocą metody `getDocumentBase` lub `getCodeBase`. Pierwsza z wymienionych metod pobiera adres URL strony HTML zawierającej aplet, a druga adres URL katalogu bazowego kodu.

Bazowy adres URL i lokalizację pliku należy przekazać do metody `getImage` lub `getAudioClip`. Na przykład:

```
Image cat = getImage(getCodeBase(), "images/cat.gif");
AudioClip meow = getAudioClip(getCodeBase(), "audio/meow.au");
```

Wyświetlania obrazów nauczyliśmy się w rozdziale 10. Jeśli chodzi o pliki audio, wystarczy wywołać metodę `play`. Metodę `play` z klasy `Applet` można nawet wywołać bez uprzedniego załadowania pliku audio.

```
play(getCodeBase(), "audio/meow.au");
```

`java.applet.Applet` **1.0**

- `URL getDocumentBase()`

Pobiera adres URL strony zawierającej aplet.

- `URL getCodeBase()`

Pobiera adres URL katalogu bazowego z kodem, z którego ładowany jest aplet. Jest to albo bezwzględny adres URL katalogu wskazywanego przez atrybut `codebase`, albo adres pliku HTML, jeśli atrybut `codebase` nie istnieje.

- `void play(URL url)`

- `void play(URL url, String name)`

Pierwsza wersja odtwarza plik dźwiękowy znajdujący się pod podanym adresem URL. Druga tworzy ścieżkę względną wobec podanego adresu URL z podanego łańcucha. Jeśli pliku audio nie ma, nic się nie dzieje.

- `AudioClip getAudioClip(URL url)`

- `AudioClip getAudioClip(URL url, String name)`

Pierwsza wersja odtwarza plik dźwiękowy znajdujący się pod podanym adresem URL. Druga tworzy ścieżkę względną wobec podanego adresu URL z podanego łańcucha. Jeśli plik audio nie istnieje, metody te zwracają wartość `null`.

- `Image getImage(URL url)`

- `Image getImage(URL url, String name)`

Zwraca obiekt typu `Image` zawierający obraz wskazywany przez adres URL. Jeśli obraz nie istnieje, zwracana jest natychmiast wartość `null`. W przeciwnym przypadku zostaje uruchomiony osobny wątek ładujący obraz.

13.4.5. Środowisko działania apletu

Aplety rezydują w przeglądarce internetowej lub przeglądarce apletów. Mogą one wysyłać do przeglądarek żądania dotyczące wykonania określonych działań, typu pobranie pliku audio, wyświetlenie komunikatu w pasku stanu lub otwarcie nowej strony. Przeglądarka może wykonać żądanie lub je zignorować. Jeśli na przykład aplet działający w przeglądarce apletów zgłosi żądanie otwarcia nowej strony, zostanie ono zignorowane.

Do komunikacji z przeglądarką aplet wykorzystuje metodę `getAppletContext`. Zwraca ona obiekt implementujący interfejs typu `AppletContext`. Konkretną implementację tego inter-

fejsu można traktować jako ścieżkę komunikacyjną pomiędzy apletem a przeglądarką. Poza metodami `getAudioClip` i `getImage` interfejs `AppletContext` zawiera także inne przydatne metody, które opisujemy w kilku kolejnych podrozdziałach.

13.4.6. Komunikacja pomiędzy apletami

Na jednej stronie internetowej może się znajdować kilka apletów. Jeśli wszystkie one pochodzą z jednej bazy kodowej, mogą się ze sobą komunikować. Jest to jednak zaawansowana technika, której nie używa się zbyt często.

Jeśli każdy aplet w pliku HTML ma określoną nazwę w atrybucie `name`, za pomocą metody `getApplet` z interfejsu `AppletContext` można utworzyć odwołanie do tego apletu. Jeśli na przykład plik HTML zawiera poniższy znacznik:

```
<applet code="Chart.class" width="100" height="100" name="Chart1">
```

odwołanie do apletu daje poniższa instrukcja:

```
Applet chart1 = getAppletContext().getApplet("Chart1");
```

Do czego można wykorzystać takie odwołanie? Jeśli klasa `Chart` zawiera metodę przyjmującą nowe dane i ponownie rysującą wykres, metodę tę można wywołać, wykonując odpowiednie rzutowanie.

```
((Chart) chart1).setData(3, "Ziemia", 9000);
```

Listę wszystkich apletów znajdujących się na stronie można wyświetlić bez względu na to, czy mają one atrybut `name`, czy nie. Metoda `getApplets` zwraca obiekt typu wyliczeniowego. Poniższa pętla drukuje nazwy klas wszystkich apletów znajdujących się na stronie:

```
Enumeration<Applet> e = getAppletContext().getApplets();
while (e.hasMoreElements())
{
    Applet a = e.nextElement();
    System.out.println(a.getClass().getName());
}
```

Aplety z różnych stron nie mogą się ze sobą komunikować.

13.4.7. Wyświetlanie elementów w przeglądarce

Aplety mają dostęp do dwóch miejsc w oknie przeglądarki: paska stanu i obszaru, na którym wyświetlane są strony. W obu przypadkach używa się metod z klasy `AppletContext`.

Aby wyświetlić tekst w pasku stanu, należy użyć metody `showStatus`. Na przykład:

```
showStatus("Ładowanie danych . . . proszę czekać");
```



Z naszego doświadczenia wynika, że zastosowanie metody `showStatus` jest ograniczone. Przeglądarka również używa paska stanu i w większości przypadków zastępuje dostarczony przez programistę tekst informacją typu *Applet running*. W związku z tym w pasku stanu można wyświetlać komunikaty typu *Ładowanie danych...*, ale nie takie, które są dla użytkownika ważne.

Aby zmusić przeglądarkę do otwarcia nowej strony, należy użyć metody `showDocument`, którą można wywołać na kilka sposobów. Najprostsze wywołanie polega na podaniu jako argumentu adresu URL strony, która ma zostać otwarta:

```
URL u = new URL("http://horstmann.com/index.html");
getAppletContext().showDocument(u);
```

Jednak takie wywołanie może być problematyczne, ponieważ nowa strona zastępuje aktualną, co powoduje zniknięcie apletu. Aby wrócić do apletu, konieczne jest naciśnięcie przycisku *Wstecz* w przeglądarce.

Aby nowa strona została otwarta w nowym oknie, należy metodzie `showDocument` podać dodatkowy parametr (zobacz tabelę 13.2). Łańcuch `_blank` wymusza otwarcie dokumentu w nowym oknie. Co ciekawsze, korzystając z ramek HTML, można podzielić okno na kilka części o uniikatowych nazwach, w których będą wyświetlane dokumenty żądane przez aplet również znajdujący się w jednej z ramek. Przykładowy kod prezentujemy w kolejnym podrozdziale.



Przeglądarka apletów nie wyświetla stron internetowych. Metoda `showDocument` jest przez nią ignorowana.

Tabela 13.2. Metoda `showDocument`

Parametr określający cel	Lokalizacja
<code>_self</code> lub brak	Bieżąca ramka
<code>_parent</code>	Ramka nadrzędna
<code>_top</code>	Ramka najwyższego rzędu
<code>_blank</code>	Nowe okno najwyższego rzędu, bez nazwy
Dowolny inny łańcuch	Ramka o podanej nazwie; jeśli nie ma ramki o takiej nazwie, otwierane jest nowe okno o takiej nazwie

java.applet.Applet **1.2**

- `public AppletContext getAppletContext()`

Tworzy dostęp do środowiska przeglądarki, w której działa aplet. Przy użyciu tych informacji można kontrolować większość przeglądarek.

- `void showStatus(String msg)`

Pokazuje podany łańcuch na pasku stanu przeglądarki internetowej.

java.applet.AppletContext **1.0**

- `Enumeration<Applet> getApplets()`

Zwraca wyliczenie (zobacz rozdział 9.) wszystkich apletów znajdujących się w tym samym środowisku, czyli na jednej stronie internetowej.

- `Applet getApplet(String name)`

Zwraca aplet o podanej nazwie znajdujący się w bieżącym kontekście. Zwraca wartość `null`, jeśli aplet nie istnieje. Przeszukiwana jest tylko bieżąca strona internetowa.

- `void showDocument(URL url)`
- `void showDocument(URL url, String target)`

Otwiera nową stronę internetową w przeglądarce. Pierwsza wersja zastępuje starą stroną nową. Druga otwiera nową stronę w miejscu określonym przez parametr `target` (tabela 13.2).

13.4.8. Piaskownica

Kiedy kod uruchamiany na komputerze jest pobierany ze zdalnego miejsca, sprawą pierwszorzędną staje się zawsze bezpieczeństwo. Wejście na stronę powoduje automatyczne uruchomienie wszystkich znajdujących się na niej apletów. Aplikację Java Web Start może uruchomić jedno kliknięcie. Gdyby kliknięcie odnośnika lub wejście na stronę internetową pozwalało na uruchomienie dowolnego kodu na komputerze użytkownika, przestępcy przeżywaliby złoty wiek wykradania poufnych informacji, danych finansowych i przejmowania komputerów użytkowników w celu rozsyłania spamu.

Technologia Java dysponuje zaawansowanym modelem ochrony, który zapobiega wykorzystywaniu jej do nikczemnych postępów. Model ten został szczegółowo opisany w tomie drugim. **Menedżer zabezpieczeń** (ang. *security manager*) kontroluje dostęp do wszystkich zasobów systemowych. Przy standardowych ustawieniach zezwala tylko na nieszkodliwe operacje. Aby możliwe było wykonywanie dodatkowych operacji, użytkownik musi wyraźnie na to zezwolić apletowi lub aplikacji.

Co pobierany zdalnie kod *może* robić na wszystkich platformach? Zawsze można wyświetlać obrazy, odtwarzać dźwięki, odpowiadać na naciśnięcia przez użytkownika klawiszy i przycisków myszki oraz wysyłać dane wprowadzone przez użytkownika do hosta, z którego został załadowany kod. Taka funkcjonalność wystarcza do zaprezentowania faktów i liczb oraz interakcji z programem edukacyjnym lub grą. Ograniczone środowisko wykonawcze jest często nazywane **piaskownicą** (ang. *sandbox*). Kod działający w piaskownicy nie może nic zmieniać w systemie użytkownika ani go szpiegować.

Programy działające w piaskownicy mają następujące ograniczenia:

- Nie mogą uruchamiać *żadnych* lokalnych programów.
- Nie mogą czytać ani zapisywać plików w lokalnym systemie plików.
- Nie mają dostępu do żadnych informacji o komputerze lokalnym, z wyjątkiem wersji Javy i kilku mało ważnych danych na temat systemu operacyjnego. Kod działający w piaskownicy w szczególności nie ma dostępu do nazwy użytkownika, adresów e-mail itd.
- Programy łaadowane z serwera zdalnego nie mogą się komunikować z żadnym hostem poza tym, z którego pochodzą — serwer taki nosi nazwę **serwera pochodzenia** (ang. *originating host*). Dzięki temu użytkownik jest chroniony przed programami, które mogą wykradać wewnętrzne zasoby.

- Wszystkie wyskakujące okna zawierają komunikat ostrzegawczy. Ma on na celu zabezpieczenie przed pomyleniem ich z oknem aplikacji lokalnej. Istnieją obawy, że nic niepodejrzujący użytkownik wejdzie na stronę internetową, podstępem zostanie zmuszony do uruchomienia zdalnego kodu, a następnie wpisze hasło lub numer karty kredytowej, które zostaną przesłane z powrotem do serwera. We wczesnych wersjach JDK komunikat ten brzmiał bardzo złowieszczo: *Untrusted Java Applet Window*. W każdej kolejnej wersji brzmienie to było nieco łagodzone: *Unauthenticated Java Applet Window* czy *Warning: Java Applet Window*. Obecnie jest to drobny trójkąt, którego większość użytkowników nawet nie zauważy.

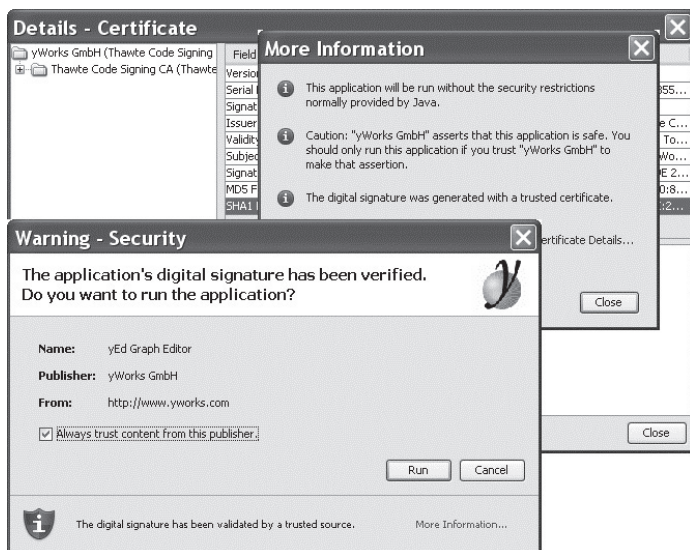
Koncepcja piaskownicy straciła już na znaczeniu. Kiedyś każdy mógł wdrażać kod do uruchamiania w tym ograniczonym środowisku i tylko kod działający poza nim musiał być cyfrowo podpisany. Obecnie podpisu cyfrowego wymagają wszystkie programy uruchamiane za pośrednictwem programu Java Plug-in, niezależnie od tego, czy działają one w piaskownicy, czy nie.

13.4.9. Podpisywanie kodu

Pliki JAR aplikacji Java Web Start muszą być **podpisane cyfrowo**. Podpisany plik JAR ma certyfikat określający tożsamość tego, kto go podpisał. Techniki kryptograficzne dają pewność, że certyfikat taki nie jest sfalszowany, a wszelkie próby zmiany jego zawartości są natychmiast wykrywane.

Wyobraźmy sobie, że pobieramy aplikację utworzoną i podpisaną cyfrowo przez firmę yWorks GmbH z certyfikatem wydanym przez ośrodek certyfikacji Thawte (rysunek 13.6). Odbierając aplikację, mamy pewność, że:

Rysunek 13.6.
Certyfikat
bezpieczeństwa



1. Kod aplikacji nie został zmieniony w żaden sposób od chwili jej podpisania.
2. Podpis rzeczywiście pochodzi od firmy yWorks.

3. Certyfikat naprawdę został wydany przez ośrodek Thawte (mechanizm Java Plug-in potrafi sprawdzać certyfikaty wydane przez Thawte i kilka innych instytucji; istnieje również możliwość zainstalowania innych „podstawowych certyfikatów”).

Jeśli klikniemy odnośnik *More Information*, dowiemy się, że aplikacja zostanie uruchomiona bez zwyczajowych ograniczeń. To, czy zainstalować tę aplikację, czy nie, w dużej mierze zależy od tego, czy ufamy firmie yWorks.

Koszty uzyskania certyfikatu od jednego z obsługiwanych dostawców wynoszą kilkaset dolarów rocznie, a niektóre organizacje wymagają dodatkowo dowodu istnienia firmy lub licencji biznesowej. Kiedyś niektórzy programiści Javy po prostu generowali własne certyfikaty, które wykorzystywali do podpisywania kodu. Oczywiście Java Plug-in nie ma możliwości sprawdzenia, czy są one poprawne. Mimo to kiedyś program ten wyświetlał taki certyfikat użytkownikowi do zatwierdzenia. Mechanizm ten był kompletnie bezużyteczny, ponieważ tylko nieliczni użytkownicy wiedzieli, jaka jest różnica między bezpiecznym i niebezpiecznym certyfikatem. Dlatego też niebezpieczne certyfikaty nie są już obsługiwane.

Obecnie ktoś, kto chce rozprowadzić aplet Java lub aplikację Web Start, nie ma wielkiego wyboru. Musi zdobyć certyfikat od odpowiedniej organizacji uznawanej przez Java Plug-in i podpisywać przy jego użyciu swoje pliki JAR.

Wiele firm nawiązuje stałą współpracę z organizacjami certyfikującymi, więc jeśli pracujesz w jednej z nich, to wystarczy zamówić certyfikat do podpisania swojego kodu. Jeżeli nie, to warto się rozejrzeć, ponieważ ceny bywają bardzo zróżnicowane i niektórzy dostawcy oferują nieco dogodniejsze warunki odbiorcom indywidualnym.

Wraz z certyfikatem otrzymasz instrukcję, jak go zainstalować w magazynie kluczy Javy — specjalnym pliku chronionym hasłem, z którego certyfikat może zostać pobrany w trakcie procesu podpisywania. Magazyn kluczy i hasło należy przechowywać w bezpiecznym miejscu.

