

WYKORZYSTAJ POTENCJAŁ LIDERA
NA RYNKU JĘZYKÓW PROGRAMOWANIA!

 PRENTICE
HALL

JAVA™

Podstawy

WYDANIE IX



CAY S. HORSTMANN · GARY CORNELL

 Helion

Tytuł oryginału: Core Java Volume I--Fundamentals (9th Edition)

Tłumaczenie: Łukasz Piwko

ISBN: 978-83-246-7758-0

Authorized translation from the English language edition, entitled CORE JAVA VOLUME I – FUNDAMENTALS, 9TH EDITION; ISBN 0137081898; by Cay S. Horstmann; and Gary Cornell; published by Pearson Education, Inc, publishing as Prentice Hall. Copyright © 2013 by Oracle and/or its affiliates, 500 Oracle Parkway, Redwood Shores, CA 94065.

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 © 2013.

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.

Wydawnictwo HELION dołożyło wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie bierze jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Wydawnictwo HELION nie ponosi 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/javpd9.zip>

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/javpd9>

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
Rozdział 1. Wstęp do Javy	21
1.1. Java jako platforma programistyczna	21
1.2. Słowa klucze białej księgi Javy	22
1.2.1. Prosty	23
1.2.2. Obiektowy	24
1.2.3. Sieciowy	24
1.2.4. Niezawodny	24
1.2.5. Bezpieczny	25
1.2.6. Niezależny od architektury	26
1.2.7. Przenośny	26
1.2.8. Interpretowany	27
1.2.9. Wysokowydajny	27
1.2.10. Wielowątkowy	28
1.2.11. Dynamiczny	28
1.3. Aplety Javy i internet	28
1.4. Krótka historia Javy	30
1.5. Główne nieporozumienia dotyczące Javy	32
Rozdział 2. Środowisko programistyczne Javy	37
2.1. Instalacja oprogramowania Java Development Kit	38
2.1.1. Pobieranie pakietu JDK	38
2.1.2. Ustawianie ścieżki dostępu	39
2.1.3. Instalacja bibliotek i dokumentacji	41
2.1.4. Instalacja przykładowych programów	42
2.1.5. Drzewo katalogów Javy	42
2.2. Wybór środowiska programistycznego	43
2.3. Używanie narzędzi wiersza poleceń	44
2.3.1. Rozwiązywanie problemów	45
2.4. Praca w zintegrowanym środowisku programistycznym	47
2.4.1. Znajdowanie błędów kompilacji	49

2.5.	Uruchamianie aplikacji graficznej	50
2.6.	Tworzenie i uruchamianie apletów	53

Rozdział 3. Podstawowe elementy języka Java 57

3.1.	Prosty program w Javie	58
3.2.	Komentarze	61
3.3.	Typy danych	62
3.3.1.	Typy całkowite	62
3.3.2.	Typy zmiennoprzecinkowe	63
3.3.3.	Typ char	65
3.3.4.	Typ boolean	66
3.4.	Zmienne	66
3.4.1.	Inicjacja zmiennych	68
3.4.2.	Stałe	68
3.5.	Operatory	69
3.5.1.	Operatory inkrementacji i dekrementacji	71
3.5.2.	Operatory relacyjne i logiczne	71
3.5.3.	Operatory bitowe	72
3.5.4.	Funkcje i stałe matematyczne	73
3.5.5.	Konwersja typów numerycznych	74
3.5.6.	Rzutowanie	75
3.5.7.	Nawiasy i priorytety operatorów	76
3.5.8.	Typ wyliczeniowy	77
3.6.	Łańcuchy	77
3.6.1.	Podłańcuchy	78
3.6.2.	Konkatenacja	78
3.6.3.	Łańcuchów nie można modyfikować	79
3.6.4.	Porównywanie łańcuchów	79
3.6.5.	Łańcuchy puste i łańcuchy null	81
3.6.6.	Współrzędne kodowe znaków i jednostki kodowe	81
3.6.7.	API String	83
3.6.8.	Dokumentacja API w internecie	85
3.6.9.	Składanie łańcuchów	86
3.7.	Wejście i wyjście	89
3.7.1.	Odbieranie danych wejściowych	89
3.7.2.	Formatowanie danych wyjściowych	91
3.7.3.	Zapis i odczyt plików	96
3.8.	Przeływ sterowania	98
3.8.1.	Zasięg blokowy	98
3.8.2.	Instrukcje warunkowe	99
3.8.3.	Pętle	101
3.8.4.	Pętle o określonej liczbie powtórzeń	106
3.8.5.	Wybór wielokierunkowy — instrukcja switch	109
3.8.6.	Instrukcje przerywające przepływ sterowania	111
3.9.	Wielkie liczby	114
3.10.	Tablice	116
3.10.1.	Pętla typu for each	117
3.10.2.	Inicjowanie tablic i tworzenie tablic anonimowych	118
3.10.3.	Kopiowanie tablicy	119
3.10.4.	Parametry wiersza poleceń	120
3.10.5.	Sortowanie tablicy	121

3.10.6.	Tablice wielowymiarowe	124
3.10.7.	Tablice postrzępione	127
Rozdział 4.	Obiekty i klasy	131
4.1.	Wstęp do programowania obiektowego	132
4.1.1.	Klasy	132
4.1.2.	Obiekty	133
4.1.3.	Identyfikacja klas	134
4.1.4.	Relacje między klasami	135
4.2.	Używanie klas predefiniowanych	137
4.2.1.	Obiekty i zmienne obiektów	137
4.2.2.	Klasa GregorianCalendar	139
4.2.3.	Metody udostępniające i zmieniające wartość elementu	141
4.3.	Definiowanie własnych klas	148
4.3.1.	Klasa Employee	148
4.3.2.	Używanie wielu plików źródłowych	151
4.3.3.	Analiza klasy Employee	151
4.3.4.	Pierwsze kroki w tworzeniu konstruktorów	152
4.3.5.	Parametry jawne i niejawne	153
4.3.6.	Korzyści z hermetyzacji	155
4.3.7.	Przywileje klasowe	157
4.3.8.	Metody prywatne	157
4.3.9.	Stałe jako pola klasy	158
4.4.	Pola i metody statyczne	158
4.4.1.	Pola statyczne	159
4.4.2.	Stałe statyczne	159
4.4.3.	Metody statyczne	160
4.4.4.	Metody fabryczne	161
4.4.5.	Metoda main	162
4.5.	Parametry metod	164
4.6.	Konstruowanie obiektów	171
4.6.1.	Przeciążanie	171
4.6.2.	Inicjacja pól wartościami domyślnymi	171
4.6.3.	Konstruktor bezargumentowy	172
4.6.4.	Jawna inicjacja pól	172
4.6.5.	Nazywanie parametrów	174
4.6.6.	Wywoływanie innego konstruktora	174
4.6.7.	Bloki inicjujące	175
4.6.8.	Niszczenie obiektów i metoda finalize	179
4.7.	Pakiety	180
4.7.1.	Importowanie klas	180
4.7.2.	Importy statyczne	182
4.7.3.	Dodawanie klasy do pakietu	182
4.7.4.	Zasięg pakietów	185
4.8.	Ścieżka klas	187
4.8.1.	Ustawianie ścieżki klas	189
4.9.	Komentarze dokumentacyjne	190
4.9.1.	Wstawianie komentarzy	190
4.9.2.	Komentarze do klas	191
4.9.3.	Komentarze do metod	191
4.9.4.	Komentarze do pól	192