Następnie trzeba zdecydować, jakie uprawnienia będą potrzebne. Do wyboru są uprawnienia piaskownicy i pełne. Wyboru dokonuje się w pliku manifestu (zobacz sekcję 13.1.2, „Manifest”).

Dodaj instrukcję `Permissions: sandbox` lub `Permissions: all-permissions`, np.:

```
Manifest-Version: 1.0
Permissions: all-permissions
```

Użyj narzędzia `jar`:

```
jar cvfm MyApplet.jar manifest.mf mypackage/*.class
```

Element `applet` w pliku HTML powinien mieć zdefiniowany atrybut `archive="MyApplet.jar"`.

Na koniec należy podpisać plik JAR. Oto odpowiednie polecenie:

```
jarsigner -keystore plikmagazynukluczy -tsa URLznacznikaczasu MyApplet.jar aliasklucza
```

Adres URL do pobrania znacznika czasu otrzymuje się od wydawcy certyfikatu. Alias klucza też został przydzielony przez wydawcę certyfikatu. Można go sprawdzić za pomocą poniższego polecenia:

```
keytool -keystore plikzkkluczami -list
```

Ewentualnie alias klucza można zmienić za pomocą opcji `-changealias` polecenia `keytool`. (Więcej informacji na temat narzędzia `keytool` znajduje się w rozdziale 9. drugiego tomu).

Teraz pozostaje już tylko wysłanie podpisanego pliku JAR i pliku HTML z elementem `applet` na serwer.



Uprawnienia aplikacji Java można dokładnie kontrolować, o czym szerzej piszemy w rozdziale 12. drugiego tomu. System ten jednak nigdy nie był używany w sposób mający jakiegokolwiek znaczenie dla zwykłych użytkowników. Program Java Plug-in udostępnia tylko dwa poziomy bezpieczeństwa — piaskownicę i wszystkie uprawnienia.

13.5. Java Web Start

Java Web Start jest technologią uruchamiania aplikacji bezpośrednio z internetu. Aplikacje Java Web Start mają następujące cechy:

- Są zazwyczaj dostarczane za pośrednictwem przeglądarki. Po pobraniu aplikacja Java Web Start może być uruchamiana bez użycia przeglądarki.
- Nie rezydują w oknie przeglądarki. Aplikacja działa we własnym oknie ramowym, poza przeglądarką.
- Nie wykorzystują implementacji Javy przeglądarki. Przeglądarka uruchamia tylko zewnętrzną aplikację, podobnie jak w przypadku innych programów, takich jak Adobe Acrobat lub Real Audio.
- Aplikacjom podpisanym cyfrowo można nadawać dowolne prawa dostępu. Niepodpisane aplikacje działają w **piaskownicy** (ang. *sandbox*), która nie zezwala na potencjalnie niebezpieczne operacje.

13.5.1. Wdrażanie aplikacji Java Web Start

Przygotowywanie aplikacji do dostarczania za pośrednictwem mechanizmu Java Web Start polega na spakowaniu jej do jednego lub większej liczby plików JAR. Następnie należy utworzyć plik deskryptora w formacie JNLP (ang. *Java Network Launch Protocol*). Pliki umieszczamy na serwerze.

Dodatkowo serwer sieciowy musi raportować typ MIME `application/x-java-jnlp-file` dla plików z rozszerzeniem `.jnlp` (typ MIME umożliwia przeglądarkom podjęcie decyzji, który program pomocniczy uruchomić). Szczegółowych informacji na ten temat należy szukać w dokumentacji serwera.



Aby wypróbować mechanizm Java Web Start, zainstaluj Tomcat dostępny na stronie <http://jakarta.apache.org/tomcat>. Jest to kontener na serwlety i strony JSP, ale serwuje też strony internetowe. Jest wstępnie skonfigurowany do serwowania poprawnego typu MIME dla plików JNLP. W dalszej części tej sekcji zakładamy, że używany jest serwer Tomcat.

Wypróbujemy mechanizm Java Web Start na kalkulatorze z rozdziału 12. Wykonaj następujące czynności:

1. Skompiluj program.

```
javac -classpath ./path/to/javaws.jar webstart/*.java
```

2. Utwórz plik JAR za pomocą poniższego polecenia:

```
jar cvfe Calculator.jar webstart.Calculator webstart/*.class
```

3. Przygotuj plik rozruchowy *Calculator.jnlp* o następującej treści:

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0+" codebase="http://localhost:8080/calculator/"
href="Calculator.jnlp">
  <information>
    <title>Przykładowy kalkulator</title>
    <vendor>Cay S. Horstmann</vendor>
    <description>Kalkulator</description>
    <offline-allowed/>
  </information>
  <resources>
    <java version="1.6.0+"/>
    <jar href="Calculator.jar"/>
  </resources>
  <application-desc/>
</jnlp>
```

(Należy pamiętać, że numer wersji to 1.6.0, a nie 6.0).

Format pliku JNLP jest bardzo prosty. Jego pełna specyfikacja znajduje się na stronie <http://www.oracle.com/technetwork/java/javase/javawebstart>.

4. Utwórz katalog *tomcat/webapps/calculator*, z którego będzie serwowana aplikacja. W ścieżce tej *tomcat* to katalog główny instalacji Tomcata. Utwórz podkatalog *tomcat/webapps/calculator/WEB-INF* i umieść w nim poniższy plik *web.xml*:

```
<?xml version="1.0" encoding="utf-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_5.xsd">
</web-app>
```

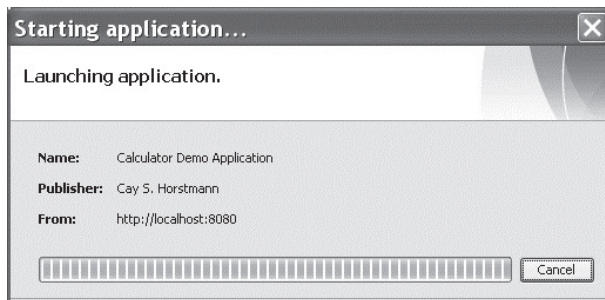
5. Umieść pliki JAR i uruchomieniowy w katalogu *tomcat/webapps/calculator*.

6. Dodaj adres *http://localhost:8080* do listy zaufanych miejsc w panelu kontrolnym Javy, postępując zgodnie z instrukcjami zawartymi w podrozdziale 2.5, „Tworzenie i uruchamianie apletów”. Ewentualnie możesz zamiast tego podpisać plik JAR zgodnie z instrukcjami zamieszczonymi w sekcji 13.4.9, „Podpisywanie kodu”.

7. Uruchom serwer Tomcat, wykonując skrypt rozruchowy znajdujący się w katalogu *tomcat/bin*.

8. Wpisz w przeglądarce adres pliku JNLP. Na przykład w przypadku użycia serwera Tomcat należy wpisać adres *http://localhost:8080/calculator/Calculator.jnlp*. Jeżeli przeglądarka została skonfigurowana pod kątem pracy z Java Web Start, powinno się pojawić okno uruchamiania Java Web Start (rysunek 13.7).

Rysunek 13.7.
Uruchamianie
aplikacji Java
Web Start



Jeśli przeglądarka nie rozpoznaje plików JNLP, może zaproponować powiązanie ich z jakąś aplikacją. W takim przypadku wybierz *jdk/bin/javaws*. W przeciwnym razie musisz dowiedzieć się, jak powiązać typ MIME *application/x-java-jnlp-file* z aplikacją *javaws*. Ewentualnie możesz jeszcze raz zainstalować pakiet JDK, który powinien to zrobić automatycznie.

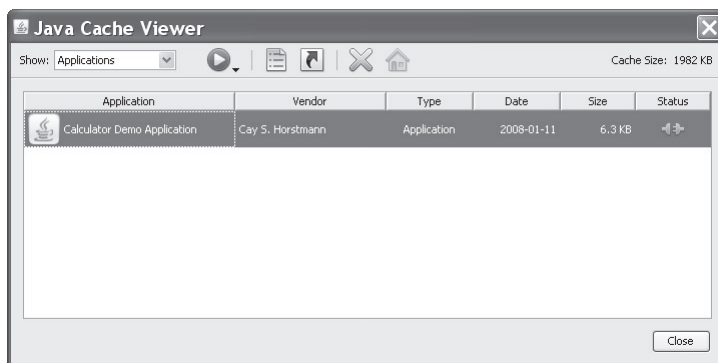
9. Chwilę później powinien się pojawić kalkulator z informacją, że jest to aplikacja Javy (rysunek 13.8).

Rysunek 13.8.
Kalkulator otwarty
za pośrednictwem
Java Web Start



10. Kiedy następnym razem spróbujesz uzyskać dostęp do pliku JNLP, program będzie pobierany z pamięci podręcznej. Zawartość tej pamięci można obejrzeć za pomocą panelu kontrolnego w postaci wtyczki Javy (rysunek 13.9). W systemie Windows wtyczki tej należy szukać w *Panelu sterowania*. W systemie Linux należy wykonać polecenie *jdk/jre/bin/ControlPanel*.

Rysunek 13.9.
Aplikacje
w pamięci
podręcznej





Aby nie uruchamiać serwera podczas testowania konfiguracji JNLP, można tymczasowo nadpisać adres URL codebase w pliku JNLP za pomocą poniższego polecenia:

```
javaws -codebase file:///katalogProgramu plikJNL
```

Na przykład w systemie Unix polecenie to można wydać w katalogu zawierającym plik JNLP:

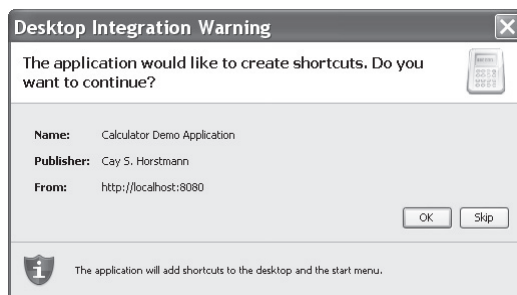
```
javaws -codebase file:///`pwd` Calculator.jnlp
```

Oczywiście nie chcemy nakłaniać użytkowników do uruchamiania przeglądarki pamięci podręcznej za każdym razem, kiedy chcą uruchomić naszą aplikację. Można sprawić, aby instalator proponował utworzenie skrótów w menu *Start* i na pulpicie. W tym celu należy dodać poniższy kod do pliku JNLP:

```
<shortcut>
  <desktop/>
  <menu submenu="Akcesoria"/>
</shortcut>
```

Podczas pierwszego pobierania aplikacji zostanie wyświetlone ostrzeżenie o dodawaniu skrótów (rysunek 13.10).

Rysunek 13.10.
Ostrzeżenie
o integracji



Dodatkowo powinno się dodać ikonę dla skrótu i ekranu uruchomieniowego. Firma Sun zaleca stosowanie ikon o rozmiarach 32×32 i 64×64 piksele. Pliki ikon należy umieścić na serwerze razem z plikami JAR i JNLP. Poniższy kod należy dodać do sekcji *information* pliku JNLP:

```
<icon href="calc_icon32.png" width="32" height="32" />
<icon href="calc_icon64.png" width="64" height="64" />
```

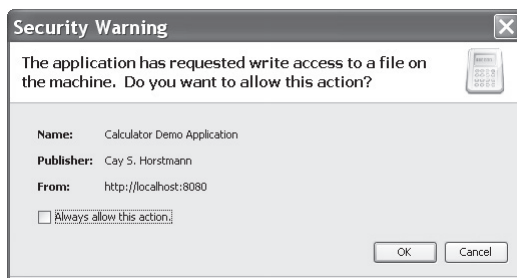
Należy pamiętać, że ikony te nie są związane z ikoną aplikacji. Aby wstawić ikonę dla aplikacji, należy dodać odrębny plik ikony do pliku JAR i wywołać metodę *IconImage* na rzecz klasy ramowej (zobacz listing 13.1).

13.5.2. API JNLP

Technologia Java Web Start ma tę przewagę nad apletami, że dla aplikacji działających w piaskownicy posiada API udostępniające przydatne usługi. Za jego pomocą aplikacje działające w piaskownicy mają dostęp do zasobów lokalnych przy zachowaniu odpowiedniego poziomu bezpieczeństwa. Istnieją na przykład usługi polegające na wysyłaniu i zapisywaniu plików.

Aplikacja nie ma dostępu do systemu plików i nie może określać nazw plików. W zamian wyświetlane jest okno dialogowe wyboru plików, w którym użytkownik programu wybiera plik. Przed wyświetleniem tego okna pojawia się ostrzeżenie i użytkownik musi wyrazić zgodę na kontynuowanie (rysunek 13.11). Ponadto opisywane API nie daje w rzeczywistości programowi dostępu do obiektu typu `File`. W szczególności aplikacja nie może sprawdzić lokalizacji pliku. W ten sposób programista zyskuje narzędzia do implementacji akcji otwierania i zapisywania plików, ale tak dużo informacji systemowych, jak to tylko możliwe, jest ukrytych przed niezaufanymi aplikacjami.

Rysunek 13.11.
Ostrzeżenie
Java Web Start



To API udostępnia następujące usługi:

- wysyłanie i zapisywanie plików,
- dostęp do schowka,
- drukowanie,
- pobieranie plików,
- wyświetlanie dokumentów w domyślnej przeglądarce,
- zapisywanie i odczytywanie stałych danych konfiguracyjnych,
- pilnowanie, aby był uruchomiony tylko jeden egzemplarz programu.

Dostęp do usługi uzyskuje się za pomocą metody `ServiceManager`:

```
FileSaveService service = (FileSaveService)
    ServiceManager.lookup("javax.jnlp.FileSaveService");
```

Powyższa instrukcja spowoduje wyjątek `UnavailableServiceException`, jeśli usługa jest niedostępna.



Aby móc kompilować programy wykorzystujące API JNLP, należy umieścić plik *javaws.jar* na ścieżce klas. Plik ten znajduje się w podkatalogu *jre/lib* w katalogu JDK.

Omówimy najbardziej przydatne usługi JNLP. Aby zapisać plik, trzeba podać ścieżkę startową i rozszerzenia plików wyświetlanych w oknie dialogowym, dane do zapisania oraz sugerowaną nazwę pliku. Na przykład:

```
service.saveFileDialog(".", new String[] { "txt" }, data, "calc.txt");
```


Dane muszą być dostarczone w strumieniu `InputStream`, co nie zawsze jest łatwe. W programie na listingu 13.6 przyjęto następującą strategię działania:

1. Utworzenie strumienia `ByteArrayOutputStream` przechowującego bajty, które mają być zapisane.
2. Utworzenie strumienia `PrintStream` wysyłającego swoje dane do strumienia `ByteArrayOutputStream`.
3. Wydrukowanie informacji, które mają być zapisane, do strumienia `PrintStream`.
4. Utworzenie strumienia `ByteArrayInputStream` odczytującego zapisane bajty.
5. Przekazanie strumienia do metody `saveFileDialog`.

Więcej na temat strumieni piszemy w rozdziale 1. drugiego tomu. Teraz wystarczy tylko pobieżnie przejrzeć przykładowy program.

Do odczytu danych z pliku służy klasa `FileOpenService`. Jej metoda `openFileDialog` odbiera sugerowaną ścieżkę początkową i rozszerzenia plików wyświetlanych w oknie dialogowym i zwraca obiekt typu `FileContents`. Następnie można za pomocą metod `getInputStream` i `getOutputStream` odczytać i zapisać dane do pliku. Jeśli użytkownik nie wybrał pliku, metoda `openFileDialog` zwraca wartość `null`.

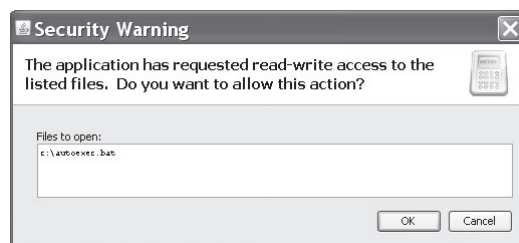
```
FileOpenService service = (FileOpenService)
ServiceManager.lookup("javax.jnlp.FileOpenService");
FileContents contents = service.openFileDialog(".", new String[] { "txt" });
if (contents != null)
{
    InputStream in = contents.getInputStream();
    . . .
}
```

Pamiętajmy, że aplikacja nie zna nazwy ani lokalizacji pliku. Aby otworzyć określony plik, można użyć klasy `ExtendedService`:

```
ExtendedService service = (ExtendedService)
ServiceManager.lookup("javax.jnlp.ExtendedService");
FileContents contents = service.openFile(new File("c:\\autoexec.bat"));
if (contents != null)
{
    OutputStream out = contents.getOutputStream();
    . . .
}
```

Użytkownik programu musi się zgodzić na dostęp do tego pliku (rysunek 13.12).

Rysunek 13.12.
Ostrzeżenie
o dostępie
do pliku



Do wyświetlenia dokumentu w domyślnej przeglądarce należy użyć interfejsu `BasicService`. Zauważmy, że niektóre systemy mogą nie mieć domyślnej przeglądarki.

```
BasicService service = (BasicService) ServiceManager.lookup("javax.jnlp.BasicService");
if (service.isWebBrowserSupported())
    service.showDocument(url);
else . . .
```

Podstawowy interfejs `PersistenceService` pozwala aplikacji zapisywać niewielkie ilości danych konfiguracyjnych i odczytywać je po jej ponownym uruchomieniu. Mechanizm ten przypomina nieco pliki cookie HTTP. Ten stały magazyn jako klucze wykorzystuje adresy URL. Adres URL nie musi prowadzić do istniejącego zasobu sieciowego. Są one w tym przypadku wykorzystywane jako wygodny sposób nazewnictwa hierarchicznego. Dla każdego klucza URL aplikacja może zapisać dowolne dane binarne (rozmiar jednego bloku danych w magazynie może być ograniczony).

Odseparowanie od siebie poszczególnych aplikacji jest możliwe dzięki temu, że każdy program może używać tylko takich kluczy, które zaczynają się od określonego podstawowego łańcucha (zgodnie z informacjami w pliku JNLP). Jeśli na przykład aplikacja zostanie pobrana ze strony `http://myserver.com/apps`, może używać tylko kluczy w formacie `http://myserver.com/apps/subkey1/subkey2/...`. Próby dostępu do innych kluczy zakończą się niepowodzeniem.

Aplikacja może sprawdzić podstawę swojego klucza URL za pomocą metody `getCodeBase` z klasy `BasicService`.

Do tworzenia kluczy służy metoda `create` z interfejsu `PersistenceService`.

```
URL url = new URL(codeBase, "mykey");
service.create(url, maxSize);
```

Aby uzyskać informacje związane z określonym kluczem, należy wywołać metodę `get`. Zwraca ona obiekt typu `FileContents`, za pośrednictwem którego można odczytywać i zapisywać dane kluczy. Na przykład:

```
FileContents contents = service.get(url);
InputStream in = contents.getInputStream();
OutputStream out = contents.getOutputStream(true); // true = nadpisz
```

Niestety nie ma dobrego sposobu na sprawdzenie, czy określony klucz już istnieje, czy trzeba go utworzyć. Można wywołać metodę `get`. Jeśli klucz nie istnieje, zostanie spowodowany wyjątek `FileNotFoundException`.

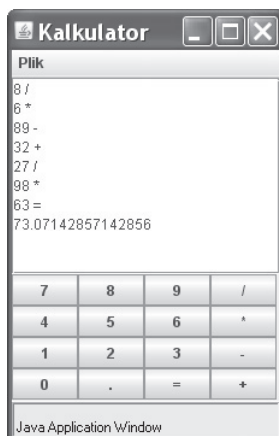


Aplikacje Java Web Start i aplety mogą drukować, wykorzystując normalne API drukowania. Pojawia się tylko okienko, w którym użytkownik jest proszony o zezwolenie na dostęp do drukarki. Więcej informacji na temat API drukowania znajduje się w rozdziale 7. drugiego tomu.

Program na listingu 13.6 jest prostym rozszerzeniem wcześniej utworzonego kalkulatora. Ma on wirtualną taśmę, która rejestruje wszystkie obliczenia. Można zapisywać i ładować historię obliczeń. Aplikacja pozwala na ustawienie tytułu ramki, co służy demonstracji trwałego magazynu. Przy ponownym uruchomieniu aplikacja pobierze wybrany przez użytkownika tytuł z trwałego magazynu (rysunek 13.13).

Rysunek 13.13.

Aplikacja
WebStartCalculator

**Listing 13.6.** webstart/CalculatorFrame.java

```
package webstart;

import java.io.BufferedReader;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.PrintStream;
import java.net.MalformedURLException;
import java.net.URL;

import javax.jnlp.BasicService;
import javax.jnlp.FileContents;
import javax.jnlp.FileOpenService;
import javax.jnlp.FileSaveService;
import javax.jnlp.PersistenceService;
import javax.jnlp.ServiceManager;
import javax.jnlp.UnavailableServiceException;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;

/**
 * Ramka z kalkulatorem i menu do ładowania oraz zapisywania historii obliczeń
 */
public class CalculatorFrame extends JFrame
{
    private CalculatorPanel panel;

    public CalculatorFrame()
    {
        setTitle();
    }
}
```

```

panel = new CalculatorPanel();
add(panel);

JMenu fileMenu = new JMenu("Plik");
JMenuBar menuBar = new JMenuBar();
menuBar.add(fileMenu);
setJMenuBar(menuBar);

JMenuItem openItem = fileMenu.add("Otwórz");
openItem.addActionListener(event -> open());
JMenuItem saveItem = fileMenu.add("Zapisz");
saveItem.addActionListener(event -> save());

pack();
}

/**
 * Pobiera tytuł z magazynu trwałego lub prosi użytkownika o podanie tytułu, jeśli
 * nie ma wcześniejszego wpisu.
 */
public void setTitle()
{
    try
    {
        String title = null;

        BasicService basic = (BasicService)
            ↪ServiceManager.lookup("javax.jnlp.BasicService");
        URL codeBase = basic.getCodeBase();

        PersistenceService service = (PersistenceService) ServiceManager
            .lookup("javax.jnlp.PersistenceService");
        URL key = new URL(codeBase, "title");

        try
        {
            FileContents contents = service.get(key);
            InputStream in = contents.getInputStream();
            BufferedReader reader = new BufferedReader(new InputStreamReader(in));
            title = reader.readLine();
        }
        catch (FileNotFoundException e)
        {
            title = JOptionPane.showInputDialog("Podaj tytuł ramki:");
            if (title == null) return;

            service.create(key, 100);
            FileContents contents = service.get(key);
            OutputStream out = contents.getOutputStream(true);
            PrintStream printOut = new PrintStream(out);
            printOut.print(title);
        }
        setTitle(title);
    }
    catch (UnavailableServiceException e)
    {
        JOptionPane.showMessageDialog(this, e);
    }
}

```

```

    }

    /**
    * Otwiera plik historii i aktualizuje zawartość wyświetlacza.
    */
    public void open()
    {
        try
        {
            FileOpenService service = (FileOpenService) ServiceManager
                .lookup("javax.jnlp.FileOpenService");
            FileContents contents = service.openFileDialog(".", new String[] { "txt" });

            JOptionPane.showMessageDialog(this, contents.getName());
            if (contents != null)
            {
                InputStream in = contents.getInputStream();
                BufferedReader reader = new BufferedReader(new InputStreamReader(in));
                String line;
                while ((line = reader.readLine()) != null)
                {
                    panel.append(line);
                    panel.append("\n");
                }
            }
        }
        catch (UnavailableServiceException e)
        {
            JOptionPane.showMessageDialog(this, e);
        }
        catch (IOException e)
        {
            JOptionPane.showMessageDialog(this, e);
        }
    }

    /**
    * Zapisuje historię kalkulatora w pliku.
    */
    public void save()
    {
        try
        {
            ByteArrayOutputStream out = new ByteArrayOutputStream();
            PrintStream printOut = new PrintStream(out);
            printOut.print(panel.getText());
            InputStream data = new ByteArrayInputStream(out.toByteArray());
            FileSaveService service = (FileSaveService) ServiceManager
                .lookup("javax.jnlp.FileSaveService");
            service.saveFileDialog(".", new String[] { "txt" }, data, "calc.txt");
        }
        catch (UnavailableServiceException e)
        {
            JOptionPane.showMessageDialog(this, e);
        }
        catch (IOException e)
        {

```

```
        JOptionPane.showMessageDialog(this, e);
    }
}
```

javax.jnlp.ServiceManager

- `static String[] getServiceNames()`
Zwraca nazwy wszystkich dostępnych usług.
- `static Object lookup(string name)`
Zwraca usługę o podanej nazwie.

javax.jnlp.BasicService

- `URL getCodeBase()`
Zwraca katalog zawierający kod aplikacji.
- `boolean isWebBrowserSupported()`
Zwraca wartość `true`, jeśli środowisko Web Start może uruchomić przeglądarkę.
- `boolean showDocument(URL url)`
Podejmuje próbę pokazania danego adresu URL w przeglądarce. Zwraca wartość `true`, jeśli żądanie kończy się powodzeniem.

javax.jnlp.FileContents

- `InputStream getInputStream()`
Zwraca strumień wejściowy do odczytu zawartości pliku.
- `OutputStream getOutputStream(boolean overwrite)`
Zwraca strumień wyjściowy do zapisu do pliku. Jeśli parametr `overwrite` ma wartość `true`, aktualna treść pliku jest nadpisywana.
- `String getName()`
Zwraca nazwę pliku (nie pełną ścieżkę katalogową).
- `boolean canRead()`
- `boolean canWrite()`
Zwraca wartość `true`, jeśli dany plik nadaje się do odczytu lub zapisu.

javax.jnlp.FileOpenService

- `FileContents openFileDialog(String pathHint, String[] extensions)`
- `FileContents[] openMultiFileDialog(String pathHint, String[] extensions)`
Wyświetla ostrzeżenie dla użytkownika i okno wyboru pliku. Zwraca deskryptory treści pliku lub plików wybranych przez użytkownika bądź wartość `null`, jeśli użytkownik nie wybrał żadnego pliku.

```
javax.jnlp.FileSaveService
```

- `FileContents saveFileDialog(String pathHint, String[] extensions, InputStream data, String nameHint)`
- `FileContents saveAsFileDialog(String pathHint, String[] extensions, FileContents data)`

Wyświetla ostrzeżenie dla użytkownika i okno wyboru pliku. Zapisuje dane i zwraca deskryptory treści pliku lub plików wybranych przez użytkownika bądź wartość `null`, jeśli użytkownik nie wybrał żadnego pliku.