4.9.5.	Komentarze ogólne	192
4.9.6.	Komentarze do pakietów i ogólne	194
4.9.7.	Generowanie dokumentacji	194
4.10.	Porady dotyczące projektowania klas	195
Rozdział 5. Dziedziczenie		199
5.1.	Klasy, nadklasy i podklasy	200
5.1.1.	Hierarchia dziedziczenia	206
5.1.2.	Polimorfizm	207
5.1.3.	Wiązanie dynamiczne	209
5.1.4.	Wyłączanie dziedziczenia — klasy i metody finalne	211
5.1.5.	Rzutowanie	212
5.1.6.	Klasy abstrakcyjne	214
5.1.7.	Ochrona dostępu	219
5.2.	Klasa bazowa Object	220
5.2.1.	Metoda equals	221
5.2.2.	Porównywanie a dziedziczenie	222
5.2.3.	Metoda hashCode	225
5.2.4.	Metoda toString	228
5.3.	Generyczne listy tablicowe	233
5.3.1.	Dostęp do elementów listy tablicowej	236
5.3.2.	Zgodność pomiędzy typowanymi a surowymi listami tablicowymi	239
5.4.	Ostroje obiektów i autoboxing	241
5.5.	Metody ze zmienną liczbą parametrów	244
5.6.	Klasy wyliczeniowe	245
5.7.	Refleksja	247
5.7.1.	Klasa Class	248
5.7.2.	Podstawy przechwytywania wyjątków	250
5.7.3.	Zastosowanie refleksji w analizie funkcjonalności klasy	252
5.7.4.	Refleksja w analizie obiektów w czasie działania programu	257
5.7.5.	Zastosowanie refleksji w generycznym kodzie tablicowym	261
5.7.6.	Wywoływanie dowolnych metod	264
5.8.	Porady projektowe dotyczące dziedziczenia	268
Rozdział 6. Interfejsy i klasy wewnętrzne		271
6.1.	Interfejsy	272
6.1.1.	Własności interfejsów	276
6.1.2.	Interfejsy a klasy abstrakcyjne	279
6.2.	Klonowanie obiektów	280
6.3.	Interfejsy a sprzężenie zwrotne	286
6.4.	Klasy wewnętrzne	289
6.4.1.	Dostęp do stanu obiektu w klasie wewnętrznej	289
6.4.2.	Specjalne reguły składniowe dotyczące klas wewnętrznych	293
6.4.3.	Czy klasy wewnętrzne są potrzebne i bezpieczne?	294
6.4.4.	Lokalne klasy wewnętrzne	296
6.4.5.	Dostęp do zmiennych finalnych z metod zewnętrznych	297
6.4.6.	Anonimowe klasy wewnętrzne	300
6.4.7.	Statyczne klasy wewnętrzne	303
6.5.	Klasy proxy	306
6.5.1.	Własności klas proxy	311

Rozdział 7. Grafika	313
7.1. Wprowadzenie do pakietu Swing	314
7.2. Tworzenie ramki	318
7.3. Pozycjonowanie ramki	321
7.3.1. Własności ramek	322
7.3.2. Określanie rozmiaru ramki	323
7.4. Wyświetlanie informacji w komponencie	327
7.5. Figury 2D	332
7.6. Kolory	340
7.7. Czcionki	343
7.8. Wyświetlanie obrazów	351
Rozdział 8. Obsługa zdarzeń	355
8.1. Podstawy obsługi zdarzeń	355
8.1.1. Przykład — obsługa kliknięcia przycisku	357
8.1.2. Nabywanie biegłości w posługiwaniu się klasami wewnętrznymi	362
8.1.3. Tworzenie słuchaczy zawierających jedno wywołanie metody	364
8.1.4. Przykład — zmiana stylu	366
8.1.5. Klasy adaptacyjne	369
8.2. Akcje	373
8.3. Zdarzenia generowane przez mysz	380
8.4. Hierarchia zdarzeń w bibliotece AWT	387
8.4.1. Zdarzenia semantyczne i niskiego poziomu	388
Rozdział 9. Komponenty Swing interfejsu użytkownika	391
9.1. Swing a wzorzec projektowy Model-View-Controller	392
9.1.1. Wzorce projektowe	392
9.1.2. Wzorzec Model-View-Controller	393
9.1.3. Analiza MVC przycisków Swing	397
9.2. Wprowadzenie do zarządzania rozkładem	398
9.2.1. Rozkład brzegowy	400
9.2.2. Rozkład siatkowy	402
9.3. Wprowadzanie tekstu	406
9.3.1. Pola tekstowe	406
9.3.2. Etykiety komponentów	408
9.3.3. Pola haseł	410
9.3.4. Obszary tekstowe	410
9.3.5. Panele przewijane	411
9.4. Komponenty umożliwiające wybór opcji	413
9.4.1. Pola wyboru	413
9.4.2. Przełączniki	415
9.4.3. Obramowanie	419
9.4.4. Listy rozwijalne	423
9.4.5. Suwaki	426
9.5. Menu	432
9.5.1. Tworzenie menu	432
9.5.2. Ikony w elementach menu	435
9.5.3. Pola wyboru i przełączniki jako elementy menu	436
9.5.4. Menu podręczne	437
9.5.5. Mnemoniki i akceleratory	438

9.5.6.	Aktywowanie i dezaktywowanie elementów menu	440
9.5.7.	Paski narzędzi	444
9.5.8.	Dymki	446
9.6.	Zaawansowane techniki zarządzania rozkładem	448
9.6.1.	Rozkład GridBagLayout	449
9.6.2.	Rozkład grupowy	459
9.6.3.	Nieużywanie żadnego zarządcy rozkładu	468
9.6.4.	Niestandardowi zarządcy rozkładu	469
9.6.5.	Kolejka dostępu	472
9.7.	Okna dialogowe	474
9.7.1.	Okna dialogowe opcji	474
9.7.2.	Tworzenie okien dialogowych	484
9.7.3.	Wymiana danych	489
9.7.4.	Okna dialogowe wyboru plików	495
9.7.5.	Okna dialogowe wyboru kolorów	505
Rozdział 10. Przygotowywanie appletów i aplikacji do użytku		511
10.1.	Pliki JAR	512
10.1.1.	Manifest	512
10.1.2.	Wykonywalne pliki JAR	514
10.1.3.	Zasoby	515
10.1.4.	Pieczątowanie pakietów	518
10.2.	Java Web Start	519
10.2.1.	Piaskownica	522
10.2.2.	Podpisywanie kodu	523
10.2.3.	API JNLP	525
10.3.	Aplety	533
10.3.1.	Prosty applet	533
10.3.2.	Znacznik applet i jego atrybuty	537
10.3.3.	Znacznik object	540
10.3.4.	Parametry przekazujące informacje do appletów	541
10.3.5.	Dostęp do obrazów i plików audio	546
10.3.6.	Środowisko działania appletu	547
10.4.	Zapisywanie preferencji użytkownika	549
10.4.1.	Mapy własności	550
10.4.2.	API Preferences	555
Rozdział 11. Wyjątki, dzienniki, asercje i debugowanie		563
11.1.	Obsługa błędów	564
11.1.1.	Klasyfikacja wyjątków	565
11.1.2.	Deklarowanie wyjątków kontrolowanych	567
11.1.3.	Zgłaszanie wyjątków	569
11.1.4.	Tworzenie klas wyjątków	570
11.2.	Przechwytywanie wyjątków	571
11.2.1.	Przechwytywanie wielu typów wyjątków	574
11.2.2.	Powtórne generowanie wyjątków i budowanie łańcuchów wyjątków	575
11.2.3.	Klauzula finally	576
11.2.4.	Instrukcja try z zasobami	580
11.2.5.	Analiza danych ze śledzenia stosu	581
11.3.	Wskazówki dotyczące stosowania wyjątków	584