```
javax.jnlp.PersistenceService
```

- `long create(URL key, long maxsize)`
Zapisuje dany klucz w pamięci trwałej. Zwraca maksymalny rozmiar przyznawany przez pamięć trwałą.
- `void delete(URL key)`
Usuwa dany klucz.
- `String[] getNames(URL url)`
Zwraca względne nazwy wszystkich kluczy, które zaczynają się od danego adresu URL.
- `FileContents get(URL key)`
Tworzy deskryptor treści, za pośrednictwem którego można modyfikować dane związane z danym kluczem. Jeśli dla danego klucza nie istnieje żaden wpis, zgłaszany jest wyjątek `FileNotFoundException`.

Na tym kończymy opis technik przygotowywania aplikacji w Javie do użytku. W ostatnim rozdziale zajmiemy się bardzo ważnym tematem współbieżności.

Skorowidz

A

- absolute positioning, 643
- abstrakcja, 209
- accelerators, 613
- access modifier, 56
- accessor method, 139
- accessory component, 673
- action map, 550
- adapter class, 543
- adnotacje, 392
- adres URL, 726, 738
- agregacja, 133
- akcelerator, 612, 613
- akcesorium, 673
- akcja bariery, 829
- akcje, 546
- aktualizacje, 37
- aktywność komponentu, 548
- aktywowanie elementów menu, 614
- algorytm, 130, 469, 474
 - quick sort, 120, 471
 - sortowanie, 470, 471
 - tasowanie, 470
 - tworzenie, 478
 - wyszukiwanie binarne, 473
 - znajdowanie liczb pierwszych, 483
- analiza
 - danych ze śledzenia stosu, 345
 - funkcjonalności klasy, 246
 - obiektów w czasie działania programu, 251
- animowane gify, 725
- anonimowe klasy wewnętrzne, 302, 313
- API, Application Programming Interface, 13, 17
- API JNLP, 735
- API Preferences, 706
 - dostęp do tablicy, 707
 - dostęp do węzła drzewa, 707
 - repozytorium, 706
 - zapis danych, 707
- aplety, 28, 693, 714
 - dostęp do JavaScriptu, 720
 - implementacja, 715
 - init(), 718
 - JApplet, 715
 - kommunikacja między apletami, 720, 727
 - obrazy, 725
 - otwieranie strony, 728
 - parametry, 720
 - pasek stanu, 727
 - pliki audio, 725
 - przeglądarka, 714, 716
 - Swing, 715
 - środowisko działania, 726
 - tworzenie, 48, 715
 - uruchamianie, 48, 716
 - viewer, 716
 - wyświetlanie elementów w przeglądarce, 727
- aplikacje graficzne, 46
- architektura, 25
 - model-widok-kontroler, 570
- argumenty, 59
- ascender, 519
- ascent, 519
- asercje, 350
 - dokumentacja założeń, 353
 - sprawdzanie parametrów, 352
 - stosowanie, 352
 - warunek wstępny, 353
 - włączanie, 352
 - wyłączanie, 352
- asocjacja, 133
- atrybuty znacznika applet, 719
- autoboxing, 31, 235, 236
- automatyczne
 - opakowywanie, 236
 - usuwanie nieużytków, 22
- autowrapping, 236
- AWT, Abstract Window Toolkit, 488

B

- bariery, 829
 - akcja, 829
 - cykliczne, 829
 - złamanie, 829
- base class, 194
- baseline, 519
- bazowy katalog drzewa pakietu, 182
- bezpieczeństwo, 24, 34
- biblioteka
 - Java2D, 506
 - refleksyjna, 242
- bitowa
 - alternatywa, 73
 - koniunkcja, 73
 - negacja, 73
- blocking queue, 796
- blok, 57, 97
 - inicjujący, 170
 - synchronizowany, 781
 - try, 245
 - try-catch, 335
 - try-finally, 341, 342
- blokad, 771
 - jawne, 778
 - odczyt-zapis, 793
 - wewnętrzne, 778
 - wielowejściowe, 771
- blokowanie po stronie klienta, 782
- błędy, 24, 328
 - dane wejściowe, 328
 - debugowanie, 372
 - kod źródłowy, 328
 - ograniczenia fizyczne, 328
 - pomyłka o jeden, 470
 - przekroczenie zakresu liczby, 60
 - urządzenia, 328
- BMP, Basic Multilingual Plane, 64
- boolean, 64
- border layout manager, 574
- bridge method, 390
- bucket, 441
- budowanie łańcuchów wyjątków, 339

C

- call by name, 159
- call by reference, 159
- call by value, 159
- callback, 278
- camel case, 56
- casting, 71, 206
- certyfikaty, 731
- checkbox, 587
- checked exception, 243, 330

- child class, 194
- chwytnie typu wieloznacznego, 408
- ciało metody, 58
- class field, 154
- class loader, 320, 352
- client-side locking, 782
- code, 718
- code planes, 64
- code point, 64
- code units, 64
- combo box, 597
- content pane, 501
- corejava, 698
- coupling, 133
- covariant return types, 204
- cykliczne bariery, 829
- czcionki, 517
 - Font, 518
 - interlinia, 519
 - linia bazowa, 519
 - logiczne nazwy, 517
 - nazwa, 517
 - odzworowanie nazw logicznych, 518
 - PostScript type 1, 518
 - rodzina, 517
 - styl, 518
 - TrueType, 518
 - wydłużenie, 519
 - wysokość, 519
- czytanie danych, 88

D

- dane
 - XML, 32
 - ze śledzenia stosu, 345
- deadlock, 774, 788
- debuger, 373
- debugowanie, 372
- deep copying, 284
- default package, 177
- definicja
 - klasy, 142
 - klasy uogólnionej, 382
 - stałej klasowej, 67
 - zmiennej, 66
- deklaracja
 - wyjątków kontrolowanych, 331
 - zmiennej tablicowej, 115
 - zmiennych, 65
- dekrementacja, 72
- delegacja, 258
 - zdarzeń, 531
- demony, 762
- derived class, 194
- descender, 519
- descent, 519

device context, 502
 dezaktywacja elementu, 561
 menu, 614
 diagramy klas, 134
 document-modal dialog, 660
 dodawanie
 akcji, 826
 klasy do pakietu, 177
 dokumentacja, 40
 API, 81, 84
 założeń, 353
 dokumenty JSR, 379
 dostęp
 chroniony, 214
 do apletu z poziomu JavaScriptu, 720
 do elementów listy tablicowej, 231
 do plików, 96
 do pól, 149
 do zasobów lokalnych, 735
 double, 61
 doubly linked list, 432
 drukowanie, 30
 drzewa, 444
 czerwono-czarne, 444
 duże liczby, 60
 dymki, 620
 dynamic binding, 199
 dynamiczność, 27
 dziedziczenie, 131, 133, 193, 261, 380
 dostęp chroniony, 214
 hierarchia, 201
 interfejs, 272
 klasy, 495
 klasy abstrakcyjne, 209
 klasy finalne, 205
 metody finalne, 205
 metody przesłaniające, 195
 nadklasy, 194
 Object, 215
 podklasy, 194
 polimorfizm, 199, 201
 porównywanie obiektów, 217
 przesłanianie metod, 197
 rzutowanie, 206
 super, 196
 typy uogólnione, 402
 wielokrotne, 274
 dzienniki, 355, 364
 filtry, 363
 formatery, 364
 Handler, 360
 konfiguracja menedżera dzienników, 358
 lokalizacja, 359
 poziomy ważności komunikatów, 356
 rotacja plików, 361
 śledzenie przepływu wykonywania, 357
 techniki zapisu, 356
 zapis, 355

E

edycja kodu źródłowego, 45
 egzemplarz klasy, 131
 elementy menu, 606
 eliminowanie wywołań funkcji, 27
 epoka, 138
 etykiety, 582
 event dispatch thread, 493, 795
 event listener, 530
 event object, 530
 event procedure, 529
 event sources, 530
 exit code, 58
 explicit parameter, 148
 external padding, 627

F

factory method, 156
 false, 64
 field accessor, 149
 figura
 2W, 505
 geometryczne, 506
 file chooser, 671
 filtr plików, 670
 filtry, 363
 float, 61
 flow layout manager, 572
 format JNLP, 732
 formatery, 364
 formatowanie danych wyjściowych, 91
 frame, 492
 funkcje
 czysto wirtualne, 210
 matematyczne, 68

G

garbage collecting, 22
 Garbage Collector, 458
 GBC, 628
 generic class, 381, 382
 generic programming, 380
 generic types, 31
 generowanie dokumentacji javadoc, 189
 generyczne listy tablicowe, 228
 dostęp do elementów, 231
 glass pane, 501
 głęboka kopia, 284
 GPL, 33
 graficzny interfejs użytkownika, GUI, 30, 487, 751
 grafika, 487
 czcionki, 517
 kolory, 513
 obrazy, 525

graphics context, 502
group layout manager, 624
grupy przycisków radiowych, 590
GTK, 490
GUI, 30, 487, 751

H

hash code, 220, 441
hash table, 221, 602
heap, 119, 450
height, 497
hermetyzacja, 131, 149
hierarchia
 dziedziczenia, 201
 interfejsów, 272
 wyjątków, 329, 349
 zdarzeń w bibliotece AWT, 560
historia Javy, 29
HSB, 679
HTML, 31

I

IDE, 43
identyfikacja klas, 132
IEEE 754, 62
IFC, Internet Foundation Classes, 488
ikony w elementach menu, 609
immutable class, 153
implementacja
 apletów, 715
 interfejsu, 265, 267
implicit parameter, 148
import, 89, 175
 klas, 175
 statyczny, 177
informacje o typach
 czasu wykonywania, 242
 generycznych w maszynie wirtualnej, 412
inheritance, 193
inheritance chain, 201
inheritance hierarchy, 201
inicjalizacja
 pól, 167
 pól wartościami domyślnymi, 166
 tablic, 117
 zmiennych, 66
inkrementacja, 72
inline method, 149
inlining, 27, 206
inner class, 265, 302
input dialog, 649
input maps, 549
instalacja
 dokumentacji, 40
 Java Development Kit, 36

instrukcja
 switch, 108
 goto, 110
instrukcje
 sterujące, 97
 warunkowe, 98
interfejs, 214, 258, 266, 428
 AdjustmentListener, 562
 Action, 546, 607, 613
 ActionListener, 278, 313, 531, 546
 AppletContext, 720, 726–728
 BasicService, 738, 742
 BlockingDeque, 801
 BlockingQueue, 800
 ButtonModel, 571, 591, 593
 Callable, 810
 ChangeListener, 601
 Cloneable, 284
 Callable, 810, 814
 Collection, 422, 426–429, 436
 Comparable, 266, 386, 450
 Comparator, 281, 301
 Condition, 777, 779, 792
 Delayed, 800
 Deque, 448, 449
 Enumerable, 419
 Enumeration, 424, 480
 ExecutorService, 816, 819, 821
 FileFilter, 671, 678
 Filter, 372
 FocusListener, 562
 Formattable, 92
 Future, 810–814, 839
 GenericArrayType, 413, 417
 InvocationHandler, 320, 324
 ItemListener, 562
 ItemSelectable, 591
 Iterable, 117, 423
 Iterator, 303, 424
 Iterator<E>, 428
 KeyListener, 562
 LayoutManager, 644, 647
 ListIterator, 433, 436
 ListIterator<E>, 439
 Lock, 769, 772, 777, 792
 Map, 429, 453
 MouseListener, 615
 MouseListener, 554, 556, 562
 MouseEvent, 554–556, 562
 MouseWheelListener, 562, 563
 NavigableSet, 430, 445, 448, 468
 NavigableMap, 430, 469
 ParametrizedType, 413, 417
 PersistenceService, 738, 743
 PersistentService, 738
 PropertyChangeListener, 674
 Queue, 420, 448
 RandomAccess, 429

Runnable, 755, 810, 816, 832
 ScheduledExecutorService, 820
 Set, 430
 Shape, 506
 SortedMap, 430, 454–457, 469
 SortedSet, 430, 447
 Thread.UncaughtExceptionHandler, 763
 TypeVariable, 413, 417
 UncaughtExceptionHandler, 763, 764
 WildcardType, 413, 417
 WindowFocusListener, 562, 563
 WindowListener, 542, 545, 562
 WindowStateListener, 545, 563
 interfejsy
 definicja, 266
 dziedziczenie, 272
 hierarchia, 272
 funkcyjne, 292
 implementacja, 265, 267
 klasy abstrakcyjne, 273
 metody, 273
 nasłuchu, 530
 programowania aplikacji, 13
 sprzężenie zwrotne, 278
 zmiennne, 272
 znacznikowe, 284
 interlinia, 519
 internal padding, 627
 internet, 28
 interpreter, 26
 interrupted state, 755
 invocation handler, 320
 iteratory, 423
 przesuwanie, 424

J

J2, 36
 Java, 13, 30, 41
 Java Archive, 694
 Java Community Process, 33
 Java EE, 37
 Java ME, 37
 Java Project, 44
 Java SDK, 36
 Java SE 6, 37
 Java Web Start, 693, 732
 javac, 41, 42, 145
 javadoc, 184, 189
 generowanie dokumentacji, 189
 komentarze, 186–188
 streszczenie, 185
 wstawianie komentarzy, 185
 znaczniki dokumentacyjne, 185
 javap, 769
 JavaScript, 34
 javaws, 735

jawna inicjalizacja pól, 167
 jconsole, 376
 jcontrol, 717
 JDK, Java Development Kit, 13, 25, 35, 36
 jednostki kodowe, 64, 80
 język
 C++, 23
 HTML, 31
 J++, 28
 Java, 13, 21
 JavaScript, 34
 UML, 134
 języki interpretowane, 33
 JFC, Java Foundation Classes, 488
 JNLP, Java Network Launch Protocol, 732–736
 JRE, Java Runtime Environment, 36, 37
 JSR, Java Specification Requests, 379
 just-in-time compilation, 25
 JVM, 696

K

kalendarz, 138
 katalog bazowy drzewa pakietu, 182
 keyboard focus, 548
 klasa, *Patrz także* plik, 56, 131, 566
 AdjustmentEvent, 561
 AccessibleObject, 254
 ActionEvent, 530, 537, 561
 AbstractAction, 547, 550, 607
 AbstractButton, 593, 609
 AbstractCollection, 426, 436
 AbstractList, 479
 Applet, 715, 718, 728
 AppletContext, 720
 Array, 256
 ArrayBlockingQueue, 799
 ArrayDeque, 430, 449
 ArrayDeque, 421
 ArrayIndexOutOfBounds, 116
 ArrayIndexOutOfBoundsException, 330, 831
 ArrayList, 116, 229–234, 380, 424, 809
 ArrayList<T>, 228
 ArrayStoreException, 402
 Arrays, 120, 122, 468
 AssertionError, 351
 AWTEvent, 560
 BadCastException, 411
 BigDecimal, 113, 115
 BigInteger, 113, 114
 BitSet, 419, 482, 483
 BitVector, 482
 BooleanHolder, 237
 BorderFactory, 593, 595
 BorderLayout, 575, 576, 644
 BoxLayout, 623
 BrokenBarrierException, 829

klasa

- ButtonGroup, 590, 591, 593, 610
- ByteArrayInputStream, 737
- ByteArrayOutputStream, 737
- CancellationException, 839
- ChangeEvent, 601
- Class, 242, 320, 411, 698
- Class<T>, 411, 412, 416
- ClassCastException, 207, 218, 466
- ClassLoader, 354
- ButtonFrame, 534
- CloneNotSupportedException, 285
- Collections, 464, 470–474
 - max(), 474
 - reverseOrder(), 471
 - unmodifiableList(), 464
- Color, 514, 516
- ColorAction, 533, 547
- Component, 499, 516, 573, 582
- ConcurrentHashMap, 802, 810
- ConcurrentLinkedQueue, 802, 803
- ConcurrentModificationException, 435, 802
- ConcurrentSkipListMap, 802–804
- ConcurrentSkipListSet, 802, 804
- Console, 89, 90
- ConsoleHandler, 360, 364, 371
- Constructor, 244, 246, 250, 252
- Constructor<T>, 412
- Container, 536, 573, 574
- CopyOnWriteArrayList, 808
- CopyOnWriteArraySet, 808
- CountDownLatch, 828
- Cursor, 554
- CyclicBarrier, 828, 829
- Date, 93, 135, 150
- DebugGraphics, 684
- DefaultButtonModel, 571, 572
- DelayQueue, 797, 800
- Deque, 448
- Dictionary, 392
- Dimension, 497
- Double, 62
- Ellipse2D, 506–509
- Ellipse2D.Double, 513
- EmptyStackException, 350
- Enum<T>, 241
- EnumMap, 430, 459, 461
- EnumSet, 430, 459, 461
- EOFException, 333
- Error, 330
- EventHandler, 538
- EventObject, 530, 538, 560
- EventQueue, 835
- Exception, 245, 330
- Exchanger, 828, 830
- ExecutorCompletionService, 821, 822
- Executors, 815, 819, 820
- ExtendedService, 737
- Field, 246, 250, 252, 255, 258
- File, 97
- FileContents, 737, 742
- FileFilter, 671
- FileHandler, 360, 361, 364
- FileInputStream, 331, 701
- FileNameExtensionFilter, 678
- FileNotFoundException, 331, 333, 738
- FileOpenService, 737, 742
- FileSaveService, 743
- FileView, 672, 678
- FlowLayout, 574
- FocusAdapter, 562
- FocusEvent, 561
- Font, 518, 522
- FontMetrics, 524
- FontRenderContext, 519
- Formatter, 364, 372
- Frame, 492, 499, 546
- FutureTask, 811, 815
- Graphics, 502, 516, 524
- Graphics2D, 506, 516–524
- GraphicsDevice, 498
- GraphicsEnvironment, 517, 690
- GregorianCalendar, 137, 140, 142
- GridBagConstraints, 625–628, 633
- GridLayout, 576, 579
- GroupLayout, 638, 641–643
- HashMap, 430, 451, 454, 809
- HashSet, 424, 430, 440, 442
 - dodawanie elementów, 445
 - iterator, 442
- Hashtable, 419, 479, 480, 809
- Handler, 360–362, 370
- GridBagLayout, 623, 638, 644
 - anchor, 627, 633
 - bottom, 627
 - dopełnienie, 627
 - dopełnienie wewnętrzne, 627
 - dopełnienie zewnętrzne, 627
 - fill, 627
 - GBC, 628
 - GridBagConstraints, 628
 - gridheight, 626, 627
 - gridwidth, 626, 627
 - gridx, 626, 627
 - ipadx, 627
 - klasa pomocnicza, 628
 - left, 627
 - ograniczenia, 626
 - right, 627
 - top, 627
 - weight, 626
- IdentityHashMap, 431, 460
- IllegalAccessException, 252
- IllegalArgumentException, 351
- IllegalStateException, 425
- Integer, 237, 238

- InputEvent, 559
- InterruptedException, 747, 752–756, 811
- IntHolder, 237
- IOException, 333, 336
- ItemEvent, 561
- JApplet, 715, 721
 - współrzędne, 506, 508
 - elipsa, 508
 - Ellipse2D, 506–509
 - figury geometryczne, 506
 - Line2D, 506
 - programowanie, 506
 - Shape, 506
 - rysowanie, 506
 - rysowanie figur, 510
 - Rectangle2D, 506–509
- JButton, 531–535, 548, 570
- JCheckBox, 588, 589
- JCheckBoxMenuItem, 610
- JColorChooser, 649, 679, 683
- JComboBox, 478, 597, 600
- JComponent, 502, 549, 582, 620
- JDialog, 659–663
- JFileChooser, 669, 677
- JFrame, 492, 495
- JLabel, 582
- JMenu, 607, 608
- JMenuBar, 606
- JMenuItem, 609, 612
- JPanel, 505, 576
- JOptionPane, 280, 649, 651, 659, 664
- JPasswordField, 584
- JPopupMenu, 612
- JRadioButton, 590, 592
- JRadioButtonMenuItem, 610
- JScrollPane, 585, 587
- JSlider, 600
- JTextArea, 580, 584, 586
- JTextComponent, 580
- JTextField, 570, 580, 582
- JUnit, 373
- JToolBar, 619, 622
- KeyAdapter, 562
- KeyEvent, 561
- KeyStroke, 548, 552, 613
- Line2D, 506
- Line2D.Double, 513
- LineMetrics, 520, 523
- LineBorder, 595, 597
- LinkedBlockingDeque, 797, 800
- LinkedBlockingQueue, 797
- LinkedHashMap, 431, 458–460
- LinkedHashSet, 430, 458, 460
- LinkedList, 421, 430, 433, 436, 440
- LinkedList<E>, 440
- List, 440, 473
- List<E>, 438
- Lock, 769, 772, 777
- Logger, 368
- LookAndFeelInfo, 541
- Math, 68, 154
- Method, 246, 250, 259, 416
- Modifier, 251
- MouseAdapter, 562
- MouseEvent, 559, 561, 612
- MouseHandler, 556
- MouseMotionAdapter, 562
- MouseMotionHandler, 556
- MouseWheelEvent, 561
- NoSuchElementException, 428, 439, 469
- NullPointerException, 330, 350, 353
- NumberFormat, 238
- NumberFormatException, 349
- Object, 131, 215, 222, 444, 781
- PaintEvent, 561
- PasswordChooser, 663, 664
- Point, 509
- Point2D, 509
- Point2D.Double, 508, 513
- Point2D.Float, 508
- Preferences, 706, 711
- PrintStream, 224, 737
- Proxy, 320, 324
- PriorityBlockingQueue, 797, 800
- PriorityQueue, 430, 451
- Properties, 481, 701–704
- PrintWriter, 95, 97
- PriorityQueue, 451
- Properties, 481
- Proxy, 324
- Queue, 448
- Random, 174
- Rectangle, 509
- Rectangle2D, 506–509
- Rectangle2D.Double, 507, 512
- Rectangle2D.Float, 507, 513
- RectangularShape, 509, 512
- ReentrantLock, 769, 772, 793
- ReentrantReadWriteLock, 793
- Robot, 686, 690
- RuntimeException, 330, 347, 349
- Runnable, 758
- Scanner, 88–90, 97
- Semaphore, 828
- SequentialGroup, 643
- ServiceManager, 742
- ServletException, 339
- SocketHandler, 360
- SoftBevelBorder, 595, 596
- SpringLayout, 624
- Stack, 419, 482
- StackTraceElement, 347
- StreamHandler, 361, 363
- StrictMath, 70
- String, 76–81
- StringBuilder, 85, 86

klasa

- SwingUtilities, 668
- SwingWorker, 835, 839, 840
- System, 705
- SystemColor, 515
- Thread, 754, 757, 763
 - currentThread(), 755
 - sleep(), 747
 - start(), 754
- ThreadDeath, 760
- ThreadGroup, 763, 764
- ThreadPoolExecutor, 816, 820
- Throwable, 245, 329, 339, 349
- Time, 93, 94
- TimeoutException, 811
- Timer, 278, 280
- Toolkit, 281, 497, 500, 559
- TreeMap, 430, 451, 454
- TreeSet, 430, 444–447, 450
- UIManager, 541
- UnsupportedOperationException, 462–466
- Update, 36
- Vector, 229, 419, 809
- WeakHashMap, 431, 457, 460
- WeakReference, 458
- Window, 499
- WindowAdapter, 543, 562
- WindowEvent, 530, 542, 561

klasy

- abstrakcyjne, 209, 273
 - interfejs, 273
 - zmienne obiektowe, 211
- adaptacyjne, 542, 543
- agregacja, 133
- anonimowe, 302, 313
- bazowe, 194
- definiowanie, 142
- diagramy, 134
- dziedziczenie, 131, 133, 194
- egzemplarz, 131
- finalne, 205
- generyczne, 228
- identyfikacja, 132
- implementacja interfejsu, 265
- komentarze, 186
- konstruktory, 135, 147
- macierzyste, 194
- metody, 131
- metody prywatne, 152
- metody statyczne, 155
- nadklasy, 194
- nazwy, 56
- parametryzowane, 381
- plik źródłowy, 183
- pochodne, 194
- podklasy, 194
- pola statyczne, 153
- pomocnicze, 628

- pośredniczące, 319
- potomne, 194
- predefiniowane, 134
- projektowanie, 190
- proxy
 - invocation handler, 320
 - ładowanie klas, 320
- publiczne, 175
- relacje między klasami, 133
- rozszerzanie, 131
- stałe, 67
- stałe pola, 152
- statyczne, 316
- ścieżka, 181
- uogólnione, 381, 382
 - definicja, 382
 - tworzenie egzemplarza, 383
- wewnętrzne, 265, 302
 - bezpieczeństwo, 309
 - dostęp do stanu obiektu, 303
 - dostęp do zmiennych, 310
 - metody, 302
 - referencja do klasy zewnętrznej, 306
 - referencja do obiektu zewnętrznego, 304
 - składnia, 303
 - this, 307
 - zastosowanie, 307
- wyliczeniowe, 240
- zagnieżdżone, 302
- zależność, 133

klasyfikacja wyjątków, 329

klawiatura, 548

klawisze, 548

kliknięcie przycisku, 531

klonowanie obiektów, 282

klucz URL, 738

kod

bajtowy, 27

mieszający, 220, 441

uogólniony, 387

wyjścia, 58

źródłowy, 56

kodowanie

Unicode, 62

UTF-16, 64

kolejka, 420, 448

blokująca, 795, 796

LinkedBlockingDeque, 797

LinkedBlockingQueue, 797

metody, 796

PriorityBlockingQueue, 797

dostępu, 647

dwukierunkowa, 448

priorytetowa, 450

kolejność bajtów, 26

kolekcje, 419

algorytmy, 469

ArrayDeque, 430

ArrayList, 430, 809

- bezpieczeństwo wątkowe, 802
- BitSet, 482, 483
- Collection, 429
- ConcurrentHashMap, 802
- ConcurrentLinkedQueue, 802
- ConcurrentSkipListMap, 802
- ConcurrentSkipListSet, 802
- CopyOnWriteArrayList, 808
- CopyOnWriteArraySet, 808
- Deque, 448
- dostęp swobodny, 429
- Enumeration, 480
- EnumMap, 430, 459
- EnumSet, 430, 459
- HashMap, 430, 451, 809
- HashSet, 424, 430, 440, 442
- Hashtable, 479, 809
- IdentityHashMap, 431, 460
- interfejsy, 419
- klasy, 430
- kolejka, 448
- kolejka priorytetowa, 450
- kolejność odwiedzania elementów, 424
- konwersje, 477
- LinkedHashMap, 431, 458
- LinkedHashSet, 430, 458
- LinkedList, 430
- List, 440
- listy cykliczne, 422
- listy powiązane, 431
- listy tablicowe, 440
- mapy, 451
- mapy własności, 481
- NavigableMap, 430
- NavigableSet, 430
- obiekty opakowujące, 462
- ograniczone, 422
- operacje opcjonalne, 466
- operacje zbiorcze, 476
- PriorityQueue, 430, 451
- Properties, 481
- Queue, 420, 448
- słabo spójne iteratory, 802
- SortedMap, 430
- SortedSet, 430
- sortowanie, 470
- Stack, 482
- starsze kolekcje bezpieczne wątkowo, 809
- stos, 482
- tablice kopiowane przy zapisie, 808
- tasowanie, 470
- TreeMap, 430, 451
- TreeSet, 430, 444
- uporządkowane, 429, 433
- Vector, 809
- warstwa interfejsów, 420
- warstwa klas konkretnych, 420
- WeakHashMap, 431, 457
- widoki kontrolowane, 465
- widoki niemodyfikowalne, 464
- widoki przedziałowe, 463
- widoki synchronizowane, 465
- wyliczenia, 480
- wyszukiwanie binarne, 473
- zbiory bitów, 482
- kolizja, 441
- nazw, 175
- kolory, 513
 - Color, 514
 - definiowanie, 514
 - system kolorów, 515
 - tło, 514
 - wybór, 679
- komentarze, 59
 - dokumentacyjne, 184
- komórki, 441
- komunikaty okna dialogowego
 - ERROR_MESSAGE, 650
 - INFORMATION_MESSAGE, 650
 - PLAIN_MESSAGE, 650
 - QUESTION_MESSAGE, 650
 - WARNING_MESSAGE, 650
- kompilacja w czasie rzeczywistym, 25
- kompilator, 42
- komponenty, 502, 565, 573
 - dymki, 620
 - etykiety, 582
 - JButton, 571
 - JCheckBox, 588
 - JCheckBoxMenuItem, 610
 - JColorChooser, 649, 679
 - JComboBox, 597
 - JFileChooser, 669
 - JLabel, 582
 - JMenuBar, 606
 - JMenuItem, 612
 - JOptionPane, 649
 - JPasswordField, 584
 - JPopupMenu, 612
 - JRadioButton, 590
 - JRadioButtonMenuItem, 610
 - JScrollPane, 585
 - JSlider, 600
 - JTextArea, 580, 584
 - JTextField, 580
 - JToolBar, 619
 - listy rozwijalne, 597
 - menu, 606
 - menu podręczne, 611
 - obszary tekstowe, 584
 - panele przewijane, 585
 - paski narzędzi, 618
 - pola haseł, 584
 - pola tekstowe, 580
 - pola wyboru, 587
 - przełączniki, 590

- komponenty
 - suwaki, 600
 - wybór opcji, 587
 - zarządzanie rozkładem, 572
- komputer wieloprocesorowy, 784
- komunikacja między apletami, 720, 727
- komunikaty o błędach, 46
- konfiguracja
 - menedżera dzienników, 358
 - projektu, 44
- konkatenacja, 76
- konstruktory, 135, 147
 - domyślny, 167
 - przeciążanie, 167
 - wirtualne, 244
- kontekst
 - graficzny, 502
 - urządzenia, 502
- kontener, 505, 573
- kontrola
 - dostępu, 303
 - grup zadań, 821
 - nazw, 303
- kontroler, 568
- konwersja
 - łańcucha na liczbę, 237
 - pomiędzy kolekcjami a tablicami, 477
 - typów numerycznych, 70
- kończenie działania programu, 58
- kopiowanie
 - głębokie, 284
 - obiektów, 282
 - plytkie, 283
 - tablicy, 118
- kowariantne typy zwracane, 204, 391
- kółko myszy, 561
- kubelki, 441
- kursory, 555

L

- layered pane, 501
- layout manager, 565
- leading, 519
- licencja GPL, 33
- liczby
 - całkowite, 26, 60
 - zmiennoprzecinkowe, 61
- linia pisma, 519
- linked list, 432
- listener interface, 530
- listener object, 279
- listy
 - cykliczne, 421
 - dwukierunkowe, 432
 - powiązane, 421, 431, 432
 - dodawanie elementów, 433, 434
 - ListIterator, 433

- metody get i set, 437
- ogniwa, 432
- usuwanie elementów, 432
- rozwijalne, 597
- tablicowe, 234, 440
- load factor, 442
- local inner class, 302
- logging proxy, 373
- logic_error, 331
- logiczne nazwy czcionek, 517
- lokalizacja, 359
- lokalne klasy wewnętrzne, 310

Ł

- ładowanie
 - klas, 320
 - usług, 712
- łańcuchy, 26, 76
 - długość, 80
 - dziedziczenia, 201
 - identyczność, 78
 - jednostki kodowe, 80
 - konkatenacja, 76
 - niezmienialność, 77
 - podłańcuchy, 76
 - porównywanie, 78
 - składanie, 85
 - String, 81
 - StringBuilder, 85
 - współdzielenie, 77
 - współrzędne kodowe znaków, 80
 - wyjątków, 339

M

- manifest, 695
- mapa, 451
 - akcji, 550
 - wejścia, 549
- mapy własności, 481, 701
- marker interface, 284
- maska BUTTON3_DOWN_MASK, 554
- maski bitowe, 553
- maszyna wirtualna Javy, 376
- mechanizm ładowania klas, 320
- menedżer zabezpieczeń, 729
- menu, 606
 - akceleratory, 612, 613
 - akcje, 607
 - aktywowanie elementów, 614
 - dezaktywowanie elementów, 614
 - elementy, 606, 607
 - ikony w elementach, 609
 - mnemoniki, 612
 - pasek, 606

- podmenu, 606
- pola wyboru, 610
- przełączniki, 610
- separatory, 607
- skróty klawiszowe, 613
- tworzenie, 606
- menu podręczne, 611
 - obsługa, 611
 - tworzenie, 611
- metadane, 31
- metoda
 - accept(), 678
 - actionPerformed(), 279, 311, 546, 607
 - add(), 113–115, 427, 622, 796
 - addGroup(), 642
 - addActionListener(), 530, 588, 607, 660
 - addAll(), 427, 439, 475, 477
 - addChoosableFileFilter(), 671, 677
 - addComponent(), 642
 - addContainerGap(), 643
 - addFirst(), 440, 449
 - addGap(), 642
 - addHandler(), 370
 - addItem(), 598, 600
 - addLast(), 440, 449
 - addLayoutComponent(), 644, 647
 - addPreferredGap(), 643
 - addPropertyChangeListener(), 546
 - addSeparator(), 607, 608, 619, 622
 - addWindowListener(), 543
 - adjustmentValueChanged(), 562
 - allOf(), 461
 - and(), 483
 - andNot(), 483
 - append(), 87, 587
 - appendCodePoint(), 87
 - ArrayAlg.getMiddle(), 385
 - await(), 774, 775
 - awaitUninterruptibly(), 793
 - await(), 774–793, 829
 - asList(), 462, 468
 - brighter(), 514
 - binarySearch(), 122, 321, 473
 - BorderFactory.createEtchedBorder(), 594
 - BorderFactory.createCompoundBorder(), 594
 - BorderFactory.createTitledBorder(), 594
 - beep(), 281
 - call(), 814
 - cancel(), 811, 815
 - canRead(), 742
 - canWrite(), 742
 - cast(), 411
 - ceiling(), 448
 - charAt(), 80, 81
 - checkedCollection(), 465, 467
 - checkedList(), 467
 - checkedMap(), 467
 - checkedSet(), 467
 - checkedSortedMap(), 467
 - checkedSortedSet(), 467
 - Class.forName(), 243, 245
 - clear(), 427, 483
 - clearAssertionStatus(), 354
 - clone(), 283, 284, 285, 324
 - close(), 370
 - codePointAt(), 80, 82
 - codePointCount(), 80, 83
 - compareTo(), 82, 114, 240, 266–270
 - Component.show(), 494
 - config(), 368
 - contains(), 427
 - containsAll(), 427
 - containsKey(), 453
 - containsValue(), 453
 - copy(), 475
 - copyArea(), 525, 527
 - copyOf(), 118, 122
 - create(), 743
 - currentThread(), 755, 757
 - createSequentialGroup(), 641
 - createScreenCapture(), 691
 - createTitledBorder(), 596
 - createBevelBorder(), 595
 - createCompoundBorder(), 596
 - createCustomCursor(), 559
 - createDialog(), 683
 - createEmptyBorder(), 595
 - createEtchedBorder(), 595
 - createFont(), 518
 - createLineBorder(), 595
 - createLoweredBevelBorder(), 595
 - createMatteBorder(), 595
 - createParallelGroup(), 641
 - darker(), 514
 - delay(), 691
 - delete(), 88, 743
 - deriveFont(), 518, 523
 - descendingIterator(), 448
 - destroy(), 718
 - drawImage(), 525, 527
 - disjoint(), 476
 - divide(), 114, 115
 - doInBackground(), 839, 840
 - Double.isNaN(), 62
 - draw(), 506, 513
 - drawString(), 503, 513, 524
 - element(), 449, 796
 - Employee.clone(), 391
 - ensureCapacity(), 230