11.4.	Asercje	587
11.4.1.	Włączanie i wyłączanie asercji	588
11.4.2.	Zastosowanie asercji do sprawdzania parametrów	589
11.4.3.	Zastosowanie asercji do dokumentowania założeń	590
11.5.	Dzienniki	591
11.5.1.	Podstawy zapisu do dziennika	592
11.5.2.	Zaawansowane techniki zapisu do dziennika	592
11.5.3.	Zmiana konfiguracji menedżera dzienników	594
11.5.4.	Lokalizacja	596
11.5.5.	Obiekty typu Handler	596
11.5.6.	Filtry	600
11.5.7.	Formatery	600
11.5.8.	Przepis na dziennik	601
11.6.	Wskazówki dotyczące debugowania	609
11.7.	Wskazówki dotyczące debugowania aplikacji z GUI	614
11.7.1.	Zaprzęgnięcie robota AWT do pracy	617
11.8.	Praca z debugerem	621

Rozdział 12. Programowanie ogólne 627

12.1.	Dlaczego programowanie ogólne	628
12.1.1.	Dla kogo programowanie ogólne	629
12.2.	Definicja prostej klasy ogólnej	630
12.3.	Metody ogólne	632
12.4.	Ograniczenia zmiennych typowych	633
12.5.	Kod ogólny a maszyna wirtualna	635
12.5.1.	Translacja wyrażeń generycznych	637
12.5.2.	Translacja metod ogólnych	637
12.5.3.	Używanie starego kodu	639
12.6.	Ograniczenia i braki	641
12.6.1.	Nie można podawać typów prostych jako parametrów typowych	641
12.6.2.	Sprawdzanie typów w czasie działania programu jest możliwe tylko dla typów surowych	641
12.6.3.	Nie można tworzyć tablic typów ogólnych	642
12.6.4.	Ostrzeżenia dotyczące zmiennej liczby argumentów	642
12.6.5.	Nie wolno tworzyć egzemplarzy zmiennych typowych	643
12.6.6.	Zmiennych typowych nie można używać w statycznych kontekstach klas ogólnych	645
12.6.7.	Obiektów klasy ogólnej nie można generować ani przechwytywać	646
12.6.8.	Uważaj na konflikty, które mogą powstać po wymazaniu typów	648
12.7.	Zasady dziedziczenia dla typów ogólnych	649
12.8.	Typy wieloznaczne	650
12.8.1.	Ograniczenia nadtypów typów wieloznacznych	652
12.8.2.	Typy wieloznaczne bez ograniczeń	655
12.8.3.	Chwyatanie typu wieloznacznego	655
12.9.	Refleksja a typy ogólne	658
12.9.1.	Zastosowanie parametrów Class<T> do dopasowywania typów	659
12.9.2.	Informacje o typach generycznych w maszynie wirtualnej	659

Rozdział 13. Kolekcje 665

13.1.	Interfejsy kolekcyjne	665
13.1.1.	Oddzielenie warstwy interfejsów od warstwy klas konkretnych	666
13.1.2.	Interfejsy Collection i Iterator	668

13.2.	Konkretne klasy kolekcyjne	674
13.2.1.	Listy powiązane	674
13.2.2.	Listy tablicowe	684
13.2.3.	Zbiór HashSet	684
13.2.4.	Zbiór TreeSet	688
13.2.5.	Porównywanie obiektów	689
13.2.6.	Kolejki Queue i Deque	694
13.2.7.	Kolejki priorytetowe	696
13.2.8.	Mapy	697
13.2.9.	Specjalne klasy Set i Map	702
13.3.	Architektura kolekcji	706
13.3.1.	Widoki i obiekty opakowujące	709
13.3.2.	Operacje zbiorcze	717
13.3.3.	Konwersja pomiędzy kolekcjami a tablicami	718
13.4.	Algorytmy	718
13.4.1.	Sortowanie i tasowanie	720
13.4.2.	Wyszukiwanie binarne	722
13.4.3.	Proste algorytmy	723
13.4.4.	Pisanie własnych algorytmów	725
13.5.	Stare kolekcje	726
13.5.1.	Klasa Hashtable	726
13.5.2.	Wyliczenia	727
13.5.3.	Mapy własności	728
13.5.4.	Stosy	729
13.5.5.	Zbiory bitów	729

Rozdział 14. Wielowątkowość **735**

14.1.	Czym są wątki	736
14.1.1.	Wykonywanie zadań w osobnych wątkach	741
14.2.	Przerywanie wątków	746
14.3.	Stany wątków	749
14.3.1.	Wątki NEW	749
14.3.2.	Wątki RUNNABLE	750
14.3.3.	Wątki BLOCKED i WAITING	750
14.3.4.	Zamykanie wątków	752
14.4.	Własności wątków	752
14.4.1.	Priorytety wątków	752
14.4.2.	Wątki demony	753
14.4.3.	Procedury obsługi nieprzechwyconych wyjątków	754
14.5.	Synchronizacja	756
14.5.1.	Przykład sytuacji powodującej wyścig	756
14.5.2.	Wyścigi	760
14.5.3.	Obiekty klasy Lock	762
14.5.4.	Warunki	765
14.5.5.	Słowo kluczowe synchronized	769
14.5.6.	Bloki synchronizowane	774
14.5.7.	Monitor	775
14.5.8.	Pola ulotne	776
14.5.9.	Zmienne finalne	777
14.5.10.	Zmienne atomowe	777
14.5.11.	Zakleszczenia	778