 - entering(), 369
 - equals(), 78, 123, 215–221, 430
 - equalsIgnoreCase(), 78, 82
 - execute(), 841
 - exiting(), 369
 - exportSubtree(), 712
 - exportNode(), 712

metoda

- fill(), 123, 475, 513, 516
- finalize(), 174
- fine(), 368
- finer(), 368
- finest(), 369
- first(), 447
- flipDone(), 785
- floor(), 448
- flush(), 370
- focusGained(), 562
- focusLost(), 562
- Font.createFont(), 518
- format(), 364, 372
- formatMessage(), 364, 372
- forName(), 243, 245
- frequency(), 475
- get(), 231, 429, 707, 814
- getActionCommand(), 537, 593
- getActionCommands(), 561
- getActionMap(), 553
- getActualTypeArguments(), 417
- getAllItems(), 478
- getAllStackTraces(), 345
- getAncestorOfClass(), 668
- getApplet(), 720, 729
- getAppletContext(), 726, 728
- getAppletInfo(), 725
- getApplets(), 727, 728
- getAscent(), 523
- getAudioClip(), 726, 727
- getAutoCreateContainerGaps(), 642
- getAutoCreateGaps(), 642
- getAvailableFontFamilyNames(), 517
- getBackground(), 516
- getBoolean(), 707, 711
- getBounds(), 417
- getByteArray(), 707, 711
- getCause(), 346
- getCenter(), 509
- getCenterX(), 509, 512
- getCenterY(), 509, 512
- getClass(), 216, 227, 242, 324
- getClassName(), 347, 541
- getClickCount(), 559
- getCodeBase(), 725, 738, 742
- getColor(), 516, 683
- getColumns(), 582
- getComponentPopupMenu(), 612
- getConstructor(), 411, 412
- getConstructors(), 246, 250
- getContentPane(), 501, 504
- getDay(), 139
- getDeclareFields(), 255
- getDeclaredConstructor(), 411, 412
- getDeclaredConstructors(), 246, 250
- getDeclaredField(), 255
- getDeclaredFields(), 246, 250–252
- getDeclaredMethods(), 246, 250
- getDeclaringClass(), 250
- getDefaultScreenDevice(), 690
- getDefaultToolkit(), 281, 497, 500
- getDefaultUncaughtExceptionHandler(), 763
- getDelay(), 797, 800
- getDescent(), 523
- getDescription(), 672, 678
- getDocumentBase(), 725, 726
- getDouble(), 707, 711
- getEnumConstants(), 411
- getExceptionTypes(), 250
- getExtendedState(), 500
- getFamily(), 523
- getField(), 255
- getFields(), 246, 250, 255
- getFileName(), 347
- getFilter(), 370
- getFintName(), 523
- getFirst(), 440, 449
- getFloat(), 707, 711
- getFont(), 524, 582
- getFontMetrics(), 521, 524
- getFontRenderContext(), 519–524
- getForeground(), 516
- getFormatter(), 370
- getGenericComponentType(), 417
- getGenericInterfaces(), 416
- getGenericParameterTypes(), 416
- getGenericReturnType(), 416
- getGenericSuperclass(), 416
- getHandlers(), 370
- getHead(), 364, 372
- getHeight(), 509, 512, 520, 524
- getHonorsVisibility(), 641
- getIcon(), 583, 672, 678
- getIconImage(), 499
- getImage(), 726, 727
- getInheritsPopupMenu(), 612
- getInputMap(), 549, 553
- getInputStream(), 737, 742
- getInstalledLookAndFeels(), 541
- getInt(), 707, 711
- getKey(), 457
- getKeyStroke(), 548, 550, 552
- getLargestPoolSize(), 820
- getLast(), 440, 449
- getLeading(), 523
- getLength(), 256, 258
- getLevel(), 370, 371
- getLineMetrics(), 520, 523
- getLineNumber(), 347
- getLocalGraphicsEnvironment(), 690
- getLogger(), 368
- getLoggerName(), 371
- getLong(), 707, 711
- getLowerBounds(), 417
- getMaxX(), 512

getMaxY(), 512
 getMessage(), 335, 371
 getMethod(), 259, 260
 getMethodName(), 347
 getMethods(), 246, 250
 getMillis(), 372
 getMinX(), 512
 getMinY(), 512
 getModifiers(), 246, 250
 getModifiersEx(), 554, 559
 getModifiersExText(), 559
 getMonth(), 139
 getName(), 228, 417, 523, 672, 742
 getNames(), 743
 getNewState(), 545
 getOldState(), 545
 getOutputStream(), 737, 742
 getOwnerType(), 417
 getPaint(), 516
 getParameter(), 721, 725
 getParameterInfo(), 725
 getParameters(), 371
 getParameterTypes(), 250
 getParent(), 370
 getPassword(), 584
 getPath(), 670
 getPoint(), 559
 getPredefinedCursor(), 554
 getProperties(), 702, 705
 getProperty(), 481, 702, 705
 getProxyClass(), 324
 getRawType(), 417
 getResource(), 698, 700
 getResourceAsStream(), 698, 700
 getResourceBundle(), 371
 getResourceBundleName(), 371
 getReturnType(), 250
 getRootPane(), 665, 668
 getScreenDevices(), 690
 getScreenSize(), 497, 500
 getSelectedFile(), 670, 677
 getSelectedFiles(), 670, 677
 getSelectedItem(), 598, 600
 getSelectedObjects(), 591
 getSelection(), 591, 593
 getSequenceNumber(), 372
 getServiceNames(), 742
 getSource(), 561
 getSourceClassName(), 371
 getSourceMethodName(), 372
 getStackTrace(), 345, 347
 getState(), 760, 841
 getStringBounds(), 519–523
 getSuperClass(), 228, 412
 getTail(), 364, 372
 getText(), 580, 581
 getThreadID(), 372
 getThrown(), 371
 getTime(), 206
 getTitle(), 496, 499
 getTotalBalance(), 772
 getTypeDescription(), 672, 678
 getTypeParameters(), 416
 getUnhandledExceptionHandler(), 763
 getUpperBounds(), 417
 getUseParentHandlers(), 370
 getValue(), 457, 546, 547, 552
 getWidth(), 507–509, 512, 520
 getX(), 512, 559
 getY(), 512, 520, 559
 getYear(), 139
 hashCode(), 220, 324, 441, 444
 hasMoreElements(), 424, 481
 hasNext(), 90, 423, 428, 435
 hasNextDouble(), 90
 hasNextInt(), 90
 hasPrevious(), 434, 439
 headMap(), 469
 headSet(), 468
 higher(), 448
 identityHashCode(), 462
 importPreferences(), 712
 indexOf(), 82, 166, 439
 indexOfSubList(), 475
 info(), 368
 init(), 718, 720
 initCause(), 346
 insert(), 87, 608
 insertSeparator(), 608
 insertItemAt(), 598, 600
 Integer.parseInt(), 237
 interrupt(), 755, 757
 interrupted(), 756, 757
 intValue(), 238
 invoke(), 259, 261, 324
 invokeAll(), 821
 invokeAndWait(), 832, 835
 invokeAny(), 821
 isAbstract(), 251
 invokeLater(), 832, 835, 842
 isAccessible(), 255
 isCancelled(), 815, 816
 isDefaultButton(), 668
 isDispatchThread(), 835
 isDone(), 784, 811, 815
 isEditable(), 580, 600
 isEmpty(), 427
 isEnabled(), 546, 552
 isFinal(), 246, 251
 isInterface(), 251
 isInterrupted(), 755–757
 isLocationByPlatform(), 497, 499
 isLoggable(), 363, 372
 isNaN(), 62
 isNative(), 251
 isNativeMethod(), 347

metoda

- isPrivate(), 246, 251
- isProtected(), 251
- isProxyClass(), 325
- isPopupTrigger(), 612
- isPublic(), 246, 251
- isResizable(), 499
- isSelected(), 590, 591, 611
- isStatic(), 251
- isStrict(), 251
- isSynchronized(), 251
- isTraversable(), 672, 679
- isUndecorated(), 500
- isVisible(), 499
- iterator(), 422, 426
- isVolatile(), 251
- itemStateChanged(), 562
- isWebBrowserSupported(), 742
- join(), 760
- JMenu.remove(), 614
- JFrame.add(), 501
- last(), 447
- lastIndexOf(), 83, 439
- lastIndexOfSubList(), 475
- layoutContainer(), 644, 647
- length(), 80, 83, 483
- linkSize(), 641
- listIterator(), 438
- load(), 482, 705
- lock(), 772, 791
- lockInterruptibly(), 792
- log(), 369
- Logger.global.info(), 355
- Logger.global.setLevel(), 355
- logp(), 357, 369
- logrb(), 369
- lookup(), 742
- lower(), 448
- main(), 56, 57, 156
- Math.round(), 71
- Math.sqrt(), 260, 351
- max(), 474, 475
- menuCanceled(), 615
- menuDeselected(), 615
- menuSelected(), 615
- min(), 475
- minimumLayoutSize(), 644, 647
- mod(), 114
- Modifier.toString(), 246
- mouseDragged(), 555, 563
- mouseClicked(), 553, 563
- mouseEntered(), 556, 563
- mouseExited(), 556, 563
- MouseListener(), 554
- mouseMove(), 691
- mouseMoved(), 555, 563
- mousePress(), 691
- mousePressed(), 553, 554, 563
- mouseRelease(), 691
- mouseReleased(), 553, 563
- move(), 746
- multiply(), 113–115
- nCopies(), 463, 467
- newCachedThreadPool(), 815, 819
- newCondition(), 777
- newFixedThreadPool(), 815, 819
- newInstance(), 244, 256, 258, 411
- newProxyInstance(), 320, 324
- newScheduledThreadPool(), 816, 820
- newSingleThreadExecutor(), 816, 819
- newSingleThreadScheduledExecutor(), 816, 820
- next(), 88, 423–425, 428
- nextDouble(), 89, 90
- nextElement(), 424, 481
- nextIndex(), 437, 439
- nextLine(), 88, 90
- nextInt(), 88, 90, 174
- node(), 711
- noneOf(), 461
- notify(), 778, 781, 783
- notifyAll(), 778–783
- Object.clone(), 285, 391
- of(), 461
- offer(), 448, 796, 801
- offerFirst(), 449, 801
- offerLast(), 449, 801
- offsetByCodePoints(), 80, 82
- openFileDialog(), 737, 742
- openMultiFileDialog(), 742
- ordinal(), 241
- or(), 483
- paint(), 747
- paintComponent(), 333, 502–506, 650, 795
- parse(), 238
- parseInt(), 237, 238, 721
- peek(), 449, 482, 796
- peekFirst(), 449
- peekLast(), 449
- play(), 726
- pop(), 482
- pow(), 69
- Preferences.systemNodeForPackage(), 707
- Preferences.systemRoot(), 707
- Preferences.userNodeForPackage(), 707
- Preferences.userRoot(), 707
- poll(), 448, 796, 801, 822
- pollFirst(), 448, 449, 801
- pollLast(), 448, 449, 801
- preferredLayoutSize(), 644, 647
- previous(), 434, 437, 439
- print(), 59, 91
- previousIndex(), 437, 439
- printf(), 238
- println(), 323
- printStack(), 245
- printStackTrace(), 246, 374

process(), 839, 841
 publish(), 370, 841
 putLong(), 711
 putValue(), 546, 552, 609, 620
 push(), 482
 put(), 429, 453, 707, 796
 putAll(), 453
 putBoolean(), 711
 putByteArray(), 711
 putDouble(), 711
 putFirst(), 801
 putFloat(), 711
 putInt(), 707, 711
 putLast(), 801
 range(), 461
 readLine(), 90
 readLock(), 793
 readPassword(), 90
 removeEldestEntry(), 461
 remove(), 423–429, 608, 796
 removeAll(), 427
 removeAllItems(), 600
 run(), 752, 754, 755
 replace(), 83
 replaceAll(), 475
 repozytorium Preferences, 706
 resetChoosableFileFilters(), 678
 resetChoosableFilters(), 671
 resize(), 718
 repaint(), 502, 505, 747
 resume(), 761
 retainAll(), 427, 476
 revalidate(), 581, 582
 reverse(), 475
 reverseOrder(), 471
 rotate(), 475
 removeFirst(), 440, 449
 removeHandler(), 370
 removeItem(), 598, 600
 removeItemAt(), 598, 600
 removeLast(), 440, 449
 removeLayoutComponent(), 644, 647
 removePropertyChangeListener(), 546
 saveAsFileDialog(), 743
 saveFileDialog(), 736, 737
 schedule(), 820
 scheduleAtFixedRate(), 820
 scheduleWithFixedDelay(), 820
 setAcceptAllFileFilterUsed(), 671, 678
 setAccessible(), 252, 254
 setAccelerator(), 613, 614
 setActionCommand(), 593
 setAction(), 608
 set(), 231, 255, 435, 483
 setAccessory(), 678
 setAnchor(), 629
 setAutoCreateContainerGaps(), 642
 setAutoCreateGaps(), 642
 setBackground(), 514–516
 setBorder(), 594, 597
 setBounds(), 494, 496, 499
 setCharAt(), 87
 setClassAssertionStatus(), 354
 setColor(), 516, 683
 setColumns(), 581, 585, 587
 setComponentPopupMenu(), 611, 612
 setCurrentDirectory(), 669, 677
 setCursor(), 560
 setDaemon(), 762
 setDebugGraphicsOptions(), 684
 setDefaultAssertionStatus(), 354
 setDefaultButton(), 665, 668
 setDefaultCloseOperation(), 494, 718
 setDefaultUncaughtExceptionHandler(), 763
 setDisplayedMnemonicIndex(), 613, 614
 setDone(), 784
 setEchoChar(), 584
 setEditable(), 580, 600
 setEnabled(), 546, 552, 614
 setExtendedState(), 498, 500
 setFileFilter(), 677
 setFileSelectionMode(), 670, 677
 setFileView(), 672, 678
 setFill(), 629
 setFilter(), 363, 370
 setFont(), 524, 582
 setForeground(), 516
 setFormatter(), 370
 setFrameFromCenter(), 510
 setFrameFromDiagonal(), 510
 setHonorsVisibility(), 641
 setHorizontalGroup(), 634, 641
 setHorizontalTextPosition(), 609
 setIcon(), 583
 setIconImage(), 494, 500
 setInheritsPopupMenu(), 611, 612
 setInverted(), 605
 setJMenuBar(), 607, 609
 setLabelTable(), 602, 606
 setLayout(), 574, 576
 setLevel(), 370
 setLineWrap(), 585, 587
 setLocation(), 494, 496, 499
 setLocationByPlatform(), 496–499
 setLookAndFeel(), 539, 541
 setMajorTickSpacing(), 601, 606
 setMinorTickSpacing(), 601, 606
 setMnemonic(), 613, 614
 setMultiSelectionEnabled(), 670, 677
 setPackageAssertionStatus(), 354
 setPaint(), 513–516
 setPaintLabels(), 602, 606
 setPaintTicks(), 601, 605
 setPaintTrack(), 605, 606
 setParent(), 370
 setPriority(), 761

metoda

- setRect(), 508
- setResizable(), 495, 499
- setRows(), 585, 587
- setSelected(), 588, 590, 610
- setSelectedFile(), 670, 677
- setSelectedFiles(), 677
- setSelectionEnd(), 842
- setSelectionStart(), 842
- setSize(), 497, 499, 718
- setSnapToTicks(), 601, 606
- setSource(), 538
- setTabSize(), 587
- setText(), 580–583, 663, 832
- setTime(), 206
- setTitle(), 495–499, 718
- setToolTipText(), 620, 622
- setUncaughtExceptionHandler(), 763
- setUndecorated(), 494, 500
- setUseParentHandlers(), 370
- setValue(), 457
- setVerticalGroup(), 641
- setVisible(), 494, 663, 718, 841
- setWrapStyleWord(), 587
- severe(), 368
- store(), 482, 701, 705
- shutdown(), 820
- signal(), 775–790
- signalAll(), 774–789
- singleton(), 467
- singletonList(), 463
- singletonMap(), 463
- size(), 230, 426, 437
- sleep(), 751, 832
- sort(), 120, 269, 470, 472
- start(), 280, 718, 754
- stop(), 281, 718, 760, 794
- show(), 494, 612
- showConfirmDialog(), 651, 657
- showDialog(), 677, 683
- showDocument(), 728, 742
- showInputDialog(), 650, 658
- showInternalConfirmDialog(), 657
- showInternalInputDialog(), 658
- showInternalMessageDialog(), 656
- showInternalOptionDialog(), 657
- showMessageDialog(), 280, 650, 656
- showOpenDialog(), 669, 677
- showOptionDialog(), 650, 657
- showSaveDialog(), 670, 677
- showStatus(), 727
- shuffle(), 472
- start(), 718
- stop(), 718
- String.format(), 93
- subSet(), 468
- submit(), 819, 822
- subMap(), 469
- subList(), 463, 468
- substring(), 76, 79, 83
- subtract(), 114, 115
- super.clone(), 284
- suspend(), 760, 794
- swap(), 475
- SwingUtilities.updateComponentTreeUI(), 539
- synchronizedCollection(), 465, 467
- synchronizedList(), 467
- synchronizedMap(), 465, 467
- synchronizedSet(), 467
- synchronizedSortedMap(), 467
- synchronizedSortedSet(), 467
- System.exit(), 58, 494
- System.getProperty(), 702
- System.out.print(), 91
- System.out.println(), 88
- System.runFinalizersOnExit(), 174
- systemNodeForPackage(), 711
- systemRoot(), 707, 711