14.5.12. Zmienne lokalne wątków	781
14.5.13. Testowanie blokad i odmierzenie czasu	782
14.5.14. Blokady odczytu-zapisu	783
14.5.15. Dlaczego metody stop i suspend są wycofywane	784
14.6. Kolejki blokujące	786
14.7. Kolekcje bezpieczne wątkowo	794
14.7.1. Szybkie mapy, zbiory i kolejki	794
14.7.2. Tablice kopiowane przy zapisie	796
14.7.3. Starsze kolekcje bezpieczne wątkowo	796
14.8. Interfejsy Callable i Future	797
14.9. Klasa Executors	802
14.9.1. Pule wątków	803
14.9.2. Planowanie wykonywania	807
14.9.3. Kontrolowanie grup zadań	808
14.9.4. Szkielet rozgałęzienie-złączenie	809
14.10. Synchronizatory	812
14.10.1. Semafony	812
14.10.2. Klasa CountdownLatch	813
14.10.3. Bariery	814
14.10.4. Klasa Exchanger	814
14.10.5. Kolejki synchroniczne	815
14.11. Wątki a biblioteka Swing	815
14.11.1. Uruchamianie czasochłonnych zadań	816
14.11.2. Klasa SwingWorker	820
14.11.3. Zasada jednego wątku	827
Dodatek A. Słowa kluczowe Javy	829
Skorowidz	831

7

Grafika

W tym rozdziale:

- Wprowadzenie do biblioteki Swing
- Tworzenie ramek
- Pozycjonowanie ramek
- Wyświetlanie informacji w komponencie
- Figury 2D
- Kolory
- Kroje czcionek
- Wyświetlanie obrazów

Do tej pory tworzyliśmy tylko programy, które pobierały dane z klawiatury, przetwarzały je i wyświetlały wyniki w konsoli. Większość użytkowników oczekuje jednak nieco więcej. Ani nowoczesne programy, ani strony internetowe nie działają w ten sposób. W tym rozdziale zaczynamy naukę pisania programów z graficznym interfejsem użytkownika (GUI). Nauczymy się przede wszystkim ustawiać rozmiar i położenie okien na ekranie, wyświetlać tekst pisany różnymi krojami czcionki, wyświetlać obrazy itd. Cały zdobyty tu warsztat przyda nam się w kolejnych rozdziałach, w których będziemy tworzyć ciekawe projekty.

Dwa następne rozdziały opisują przetwarzanie zdarzeń, takie jak wciśnięcie klawisza na klawiaturze lub kliknięcie przyciskiem myszy, oraz dodawanie do aplikacji takich elementów interfejsu jak menu i przyciski. Po zapoznaniu się z tymi trzema rozdziałami będziesz dysponować wiedzą, która jest niezbędna do pisania aplikacji graficznych. Bardziej zaawansowane techniki związane z programowaniem grafiki zostały opisane w drugim tomie.

Osoby zainteresowane wyłącznie programowaniem po stronie serwera, które nie mają w planach pisania GUI, mogą bezpiecznie pominąć te rozdziały.

7.1. Wprowadzenie do pakietu Swing

W Java 1.0 udostępniono bibliotekę klas o nazwie Abstract Window Toolkit (AWT), która dostarczała podstawowe narzędzia do programowania GUI. Podstawowa biblioteka AWT przerzuca zadania dotyczące tworzenia elementów interfejsu oraz obsługi ich zachowań na natywne narzędzia GUI danej platformy (Windows, Solaris, Mac OS X itd.). Jeśli na przykład przy użyciu oryginalnej biblioteki AWT umieszczono w oknie pole tekstowe, dane wprowadzane do niego były w rzeczywistości obsługiwane przez odpowiednik tego pola w systemie. Dzięki temu program napisany w Javie mógł teoretycznie działać na dowolnej platformie, a jego styl był taki sam jak innych programów w danym systemie. Stąd wziął się slogan firmy Sun: „Napisz raz, uruchamiaj wszędzie”.

Metoda programowania oparta na odpowiednikach sprawdzała się w przypadku prostych aplikacji. Natomiast szybko wyszło na jaw, że napisanie wysokiej jakości przenośnej biblioteki graficznej opartej na natywnych elementach interfejsu użytkownika jest niezwykle trudnym zadaniem. Elementy interfejsu, jak menu, paski przewijania i pola tekstowe, mogą się nieco różnić na różnych platformach. Przez to trudno było stworzyć program działający identycznie w różnych systemach. Ponadto niektóre środowiska graficzne (jak np. X11/Motif) nie dysponują tak szeroką gamą elementów interfejsu jak systemy Windows czy Mac OS X. Ten fakt ograniczał zasobność przenośnej biblioteki do tych elementów, które można znaleźć na wszystkich platformach. W wyniku tego aplikacje GUI budowane na bazie AWT ustępowały wyglądem i funkcjonalnością natywnym aplikacjom takich systemów jak Windows czy Mac OS X. Na domiar złego na różnych platformach biblioteka AWT zawierała różne błędy. Programiści narzekali, że muszą testować swoje aplikacje na wszystkich platformach, co złośliwie nazywano „napisz raz, testuj wszędzie”.

W 1996 roku firma Netscape stworzyła bibliotekę GUI o nazwie IFC (ang. *Internet Foundation Classes*), w której zastosowano zupełnie inne podejście. Elementy interfejsu użytkownika, jak przyciski, menu itd., **były rysowane** w pustym oknie. Rola systemu polegała tylko na wyświetlaniu okien i rysowaniu w nich. Dzięki temu widżety firmy Netscape wyglądały i działały zawsze tak samo, bez względu na platformę. Firma Sun podjęła współpracę z Netscape w celu udoskonalenia tej technologii, czego owocem była biblioteka o nazwie kodowej Swing. Biblioteka ta została udostępniona w postaci dodatku w Java 1.1, a do biblioteki standardowej wcielono ją w Java SE 1.2.

Zgodnie z myślą Duke’a Ellingtona, że „Nic nie ma znaczenia, jeśli jest pozbawione swingu”, Swing stał się oficjalną nazwą zestawu narzędzi do tworzenia GUI, niewykorzystującego systemowych odpowiedników elementów. Swing wchodzi w skład Java Foundation Classes (JFC). Biblioteka JFC jest bardzo duża i zawiera wiele innych narzędzi poza Swingiem. Zaliczają się do nich interfejsy API dostępności, grafiki dwuwymiarowej oraz obsługi operacji przeciągania i upuszczania.

Oczywiście elementy interfejsu oparte na Swingu pojawiają się z pewnym opóźnieniem w stosunku do komponentów używanych przez AWT. Z naszego doświadczenia wynika, że różnica ta nie powinna stanowić problemu na żadnym w miarę niezbyt starym sprzęcie. Z drugiej strony argumenty przemawiające za użyciem Swinga są przytłaczające:

- Swing udostępnia bogaty i wygodny w użyciu zestaw elementów interfejsu użytkownika.



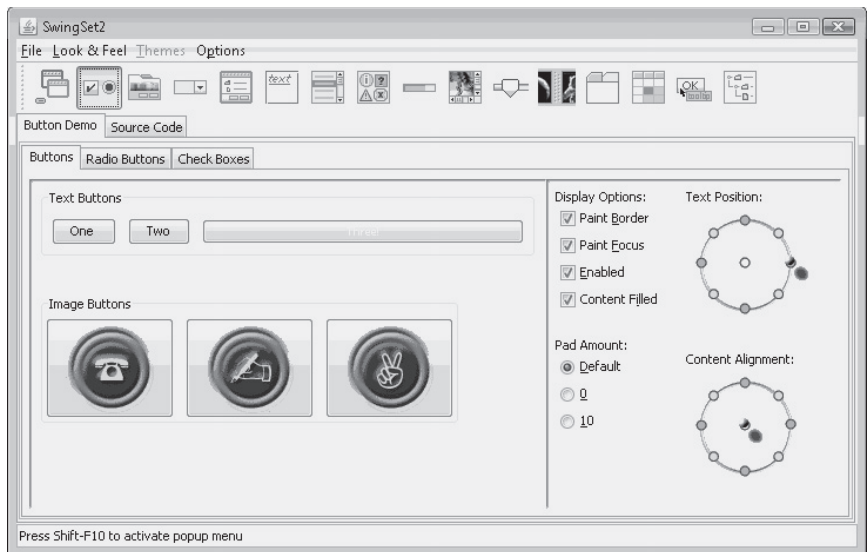
Biblioteka Swing nie zastąpiła AWT, tylko została zbudowana w oparciu o architekturę swojej poprzedniczki. Komponenty Swing oferują znacznie większe możliwości. Używając biblioteki Swing, zawsze korzysta się z podstawowej biblioteki AWT, zwłaszcza przy obsłudze zdarzeń. Od tej pory pisząc „Swing”, mamy na myśli klasy rysujące interfejs użytkownika, a pisząc „AWT”, myślimy o podstawowych mechanizmach okien, takich jak obsługa zdarzeń.

- Swing w niewielkim stopniu zależy od platformy, dzięki czemu jest mniej podatny na błędy związane z danym systemem.
- Sposób działania i wygląd elementów Swinga jest taki sam na różnych platformach.

Niemniej trzecia z wymienionych zalet może być uważana za wadę — jeśli elementy interfejsu użytkownika wyglądają tak samo na wszystkich platformach, to znaczy, że wyglądają **inaczej** niż elementy natywne, co z kolei oznacza, że będą słabiej znane użytkownikom.

W pakiecie Swing problem ten został rozwiązany w bardzo elegancki sposób. Programista piszący program przy użyciu klas Swing może nadać mu specyficzny charakter. Rysunki 7.1 i 7.2 przedstawiają ten sam program w stylu systemu Windows i GTK.

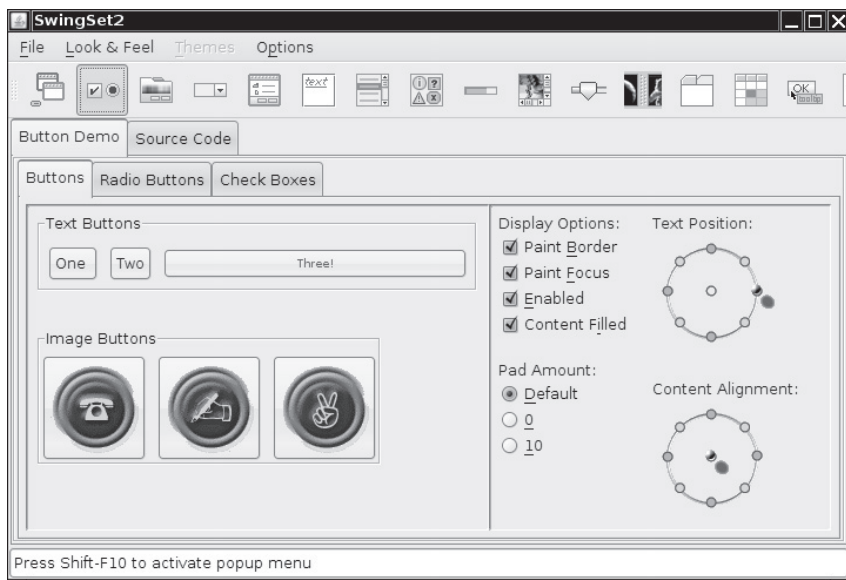
Rysunek 7.1.
Styl systemu
Windows



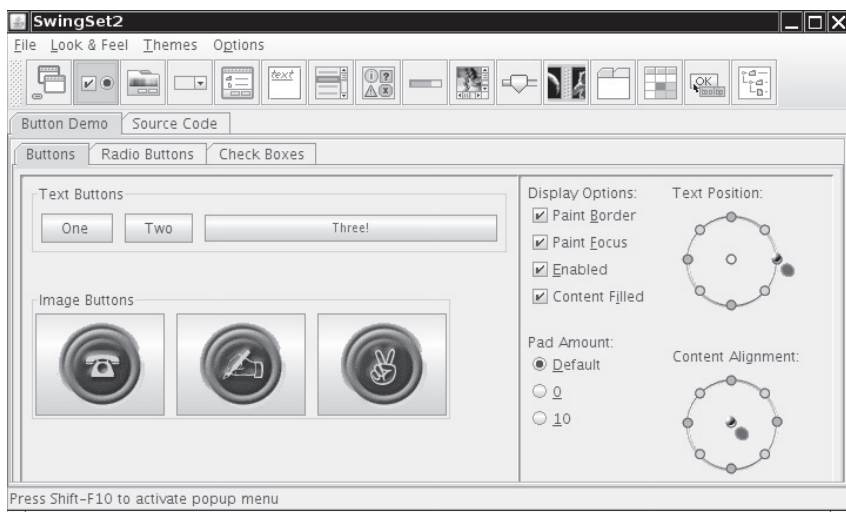
Dodatkowo firma Sun opracowała niezależny od platformy styl o nazwie **Metal**, który później przez specjalistów od marketingu przechrzczono na **Java look and feel**. Jednak większość programistów nadal używa określenia „Metal” i my również trzymamy się tej konwencji w tej książce.

Ze względu na krytyczne głosy kierowane pod adresem stylu Metal, który według niektórych wydawał się zbyt ciężki, w Java SE 5.0 nieco go odświeżono (zobacz rysunek 7.3). Obecnie styl Metal obsługuje wiele motywów — różnych wersji kolorystycznych i zestawów czcionek. Motyw domyślny nazywa się Ocean.

Rysunek 7.2.
Styl GTK



Rysunek 7.3.
Motyw Ocean
stylu Metal

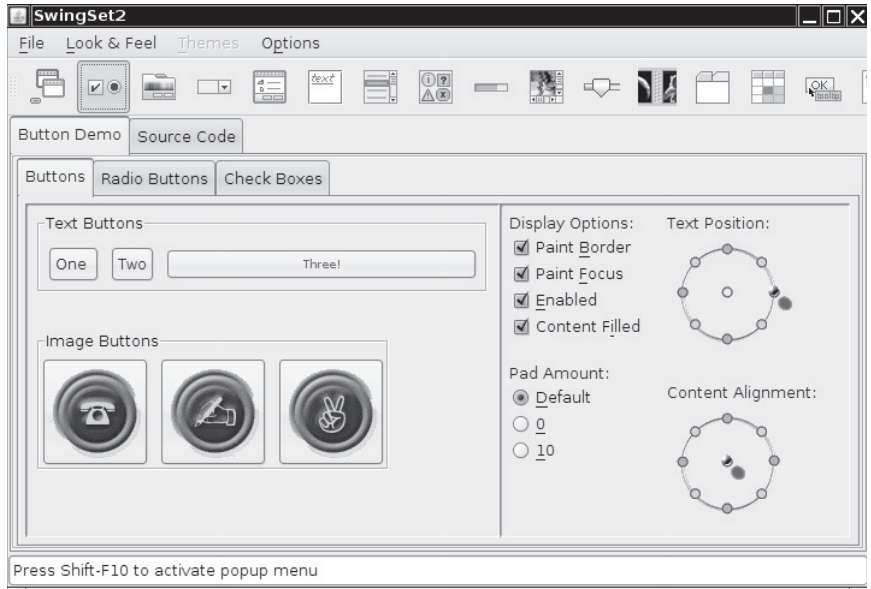


W Java SE 6 poprawiono obsługę natywnego stylu systemu Windows i GTK. Aktualnie aplikacje Swing obsługują schematy kolorów i pulsujące przyciski oraz paski przewijania, które stały się ostatnio modne.