- tailMap(), 469
- tailSet(), 468
- take(), 796, 801, 822
- takeFirst(), 801
- takeLast(), 801
- text(), 90
- Thread.dumpStack(), 374
- Thread.getAllStackTraces(), 345
- throwing(), 369
- toArray(), 427, 428
- toBack(), 499
- toFront(), 495, 499
- toLowerCase(), 83
- toString(), 88, 122, 222, 238–241, 347
- toUpperCase(), 83
- transfer(), 772
- trim(), 83, 581
- trimToSize(), 230, 231
- tryLock(), 792
- uncaughtException(), 763
- unlock(), 770, 772
- unmodifiableCollection(), 465
- unmodifiableList(), 464–466
- unmodifiableMap(), 467
- unmodifiableSet(), 465
- unmodifiableSortedMap(), 467
- unmodifiableSortedSet(), 466
- updateComponentTreeUI(), 539
- userNodeForPackage(), 711
- userRoot(), 707, 711
- validate(), 582, 842
- valueOf(), 114, 238, 241
- void(), 755
- warning(), 368
- windowActivated(), 542, 545, 563
- wait(), 778–783, 790
- windowClosed(), 542, 545, 563
- windowClosing(), 542, 545, 563

windowDeactivated(), 542, 563
 WindowDeactivated(), 545
 windowDeiconified(), 542, 545, 563
 WindowEvent(), 545
 windowGainedFocus(), 563
 windowIconified(), 542, 545, 563
 windowLostFocus(), 563
 windowOpened(), 542, 545, 563
 windowStateChanged(), 545, 563
 writeLock(), 794
 xor(), 483
 yield(), 762
 metody, 57, 131
 abstrakcyjne, 209
 ciało, 58
 domyślne, 275
 generyczne, 384, 425
 dostęp do pól, 149
 fabrykujące, 156
 finalne, 205
 komentarze, 186
 parametry, 59, 159
 parametryzowane, 384
 pomostowe, 390
 prywatne, 152
 przeciążanie, 165
 rodzime, 155
 statyczne, 69, 155, 274
 sygnatura, 166, 204
 synchronized, 778
 udostępniające, 139
 uogólnione, 384, 425
 wstawiane, 149
 wnioskowanie o typie, 385
 MIME, 732
 mnemoniki, 612
 modal dialog box, 648
 modalne okna dialogowe, 648
 modalność, 659
 model, 568, 569
 modeless dialog box, 648
 modifier keys, 554
 moduł ładujący klasy, 352
 modyfikacja parametru obiektowego, 161
 modyfikatory dostępu, 56, 214
 monitor, 783
 motywy, 491
 mouseWheelMoved, 563
 multiple inheritance, 274
 multithreaded, 745
 mutable class, 153
 MVC, Model-View-Controller, 565–568
 mysz, 553, 561
 kursory, 555

N

nadklasy, 194
 nadtypy typów wieloznacznych, 405
 NaN, 62, 68
 narzędzia wiersza poleceń, 41
 nawiasy, 74
 klamrowe, 57
 nazwy, 56
 klas, 56, 192
 metod, 192
 parametrów, 168
 rodziny czcionek, 517
 zmiennych, 65
 NetBeans, 36
 New Project, 43
 niemodalne okna dialogowe, 648
 niemodyfikowalne widoki, 464
 niezależność od architektury, 25
 niszczenie obiektów, 174
 notacja wielbłądzia, 56
 null, 137, 166

O

obiekt, 131
 Handler, 362
 System.out, 68
 zdarzeń, 530
 obiektowość, 23
 obiekty, 23, 60
 autoboxing, 235
 Handler, 360
 hermetyzacja, 131
 klonowanie, 282
 konstruktory, 135, 147
 kopiowanie, 282
 metody, 131, 139, 152
 nasłuchujące, 279
 niszczenie, 174
 opakowujące, 462
 polimorfizm, 199
 porównywanie, 215
 pośredniczące, 320
 składowe, 131, 150
 stan, 131
 tożsamość, 131
 tworzenie, 135, 165
 warunków, 772
 właściwości, 131
 zachowanie, 131
 obliczenia, 68
 obramowanie, 593
 obrazy, 525
 wczytywanie, 525
 wyświetlanie, 525

- obsługa
 - błędów, 328
 - ramek, 498
 - wyjątków, 243, 327, 348
 - zdarzeń, 279, 529, 562
 - ActionEvent, 530, 561
 - ActionListener, 546
 - actionPerformed(), 531
 - AdjustmentEvent, 561
 - akcje, 546
 - AWTEvent, 560
 - EventObject, 530, 560
 - FocusEvent, 561
 - hierarchia zdarzeń, 560
 - interfejs nasłuchu, 530
 - ItemEvent, 561
 - KeyEvent, 561
 - klasy adaptacyjne, 542, 543
 - kliknięcie przycisku, 531
 - MouseEvent, 561
 - MouseMotionListener, 554
 - MouseWheelEvent, 561
 - mysz, 553
 - obiekt zdarzeń, 530
 - procedura obsługi, 529
 - słuchacz, 530
 - WindowEvent, 530, 561
 - źródło, 530
 - obszary
 - surogatów, 64
 - tekstowe, 584
 - odbieranie danych wejściowych, 88
 - odczyt
 - danych, 88
 - z pliku, 95
 - odmierzanie czasu, 278, 791
 - opakowywanie, 236
 - odrzucone metody, 139
 - odzworowanie nazw czcionek, 518
 - ogniwa, 432
 - ograniczenia zmiennych typowych, 385
 - okna, 542
 - okno dialogowe, 648
 - akcesorium, 673
 - dane wejściowe, 649
 - JColorChooser, 649, 679
 - JFileChooser, 669
 - JOptionPane, 649
 - klawisz wyzwolenia, 665
 - komunikaty, 650
 - modalne, 648, 660
 - modalność dokumentu, 660
 - modalność zestawu narzędzi, 660
 - niemodalne, 648
 - opcje, 649
 - panel główny, 665
 - potwierdzenia, 651
 - przycisk domyślny, 665
 - przyciski, 650
 - ramka nadrzędną, 659
 - tworzenie, 659
 - wybór kolorów, 679
 - wybór plików, 669
 - wymiana danych, 663
 - wyświetlanie, 664
 - OOP, Object Oriented Programming, 130
 - opakowywanie, 236
 - opcje okna dialogowego
 - CANCEL_OPTION, 651
 - CLOSED_OPTION, 651
 - DEFAULT_OPTION, 650
 - OK_CANCEL_OPTION, 650, 651
 - OK_OPTION, 651
 - NO_OPTION, 651
 - YES_NO_CANCEL_OPTION, 650
 - YES_NO_OPTION, 650
 - YES_OPTION, 651
 - operacje
 - opcjonalne, 466
 - zbiorcze, 476
 - operator
 - new, 88, 135, 758
 - nierówności, 73
 - równości, 73
 - trójargumentowy, 73
 - operatory, 67
 - arytmetyczne, 67, 72
 - bitowe, 73
 - dekrementacja, 72
 - inkrementacja, 72
 - instanceof, 208, 272
 - logiczne, 73
 - nawiasy, 74
 - priorytety, 74
 - przesunięcie bitowe, 74
 - przyrostkowe, 72
 - relacyjne, 73
 - skraccanie, 72
 - sprawdzania zakresu, 119
 - opisanie prostokąta, 508
 - optional operations, 466
 - osłona obiektów, 235
 - ośrodek certyfikacji, 730
 - overloading resolution, 166
- P**
- pakiet
 - CORBA, 237
 - JDK, 36
 - java.lang.reflect, 246, 413
 - javax.swing, 278, 493
 - java.util.concurrent, 759, 770, 797
 - java.util.EventObject, 530
 - java.util.logging.config.class, 359
 - java.util.logging.config.file, 358

- java.util.logging.LogManager, 359
- java.util.logging.manager, 359
- Swing, 488
- pakiety, 174
 - dodawanie klasy, 177
 - domyślny, 177
 - import klas, 175
 - komentarze, 188
 - lokalizacyjne, 359
 - pieczętowanie, 700
 - zasięg, 180
- panel przewijany, 585
- parametry, 59, 159
 - aplety, 720
 - jawne, 148
 - nazwy, 168
 - niejawne, 148
 - typowe, 380
 - wiersza poleceń, 119
- parent class, 194
- pasek menu, 606
- paski narzędzi, 618
- pętle, 100
 - break, 110
 - continue, 112
 - do while, 103
 - for, 104
 - for each, 30, 116
 - liczba iteracji, 104
 - natychmiastowe przejście do nagłówka, 112
 - przerywanie działania, 110
 - while, 100, 103
- piaskownica, 729, 732
- pieczętowanie pakietów, 700
- planowanie wykonywania, 820
- platforma programistyczna, 21
- plik
 - AboutDialog.java, 662
 - ActionFrame.java, 551
 - AnonymousInnerClassTest.java, 314
 - Ball.java, 749
 - BallComponent.java, 750
 - Bank.java, 779
 - BlockingQueueTest.java, 797
 - Bounce.java, 747
 - BounceThread.java, 752
 - ButtonFrame.java, 534
 - ButtonPanel.java, 655
 - CalculatorFrame.java, 739
 - Chart.java, 723
 - CircleLayout.java, 644
 - CircleLayoutFrame.java, 646
 - CloneTest.java, 286
 - ColorChooserPanel.java, 681
 - DataExchangeFrame.java, 665
 - DialogFrame.java, 661
 - DrawTest.java, 511
 - Employee.java, 270, 287
 - EmployeeSortTest.java, 269
 - EventTracer.java, 685
 - FileIconView.java, 676
 - FirstSample.java, 59
 - FontFrame.java, 629, 638
 - FontTest.java, 521
 - forkJoinTest.java, 823
 - FutureTest.java, 812
 - GBC.java, 631
 - GenericReflectionTest.java, 414
 - ImagePreviewer.java, 675
 - ImageTest.java, 525
 - ImageViewer.java, 47
 - ImageViewerFrame.java, 674
 - InnerClassTest.java, 305
 - LambdaTest.java, 291
 - LoggingImageViewer.java, 365
 - LotteryOdds.java, 107
 - MenuFrame.java, 616
 - MouseComponent.java, 556
 - MouseFrame.java, 556
 - NotHelloWorld.java, 503
 - OptionDialogFrame.java, 652
 - PairTest1.java, 383
 - PairTest2.java, 387
 - PairTest3.java, 409
 - PasswordChooser.java, 666
 - PlafFrame.java, 539
 - PreferencesTest.java, 708
 - PropertiesTest.java, 702
 - ProxyTest.java, 321
 - ResourceTest.java, 699
 - Retirement.java, 101
 - Retirement2.java, 103
 - RoadApplet.html, 49
 - RoadApplet.java, 52
 - RobotTest.java, 688
 - Sieve.cpp, 485
 - Sieve.java, 484
 - SimpleFrameTest.java, 492
 - SizedFrameTest.java, 497
 - StackTraceTest.java, 346
 - StaticInnerClassTest.java, 318
 - SwingThreadTest.java, 832
 - SwingWorkerTest.java, 836
 - ThreadPoolTest.java, 817
 - TimerTest.java, 279
 - ToolBarFrame.java, 621
- pliki
 - audio
 - AIFF, 725
 - AU, 725
 - MIDI, 725
 - WAV, 725
 - graficzne
 - GIF, 725
 - JPEG, 725
 - PNG, 725

- pliki
 - java, 56
 - JAR, 181, 694
 - manifest, 695
 - pieczętowanie pakietów, 700
 - sekcja główna, 695
 - wykonywalne pliki, 696
 - zasoby, 697
 - zmiana zawartości pliku manifestu, 695
 - jnlp, 732
 - kod źródłowy, 56
 - odczyt, 95
 - tekstowe, 96
 - zapis, 95
 - ZIP, 694
 - źródłowe, 40
- pluggable look and feel, 570
- plytka kopia, 283
- pobieranie
 - danych, 88
 - pakietu JDK, 36
- podklasy, 194
- podłańcuchy, 76
- podmenu, 606
- podpisywanie kodu, 730
- podwójna precyzja, 61
- pola
 - hasel, 584
 - klasowe, 154
 - kombi, 597
 - statyczne, 153
 - tekstowe, 580
 - ulotne, 783
 - wyboru, 587
- polimorfizm, 199, 201, 263
- pop-up menu, 611
- pop-up trigger, 611
- porównywanie
 - łańcuchów, 78
 - obiektów, 215
- powiadamanie o zdarzeniach, 532
- powiązana tablica mieszająca, 458
- powiązania między klasami, 133
- powtórne generowanie wyjątków, 339
- pozycjonowanie
 - bezwzględne, 643
 - ramki, 494
- preferencje użytkownika, 701
 - API Preferences, 706
 - mapy własności, 701
- priorytety
 - operatorów, 74, 75
 - wątków, 761
- procedura obsługi
 - błędów, 329
 - nieprzechwyconych wyjątków, 762
 - zdarzeń, 529
- procesy, 746
- program, 56
 - java, 42
 - OptionDialogTest, 652
 - Swing graphics debugger, 684
- programowanie
 - generyczne, 379
 - proceduralne, 130
 - uogólnione, 380
 - dziedziczenie, 402
 - egzemplarze zmiennych typowych, 395
 - klasy uogólnione, 381, 382
 - konflikty, 401
 - maszyna wirtualna, 387
 - metody uogólnione, 384
 - ograniczenia, 393
 - parametry typowe, 380
 - refleksja, 411
 - sprawdzanie typów, 393
 - statyczny kontekst klas uogólnionych, 398
 - tablice typów uogólnionych, 394
 - translacja metod uogólnionych, 389
 - translacja wyrażeń generycznych, 389
 - typy proste, 393
 - typy surowe, 388
 - typy wieloznaczne, 404
 - wyjątki, 398
 - zastosowanie, 381
 - zachowawcze, 350
 - zorientowane obiektowo, 130
- programy
 - refleksyjne, 242
 - wielowątkowe, 745
- projektowanie
 - klas, 190
 - zorientowane obiektowo, 23
- property map, 481, 701
- prostokąt, 507, 508
- prywatne pola, 150
- przechwytywanie wyjątków, 244, 335, 337
- przeciążanie, 165, 166
 - konstruktorów, 167
- przeglądarka appletów, 716
- przekazywanie
 - przez wartość, 162
 - wyjątków, 350
- przełączniki, 590
 - powiadamanie o zdarzeniach, 590
- przenośność, 26
- przepełnienie stosu, 24
- przepływ sterowania, 97
 - instrukcji do-while, 105
 - instrukcji switch, 109
 - instrukcji while, 102
 - pętli for, 105
- przerywanie
 - działania pętli, 110
 - przepływu sterowania, 110
 - wątków, 755

przestrzeń numeracyjna, 64
 przesunięcie bitowe, 74
 przesuwanie iteratora, 424
 przetwarzanie
 danych wejściowych, 89
 wyrażeń lambda, 298
 przyciski, 531, 571, 623
 przywileje klasowe, 151
 publiczne
 metody akcesora, 150
 metody mutatora, 150
 pola danych, 150
 puchnięcie kodu szablonów, 388
 pule wątków, 815
 kontrola grup zadań, 821
 planowanie wykonywania, 820
 tworzenie, 815

Q

queue, 420
 quick sort, 120, 471

R

race condition, 764
 radio button, 565, 587
 ragged arrays, 127
 ramki, 492
 pozycjonowanie, 494
 rozmiar, 497
 warstwy, 501
 własności, 496
 wyświetlanie, 494
 referencja
 do obiektu, 136
 do konstruktora, 295
 do metody, 293
 reflection, 193
 refleksja, 193, 242, 264
 analiza funkcjonalności klasy, 246
 analiza obiektów, 251
 Class, 242
 Constructor, 246
 Field, 246, 258
 generyczny kod tablicowy, 255
 informacje o typach generycznych, 412
 klasy, 242
 Method, 246
 nazwy pól, 251
 parametry `Class<T>`, 412
 typy pól, 251
 typy uogólnione, 411
 wskaźniki do metod, 258
 rejestrujący obiekt pośredni, 373

relacje między klasami, 133, 262
 resource bundle, 359
 RGB, 680
 rodzina czcionek, 517
 root pane, 501
 rozkład
 brzegowy, 574
 grupowy, 634
 siatkowy, 576
 sprężynowy, 624
 rozmiar
 ekranu, 497
 ramki, 497
 rozstrzyganie przeciążania, 166, 203
 rozszerzanie klas, 131
 runtime_error, 331
 rysowanie, 488, 502
 figury 2W, 505
 rzutowanie, 71, 206, 208

S

sandbox, 729, 732
 dostęp do zasobów lokalnych, 735
 scroll pane, 585
 SDK, 36
 SE, 36
 security manager, 729
 semafore, 827
 sieciowość, 24
 skład tekstów, 520
 składanie łańcuchów, 78, 85
 składnia
 Javy, 23
 wyrażeń lambda, 289
 składowe, 131, 150
 slider, 587
 słabe referencje, 458
 słabo spójne iteratory, 802
 słowo kluczowe, 843
 abstract, 209
 assert, 351
 break, 97, 108, 110
 case, 108
 catch, 245, 335
 class, 56, 143
 const, 67
 continue, 112
 default, 108
 else, 99
 enum, 75
 extends, 194, 272
 final, 67, 152, 205
 finally, 340, 342
 if, 98
 implements, 267
 import, 89, 175

- słowo kluczowe
 - instanceof, 208, 284
 - interface, 266
 - new, 88, 115, 135
 - package, 177, 181
 - packages, 174
 - private, 146, 152
 - protected, 185, 214
 - public, 56, 146, 180, 214
 - static, 154–156, 171
 - static binding, 203
 - static final, 67
 - static void, 58
 - strictfp, 68
 - super, 196, 197
 - switch, 108
 - synchronized, 769, 777
 - this, 148, 155, 307
 - throw, 333
 - throws, 96, 331
 - try, 245, 335
 - view, 462
 - void, 58
 - volatile, 784
 - while, 100, 103
- słowa zarezerwowane, 65
- słowniki klawiaturowe, 549
- słuchacz zdarzeń, 530
- sortowanie, 470
 - quick sort, 471
 - tablica, 120
- specyfikacja wyjątku, 332
- specyfikator formatu, 91
- sprawdzanie
 - parametrów, 352
 - zakresu, 119