W Java 7 dodano nowy styl o nazwie Nimbus (rysunek 7.4), ale nie jest on domyślnie dostępny. W stylu tym wykorzystywana jest grafika wektorowa zamiast bitmapowej, dzięki czemu interfejsy tworzone przy jego użyciu są niezależne od rozmiaru ekranu.

Niektórzy użytkownicy wolą, aby aplikacje w Javie wyglądały tak jak inne programy na danej platformie, inni wolą styl Metal, a jeszcze inni preferują styl całkiem innego producenta. Jak przekonamy się w rozdziale 8., umożliwienie użytkownikom wyboru dowolnego stylu jest bardzo łatwe.

Rysunek 7.4.
Styl Nimbus



W Javie możliwe jest rozszerzenie istniejącego stylu, a nawet utworzenie całkiem nowego. W tej książce nie mamy wystarczająco dużo miejsca, aby opisać ten żmudny proces polegający na określeniu sposobu rysowania każdego komponentu. Jednak niektórzy programiści pokusili się o to, zwłaszcza ci, którzy przenosili programy w Javie na nietypowe platformy, jak terminale sklepowe czy urządzenia kieszonkowe. Zbiór ciekawych stylów można znaleźć pod adresem <http://www.javootoo.com/>.

W Java SE 5.0 wprowadzono styl o nazwie Synth, który upraszcza cały ten proces. W Synth styl można definiować poprzez dostarczenie obrazów i deskryptorów XML. Nie trzeba nic programować.



Styl Napkin (<http://napkinlaf.sourceforge.net>) nadaje elementom interfejsu użytkownika wygląd przypominający odręczne rysowanie. Wysyłając do klienta utworzony w nim prototyp, dajemy wyraźny sygnał, że nie jest to jeszcze ukończony produkt.



Programowanie interfejsu użytkownika w Javie opiera się obecnie w większości na pakiecie Swing. Jest tylko jeden godny uwagi wyjątek. Środowisko zintegrowane Eclipse używa zestawu narzędzi graficznych o nazwie SWT, który podobnie jak AWT odwzorowuje natywne komponenty na różnych platformach. Artykuły na temat SWT można znaleźć na stronie <http://www.eclipse.org/articles/>.

Firma Oracle pracuje nad alternatywną technologią o nazwie JavaFX, która może w przyszłości zastąpić Swing. Jeśli chcesz dowiedzieć się o niej więcej, zajrzyj na stronę <http://www.oracle.com/technetwork/java/javafx/overview>.

Każdy, kto pisał programy dla systemu Microsoft Windows w językach Visual Basic lub C#, wie, jak dużym ułatwieniem są graficzne narzędzia do projektowania układu i edytory zasobów. Narzędzia te umożliwiają zaprojektowanie całej wizualnej strony aplikacji oraz automatycznie generują większość (często całość) kodu GUI. Dla Javy również dostępne są narzędzia

wspomagające budowanie GUI, ale naszym zdaniem, aby się nimi sprawnie posługiwać, trzeba najpierw nauczyć się robić to samodzielnie. Pozostała część tego rozdziału została poświęcona opisowi technik wyświetlania okien i rysowania w nich różnych elementów.

7.2. Tworzenie ramki

Okno najwyższego poziomu, tzn. takie, które nie jest zawarte w żadnym innym oknie, nazywa się w Javie **ramką** (ang. *frame*). W bibliotece AWT dla tego typu okien utworzono klasę o nazwie `Frame`. Wersja Swing tej klasy nosi nazwę `JFrame` i ją rozszerza. Ramka `JFrame` jest jednym z niewielu komponentów Swinga, które nie są rysowane w obszarze roboczym. W związku z tym elementy dodatkowe (przyciski, pasek tytułu, ikony itd.) są rysowane przez system operacyjny, a nie klasy Swing.



Nazwy większości klas komponentów Swing zaczynają się od litery J, np. `JButton`, `JFrame` itd. Istnieją także klasy `Button` i `Frame`, ale są to komponenty AWT. Jeśli litera J zostanie przypadkowo pominięta, program może przejść kompilację bez problemu, ale mieszanina komponentów AWT i Swing może spowodować nietypowe zachowania lub wygląd aplikacji.

W tym podrozdziale przejrzymy najpopularniejsze metody pracy z komponentem Swing o nazwie `JFrame`. Listing 7.1 przedstawia prosty program wyświetlający na ekranie pustą ramkę widoczną na rysunku 7.5.

Listing 7.1. `simpleFrame/SimpleFrameTest.java`

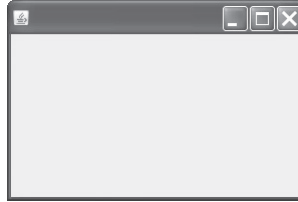
```
package simpleFrame;

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

/**
 * @version 1.32 2007-06-12
 * @author Cay Horstmann
 */
public class SimpleFrameTest
{
    public static void main(String[] args)
    {
        EventQueue.invokeLater(new Runnable()
        {
            public void run()
            {
                SimpleFrame frame = new SimpleFrame();
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.setVisible(true);
            }
        });
    }
}
```

Rysunek 7.5.

Najprostsza
widoczna ramka



```
class SimpleFrame extends JFrame
{
    private static final int DEFAULT_WIDTH = 300;
    private static final int DEFAULT_HEIGHT = 200;

    public SimpleFrame()
    {
        setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
    }
}
```

Przeanalizujemy powyższy program wiersz po wierszu.

Klasy Swing znajdują się w pakiecie `javax.swing`. Nazwa pakietu `javax` oznacza, że jest to pakiet rozszerzający Javy, a nie podstawowy. Swing jest uznawany za rozszerzenie ze względów historycznych. Jest jednak dostępny w każdej wersji Java SE od 1.2.

Domyślny rozmiar ramki 0×0 jest raczej mało atrakcyjny. Zdefiniowaliśmy podklasę o nazwie `SimpleFrame`, której konstruktor ustawia rozmiar na 300×200 pikseli. Jest to jedyna różnica pomiędzy klasami `SimpleFrame` i `JFrame`.

W metodzie `main` klasy `SimpleFrameTest` tworzony jest obiekt klasy `SimpleFrame`, który następnie został uwidoczniony.

W każdym programie opartym na Swingu trzeba poradzić sobie z dwoma zagadnieniami technicznymi.

Po pierwsze, konfiguracja każdego komponentu Swing musi się odbywać w **wątku dystrybucji zdarzeń** (ang. *event dispatch thread*), który steruje wysyłaniem zdarzeń, jak kliknięcia przyciskiem myszy lub wciśnięcie klawisza na klawiaturze, do elementów interfejsu użytkownika. Poniższy fragment programu wykonuje instrukcje w wątku dystrybucji zdarzeń:

```
EventQueue.invokeLater(new Runnable()
{
    public void run()
    {
        instrukcje
    }
});
```

Szczegółowy opis tego zagadnienia znajduje się w rozdziale 14. Na razie potraktujmy to jako magiczny fragment kodu potrzebny do uruchomienia programu Swing.



Wiele programów nie inicjuje interfejsu użytkownika w wątku dystrybucji zdarzeń. Nie ma żadnych przeszkód, aby operację tę przeprowadzać w wątku głównym. Niestety ze względu na to, że komponenty Swing stawały się coraz bardziej złożone, programiści w firmie Sun nie mogli zagwarantować bezpieczeństwa tej metody. Prawdopodobieństwo wystąpienia błędu jest niezwykle niskie, ale nikt nie chciałby być tym pechowcem, któremu się to przydarzy. Lepiej pisać właściwy kod, nawet jeśli wygląda nieco tajemniczo.

Po drugie, określamy, co ma się stać, kiedy użytkownik zamknie ramkę aplikacji. W tym przypadku chcemy, aby program został zamknięty. Odpowiedzialna jest za to poniższa instrukcja:

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

W programach składających się z wielu ramek program nie powinien kończyć działania w wyniku zamknięcia jednej z nich. Przy standardowych ustawieniach, jeśli użytkownik zamknie ramkę, zostanie ona ukryta, a program nie zakończy działania (dobrze by było, gdyby program był wyłączany w chwili zamknięcia jego **ostatniej** ramki, ale Swing działa inaczej).

Samo utworzenie ramki nie oznacza, że zostanie ona wyświetlona. Ramki na początku swojego istnienia są niewidoczne. Dzięki temu programista może dodać do nich wszystkie komponenty, zanim ukażą się po raz pierwszy. Aby wyświetlić ramkę, metoda `main` wywołuje na jej rzecz metodę `setVisible`.



Przed Java SE 5.0 można było używać metody `show` dziedziczonej przez klasę `JFrame` po nadklasie `Window`. Nadklasą klasy `Window` jest `Component`, która także zawiera metodę `show`. Stosowanie metody `Component.show` zaczęto odradzać w Java SE 1.2. W zamian, aby wyświetlić komponent, należy użyć metody `setVisible(true)`. Natomiast metoda `Window.show` **nie była** odradzana aż do Java SE 1.4. Możliwość uwidocznienia okna i przeniesienia go na przód była nawet przydatna. Niestety metoda `show` dla okien również jest odradzana od Java SE 5.0.

Po rozplanowaniu instrukcji inicjujących metoda `main` kończy działanie. Zauważmy, że zakończenie metody `main` nie oznacza zamknięcia programu, a jedynie głównego wątku. Wątek dystrybucji zdarzeń podtrzymuje działanie programu aż do jego zakończenia poprzez zamknięcie ramki lub wywołanie metody `System.exit`.

Uruchomiony program przedstawia rysunek 7.5 — jest to zwykłe szare okno najwyższego poziomu. Jak widać, pasek tytułu i pozostałe dodatki, jak zaokrąglone rogi służące do zmiany rozmiaru okna, zostały narysowane przez system operacyjny, a nie klasy Swing. Elementy te będą wyglądały inaczej, jeśli uruchomimy ten program w systemach Windows, GTK czy Mac OS X. Biblioteka Swing rysuje wszystko, co znajduje się wewnątrz ramki. W tym przypadku jej zadanie sprowadza się jedynie do wypełnienia domyślnym kolorem tła.



Od Java SE 1.4 istnieje możliwość wyłączenia wszelkich dodatków za pomocą wywołania metody `frame.setUndecorated(true)`.

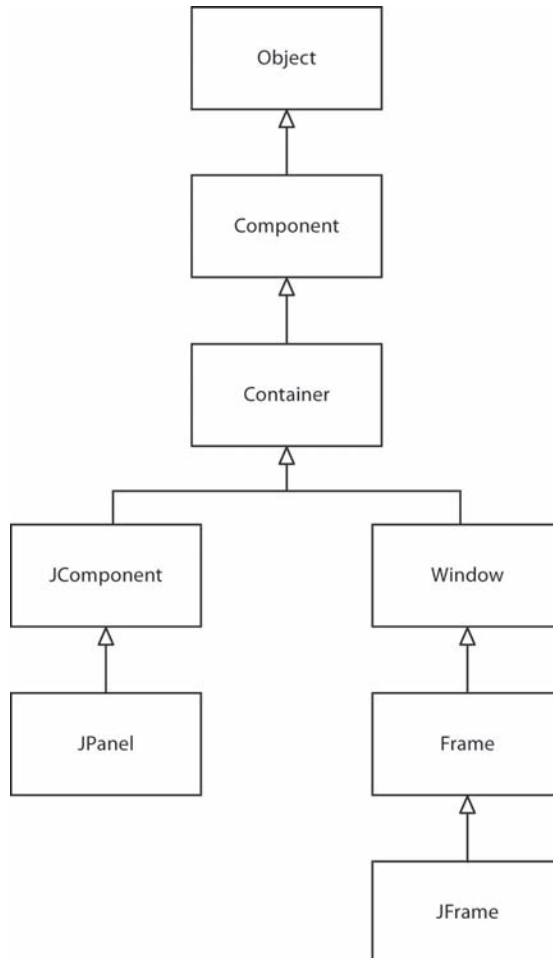
7.3. Pozycjonowanie ramki

Klasa `JFrame` udostępnia tylko kilka metod zmieniających wygląd ramek. Oczywiście większość metod służących do zmiany wymiarów i położenia ramki jest w klasie `JFrame` dziedziczona po różnych nadklasach. Poniżej znajduje się lista najważniejszych z tych metod:

- `setLocation` i `setBounds` — ustawiają położenie ramki.
- `setIconImage` — określa ikonę wyświetlaną w pasku tytułu, w zasobniku systemowym itd.
- `setTitle` — ustawia tekst w pasku tytułu.
- `setResizable` — pobiera wartość logiczną określającą, czy użytkownik może zmieniać rozmiar ramki.

Rysunek 7.6 przedstawia hierarchię dziedziczenia klasy `JFrame`.