- spring layout, 624
- sprężenie zwrotne, 278
- stack trace, 345
- stałe, 67
 - pola klasy, 152
 - statyczne, 154
- stałe interfejsu Action, 547
 - ACCELERATOR_KEY, 547
 - ACTION_COMMAND_KEY, 547
 - DEFAULT, 547
 - NAME, 547
 - LONG_DESCRIPTION, 547
 - MNEMONIC_KEY, 547
 - SMALL_ICON, 547
 - SHORT_DESCRIPTION, 547
- stałe klasowe, 67
- stałe łańcuchowe, 79
- stałe matematyczne, 68
- stan obiektu, 131
- stany wątków, 758
- statyczne
 - funkcje składowe, 58
 - klasy wewnętrzne, 316
 - sterta, 119, 450
- STL, Standard Template Library, 420
- stos, 482
- stosowanie
 - kilku plików źródłowych, 145
 - wyjątków, 348
- strona HTML, 714
- struktury danych, 420
 - kolejka, 448
 - kolejka priorytetowa, 450
 - listy powiązane, 431
 - listy tablicowe, 440
 - mapy, 451
 - sterta, 450
 - tablice mieszające, 441
 - zbiór, 442
- strumienie, 88
- styl Metal, 538
- subclass, 194
- substitution principle, 202
- superclass, 194
- supplementary characters, 64
- surrogates area, 64
- suwaki, 600
 - podziałka, 601
- Swing, 488
 - akceleratory, 613
 - aplety, 715
 - czcionki, 517
 - dodawanie komponentów, 575
 - dymki, 620
 - etykiety, 582
 - figury 2W, 505
 - ikony, 494
 - kolejka dostępu, 647
 - kolory, 513
 - komponenty, 502, 565, 573
 - konfiguracja komponentów, 493
 - kontener, 505, 573
 - listy rozwijalne, 597
 - mapa wejścia, 549
 - menu, 606
 - menu podręczne, 611
 - motywy, 491
 - nazwy klas, 492
 - niestandardowi zarządcy rozkładu, 643
 - obramowanie, 593
 - obsługa zdarzeń, 502
 - obszary tekstowe, 584
 - okna dialogowe, 648
 - panel przewijany, 585
 - paski narzędzi, 618
 - pola haseł, 584
 - pola tekstowe, 580
 - pola wyboru, 587
 - położenie ramki, 494
 - pozycjonowanie bezwzględne, 643
 - pozycjonowanie ramki, 494

przełączniki, 590
 przyciski, 571, 623
 ramki, 492
 rozkład brzegowy, 574
 rozkład siatkowy, 576
 rozmiar ekranu, 497
 rozmiar ramki, 497
 rysowanie, 502
 style, 490
 suwaki, 600
 tekst pasku tytułu, 495
 treść, 568
 tryb pełnoekranowy, 498
 uruchamianie czasochłonnych zadań, 831
 warstwy ramki, 501
 wątek dystrybucji zdarzeń, 493
 wątki, 830
 wprowadzanie tekstu, 580
 wybór opcji, 587
 wygląd, 568
 wysyłanie zdarzeń, 493
 wyświetlanie
 informacji w komponencie, 500
 ramki, 494
 tekstu, 503
 zachowanie, 568
 zamykanie ramki aplikacji, 494
 zarządzanie rozkładem, 572, 623
 zasada jednego wątku, 832, 841
 zmiana stylu, 538
 sygnatura metody, 166, 204
 symbole zastępcze znaków specjalnych, 63
 synchronizacja wątków, 764
 synchronizatory, 827
 bariery, 829
 CountDownLatch, 828
 CyclicBarrier, 828, 829
 Exchanger, 828, 830
 semafory, 827
 Semaphore, 828
 SynchronousQueue, 828
 zatrząsk z licznikiem, 828
 system
 kolorów, 515
 usuwania nieużytków, 458
 System.err, 360, 374
 System.in, 88
 System.out, 96, 374

Ś

ścieżka
 dostępu, 38
 klas, 181
 ścisła kontrola typów, 268
 śledzenie
 przepływu wykonywania, 357
 stosu, 245, 345

średnik, 65
 środowisko
 działania apletu, 726
 programistyczne, 35

T

tabela metod, 204
 tablice, 115
 anonimowe, 117
 deklaracja, 115
 inicjalizacja, 117
 Iterable, 117
 kopiowane przy zapisie, 808
 kopiowanie, 118
 liczba elementów, 116
 tablice mieszające, 221, 441
 kolizja, 441
 komórki, 441
 kubelki, 441
 reorganizacja, 442
 współczynnik zapęnlwienia, 442
 numerowanie elementów, 116
 postrzępione, 126
 przetwarzanie, 116
 rozmiar 0, 118
 sortowanie, 120
 tworzenie, 115
 wielowymiarowe, 123
 tagging interface, 284
 tasowanie, 470
 technologia Matisse, 634
 template code bloat, 388
 testowanie blokad, 791
 this, 148, 155, 197, 307
 thread, 745
 thread of control, 745
 toolbar, 618
 toolkit-modal dialog, 660
 tooltips, 620
 tożsamość obiektu, 131
 translacja
 metod uogólnionych, 389
 wyrażeń generycznych, 389
 traversal order, 647
 treść, 568
 trigger key, 665
 try, 245, 335
 tryb pełnoekranowy, 498
 try-finally, 341, 342
 tworzenie
 akceleratorzy, 613
 algorytm, 478
 aplety, 48
 egzemplarz klasy, 131
 egzemplarz typu uogólnionego, 383
 klasy wyjątków, 334

tworzenie

- klucz URL, 738
- konstruktory, 147
- menu, 606
- metody uogólnione, 384
- obiekty, 135, 136, 165
- obiekty pośredniczące, 320
- okna dialogowe, 659
- pliki JAR, 694
- pule wątków, 815
- ramki, 492
- tablice, 115
- tablice anonimowe, 117
- tablice postrzępione, 127
- wątki, 754
- zegar, 278

type parameters, 380

typy

- interfejsowe, 422
 - MIME, 732
 - sparametryzowane, 31, 379
 - surowe, 229, 388
 - uogólnione
 - refleksja, 411
 - wieloznaczne, 381, 404
 - bez ograniczeń, 408
 - chwytywanie, 408
 - nadtypy, 405
 - ograniczenia nadtypów, 405
 - wyliczeniowe, 75
- typy danych, 60
- boolean, 64
 - całkowite, 60
 - char, 62
 - int, 26
 - long int, 26
 - liczby całkowite, 60
 - liczby zmiennoprzecinkowe, 61
 - łańcuchy, 76
 - numeryczne, 61
 - rzutowanie, 71

U

ukrywanie danych, 131

UML, Unified Modeling Language, 134

- powiązania między klasami, 134

unchecked exception, 330

Unicode, 62, 76

Update, 36

updates, 37

uruchamianie

- apletów, 48, 716
- aplikacji, 57
- aplikacji bezpośrednio z internetu, 732

- aplikacji graficznej, 46
- czasochłonnych zadań, 831
- wątku, 752

usługi, 712

ustawianie ścieżki klas, 184

usuwanie błędów, 372

UTC, 138

UTF-16, 64

V

varargs, 238, 357

W

wartości logiczne, 64

warunki, 772

wątek

- dystribucji zdarzeń, 493, 795
- sterowania, 745

wątki, 746, 747

- bariery, 829
- BLOCKED, 759
- blok synchronizowany, 781
- blokada, 771
- blokada odczytu-zapisu, 793
- blokowanie po stronie klienta, 782
- Callable, 810
- Condition, 777
- demony, 762
- Executors, 815
- Future, 810, 811
- kolejka blokująca, 795
- kolekcje bezpieczne wątkowo, 802
- Lock, 769, 772, 777
- monitor, 783
- NEW, 758
- nieprzechwycone, 762
- oczekujące, 774
- odmierzanie czasu, 791
- poła ulotne, 783
- priorytety, 761
- przerywanie, 755
- pule, 815
- ReentrantLock, 772, 793
- ReentrantReadWriteLock, 793
- run(), 754
- RUNNABLE, 758
- semafory, 827
- signal(), 775
- signalAll(), 774, 775
- stany, 758
- status przerwania, 755
- stop(), 794
- suspend(), 794
- Swing, 830

- synchronizacja, 764
- synchronizatory, 827
- synchronized, 777
- tablice kopiowane przy zapisie, 808
- TERMINATED, 758
- testowanie blokad, 791
- Thread, 754
- tworzenie, 754
- uruchamianie, 752
- volatile, 784
- WAITING, 759
- warunki, 772
- wykonywanie zadań w osobnych wątkach, 751
- wyścig, 764, 768
- wyłaszczanie, 758
- wyzerowanie statusu przzerwania, 756
- zablokowane, 755, 759
- zakleszczenia, 774, 787
- zamykanie, 759
- zatrzymanie wykonywania, 751
- weak reference, 458
- weakly consistent iterators, 802
- wejście, 88
- wersje języka Java, 32
- wiązanie
 - dynamiczne, 199
 - statyczne, 203
- widoczność, 205
- widoki, 476, 568
 - kontrolowane, 465
 - niemodyfikowalne, 464
 - przedziałowe, 463
 - synchronizowane, 465
- wielkie liczby, 113
- wielkość liter, 56
- wielokropek, 239
- wielowątkowość, 27
- wiersz poleceń, 41, 119
- wildcard type, 381, 404
- własności interfejsów, 272
- właściwości
 - klas pośredniczących, 324
 - list, 471
- włączanie asercji, 352
- wprowadzanie tekstu, 580
- wskaźniki do metod, 258
- współbieżność, 745
- współczynnik zapelnienia, 442
- współrzędne, 508
 - kodowe znaków, 64, 80
- wstawianie komentarzy javadoc, 185
- wybór
 - kolorów, 679
 - opcji, 587
 - plików, 669
- wyciszanie wyjątków, 349
- wydajność kodu bajtowego, 27
- wygląd, 568
- wyjątki, 328, 329
 - analiza danych ze śledzenia stosu, 345
 - ArrayIndexOutOfBoundsException, 116
 - ArrayIndexOutOfBoundsException, 330, 331
 - BadCastException, 411
 - blok try-catch, 335
 - blok try-finally, 341, 342
 - BrokenBarrierException, 829
 - budowanie łańcuchów, 339
 - CancellationException, 839
 - ClassCastException, 207, 256, 403, 466
 - CloneNotSupportedException, 285
 - ConcurrentModificationException, 435, 802
 - diagram hierarchii, 329
 - EmptyStackException, 350
 - Error, 330
 - Exception, 330
 - FileNotFoundException, 331, 333, 738
 - finally, 340
 - IllegalAccessException, 252
 - IllegalStateException, 425
 - InterruptedException, 747, 752, 755
 - IOException, 333, 336
 - klasy wyjątków, 334
 - klasyfikacja, 329
 - kontrolowane, 243, 245, 330–332
 - metody, 331
 - niekontrolowane, 245, 330
 - NullPointerException, 330, 350
 - NumberFormatException, 349
 - opis, 335
 - powtarzane generowanie, 339
 - przechwytywanie, 244, 335, 337
 - przekazywanie, 350
 - RuntimeException, 330, 349
 - specyfikacja, 332
 - stosowanie, 348
 - Throwable, 329
 - TimeoutException, 811
 - tworzenie klas wyjątków, 334
 - UnsupportedOperationException, 462, 464
 - wyciszanie, 349
 - wykonawcze, 330
 - zamykanie zasobów, 341
 - zgłaszanie, 333
- wykonywalne pliki JAR, 696
- wykonywanie zadań w osobnych wątkach, 751
- wyliczenia, 240, 480
- wyłączanie
 - asercji, 352
 - dziedziczenia, 205
- wymazywanie typów, 389
- wypełnianie figur, 513
- wypisywanie danych, 59
- wrażenia lambda, 288
- wysyłanie zdarzeń, 493

- wyszukiwanie
 - binarne, 473
 - błędów, 372
- wyścig, 764, 768
- wyświetlanie
 - informacji w komponencie, 500
 - obrazów, 525
 - ramki, 494
 - tekstu, 503
- wyłączanie wątków, 758
- wywołanie
 - innego konstruktora, 169
 - przez nazwę, 159
 - przez referencję, 159
 - przez wartość, 159
- wyzerowanie statusu przerywania wątku, 756
- wzorce projektowe, 566
- wzorzec
 - Composite, 567
 - Decorator, 567
 - Strategy, 567

X

- X11, 502
- XML, 32

Z

- zablokowane wątki, 759
- zachowanie obiektu, 131
- zadanie, 754
- zakleszczenia, 774, 787
- zakres
 - dostępności zmiennych, 295
 - typów caloitych, 60
- zależność, 133
- zamiana parametrów obiektowych, 163
- zamykanie
 - aplikacji, 494
 - wątków, 759
 - zasobów, 341
- zaokrąglenie liczb, 71
- zapis
 - do dziennika, 355
 - do pliku, 95
 - preferencji użytkownika, 701
- zarządca rozkładu
 - brzegowego, 574
 - ciągłego, 572
 - grupowego, 624
 - siatkowego, 576
- zarządzanie rozkładem, 572, 623
 - GridBagLayout, 624
- zasada
 - jednego wątku, 832, 841
 - zamienialności, 202

- zasięg
 - blokowy, 97
 - pakietów, 180
 - zmiennych, 97
- zasoby, 697
- zatrząsk z licznikiem, 828
- zbiory bitów, 482
- zbiór, 442
 - HashSet, 440
 - TreeSet, 444
- zdarzenia, 279, 529
 - AWT, 560
 - ChangeEvent, 601
 - interfejs nasłuchu, 530
 - klasy adaptacyjne, 542
 - mysz, 553
 - niskiego poziomu, 561
 - obiekty nasłuchujące, 279
 - obsługa, 279
 - okna, 542
 - procedura obsługi, 529
 - semantyczne, 561
 - sluchacz, 530
 - WindowEvent, 542
 - źródło, 530
- zegar, 278
- zgłaszanie wyjątków, 333
- zintegrowane środowisko programistyczne, IDE, 43
- zmiana
 - stanu okna, 561
 - stylu, 538
- zmienna liczba parametrów, 238
- zmiennne, 65
 - atomowe, 785
 - definicja, 66
 - deklaracja, 65, 66
 - finalne, 785
 - interfejsowe, 272
 - lokalne wątków, 790
 - nazwy, 65
 - obiektywne, 135
 - polimorficzne, 202
 - środowiskowe
 - CLASSPATH, 184
 - warunkowe, 772
 - zasięg, 97
- zmiennne typowe, 385
 - ograniczenia, 386
 - statyczny kontekst klas uogólnionych, 398
- znacznik
 - @author, 185, 187
 - @deprecated, 187
 - @link, 188
 - @Override, 219
 - @param, 185, 186
 - @return, 186
 - @see, 188
 - @since, 187

@SuppressWarnings, 392
@throws, 186
@version, 187
<applet>, 50, 717
align, 719
alt, 719
archive, 719
code, 718, 719
codebase, 719
height, 719
hspace, 719
name, 720
object, 719
vspace, 719
width, 719
<param>, 720

znaczniki dokumentacyjne, 185
znajdowanie liczb pierwszych, 483
znak
@, 185
echa, 584
wielokropka, 239
znaki
dodatkowe, 64
konwersji, 91
specjalne, 63
Unicode, 65
zwolnienie klawisza, 561

Ż

źródło zdarzeń, 530

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**

POZNAJ JAVĘ

— IDEALNY JĘZYK DLA NAJLEPSZYCH PROGRAMISTÓW!

Niemal od początku swojego istnienia Java była traktowana jako wszechstronne narzędzie do budowy bezpiecznych aplikacji internetowych. Zaaprobowała ją znacząca większość poważnych firm informatycznych. Dziś jest uważana za niezwykle starannie zaprojektowany i rozwijany język, w który wbudowano wyrafinowane funkcje ułatwiające implementację wielu złożonych zadań programistycznych. Aby w pełni wykorzystać możliwości Javy, trzeba zrozumieć rządzące nią zasady i poznać jej zaawansowane cechy.

Książka, którą trzymasz w dłoni, to poważny podręcznik dla poważnych programistów. Opisano tu podstawy języka oraz najważniejsze zagadnienia związane z programowaniem interfejsu użytkownika. Przedstawiono pakiet Java Development Kit. Pakiet ten obejmuje obecnie tak różne aspekty tworzenia aplikacji, jak konstruowanie interfejsu użytkownika, zarządzanie bazami danych, internacjonalizacja, bezpieczeństwo i przetwarzanie XML. W książce znajdziesz również mnóstwo przykładów kodu obrazujących zasady działania niemal każdej opisywanej funkcji i biblioteki.

NAJWAŻNIEJSZE ZAGADNIENIA:

- programowanie obiektowe
- mechanizm refleksji i obiekty proxy
- interfejsy i klasy wewnętrzne
- system kolekcji
- projektowanie graficznego interfejsu użytkownika za pomocą pakietu narzędzi Swing UI
- współbieżność

CAY S. HORSTMANN — jest autorem wielu książek dotyczących programowania w Javie, przeznaczonych dla zawodowych programistów oraz studentów informatyki. Jest profesorem informatyki na Uniwersytecie Stanowym w San Jose. Zdobył tytuł Java Champion. Często wygłasza referaty na konferencjach informatycznych.



księgarnia internetowa

<http://helion.pl>

zamówienia telefoniczne



0 801 339900



0 601 339900

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

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

PRENTICE HALL
PEARSON EDUCATION

ISBN 978-83-283-2480-0



9 788328 324800

cena: 99,00 zł

Informatyka w najlepszym wydaniu