Rysunek 7.6.
Hierarchia dziedziczenia klas ramek i komponentów w pakietach AWT i Swing





W wyciągach z API do tego podrozdziału przedstawiamy te metody, które naszym zdaniem mają największe znaczenie przy nadawaniu ramkom odpowiedniego stylu. Niektóre z nich są udostępnione w klasie `JFrame`. Inne z kolei pochodzą od różnych nadklas klasy `JFrame`. Czasami konieczne może być przeszukanie dokumentacji API w celu znalezienia metod przeznaczonych do określonego celu. Niestety jest to dość żmudna praca, jeśli chodzi o metody dziedziczone. Na przykład metoda `ToFront` ma zastosowanie do obiektów typu `JFrame`, ale ponieważ jest dziedziczona po klasie `Window`, w dokumentacji `JFrame` nie ma jej opisu. Jeśli uważasz, że powinna istnieć metoda wykonująca określone działanie, ale nie ma jej w dokumentacji danej klasy, szukaj dokumentację metod nadklas tej klasy. Na górze każdej strony w dokumentacji API znajdują się odnośniki do nadklas, a lista metod dziedziczonych znajduje się pod zestawieniem nowych i przesłoniętych metod.

Według dokumentacji API metody służące do zmieniania rozmiaru i kształtu ramek znajdują się w klasach `Component` (będącej przodkiem wszystkich obiektów GUI) i `Window` (będącej nadklasą klasy `Frame`). Na przykład do zmiany położenia komponentu można użyć metody `setLocation` z klasy `Component`. Wywołanie:

```
setLocation(x, y)
```

umieści lewy górny róg komponentu w odległości x pikseli od lewej krawędzi ekranu i y pikseli od góry. Wartości $(0, 0)$ oznaczają lewy górny róg ekranu. Podobnie metoda `setBounds` w klasie `Component` umożliwi zmianę rozmiaru i położenia komponentu (zwłaszcza ramki `JFrame`) w jednym wywołaniu:

```
setBounds(x, y, width, height)
```

Istnieje też możliwość pozostawienia decyzji o położeniu okna systemowi. Wywołanie:

```
setLocationByPlatform(true);
```

przed wyświetleniem okna spowoduje, że system we własnym zakresie określi jego położenie (ale nie rozmiar). Zazwyczaj nowe okno jest wyświetlane z nieznacznym przesunięciem względem poprzedniego.



W przypadku ramki współrzędne metod `setLocation` i `setBounds` są względne do całego ekranu. W rozdziale 9. dowiemy się, że współrzędne komponentów znajdujących się w kontenerze odnoszą się do tego kontenera.

7.3.1. Własności ramek

Wiele metod klas komponentów występuje w parach get-set, jak poniższe metody klasy `Frame`:

```
public String getTitle()
public void setTitle(String title)
```

Taka para metod typu get-set nazywa się **własnością** (ang. *property*). Własność ma nazwę i typ. Nazwa jest taka sama jak słowo uzyskane w wyniku opuszczenia członu get i zamienienia pierwszej litery powstałego słowa na małą. Na przykład klasa `Frame` ma własność o nazwie `title` i typie `String`.

Z założenia `title` jest własnością ramki. Ustawienie (`set`) niniejszej własności powoduje zmianę tytułu na ekranie użytkownika. Pobranie jej (`get`) zwraca wartość, która została wcześniej ustawiona.

Nie wiemy (i nie obchodzi nas to), jak klasa `Frame` implementuje tę własność. Prawdopodobnie wykorzystuje swój odpowiednik ramki do przechowywania tytułu. Możliwe, że ma następujące pole:

```
private String title; // nie jest wymagane dla własności
```

Jeśli klasa zawiera pasujące pole, nie wiemy (lub nie obchodzi nas to), jak metody dostępowe i ustawiające są zaimplementowane. Prawdopodobnie po prostu odczytują i ustawiają dane pole. Możliwe, że robią jeszcze coś, np. powiadamiają system o każdej zmianie tytułu.

Jest tylko jeden wyjątek od konwencji `get-set`: metody własności typu logicznego zaczynają się od przedrostka `is`. Na przykład dwie przedstawione poniżej metody definiują własność `locationByPlatform`:

```
public boolean isLocationByPlatform()
public void setLocationByPlatform(boolean b)
```

Znacznie więcej na temat własności piszemy w rozdziale 8. drugiego tomu.



Wiele języków programowania, zwłaszcza Visual Basic i C#, standardowo obsługuje własności. Niewykluczone, że w przyszłości w Javie również pojawi się podobna konstrukcja językowa.

7.3.2. Określanie rozmiaru ramki

Pamiętajmy, że jeśli nie ustawimy własnego rozmiaru ramek, wszystkie będą miały wymiary 0×0 pikseli. Aby nie komplikować naszych przykładowych programów, ustawiamy ramki na rozmiar do przyjęcia na większości ekranów. Jednak w profesjonalnej aplikacji należy sprawdzić rozdzielczość ekranu użytkownika i napisać podprogram zmieniający rozmiar ramek w zależności od potrzeby. Na przykład okno, które wygląda znakomicie na ekranie laptopa, na ekranie o wysokiej rozdzielczości skurczy się do rozmiarów znaczka pocztowego.

Aby sprawdzić rozmiar ekranu, należy wykonać następujące działania: wywołaj statyczną metodę `getDefaultToolkit` klasy `Toolkit` w celu utworzenia obiektu typu `Toolkit` (klasa `Toolkit` jest zbiornikiem rozmaitych metod, które współpracują z systemem). Następnie wywołaj metodę `getScreenSize`, która zwraca rozmiar ekranu w postaci obiektu `Dimension`. Obiekt tego typu przechowuje wysokość i szerokość w zmiennych publicznych (!) o nazwach `width` i `height`. Poniżej przedstawiamy opisywany fragment programu:

```
Toolkit kit = Toolkit.getDefaultToolkit();
Dimension screenSize = kit.getScreenSize();
int screenWidth = screenSize.width;
int screenHeight = screenSize.height;
```

Rozmiar ramki ustawiamy na połowę ekranu i pozwalamy systemowi na ustalenie położenia ramki:

```
setSize(screenWidth / 2, screenHeight / 2);
setLocationByPlatform(true);
```

Dodatkowo dostarczymy ikonę. Do wygodnego ładowania obrazów można wykorzystać klasę `ImageIcon`. Poniżej przedstawiony jest sposób jej użycia:

```
Image img = new ImageIcon("icon.gif").getImage();
setIconImage(img);
```

Ikona może się pojawić w różnych miejscach, w zależności od systemu operacyjnego. W systemie Windows pojawi się na przykład w lewym górnym rogu okna oraz będzie widoczna wśród aktywnych zadań pojawiających się w wyniku naciśnięcia kombinacji klawiszy *Alt+Tab*.

Listing 7.2 przedstawia pełny kod omawianego programu. Po jego uruchomieniu zwróć uwagę na ikonę *Core Java*.

Listing 7.2. `sizedFrame/SizedFrameTest.java`

```
package sizedFrame;

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

/**
 * @version 1.32 2007-04-14
 * @author Cay Horstmann
 */
public class SizedFrameTest
{
    public static void main(String[] args)
    {
        EventQueue.invokeLater(new Runnable()
        {
            public void run()
            {
                JFrame frame = new SizedFrame();
                frame.setTitle("SizedFrame");
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.setVisible(true);
            }
        });
    }
}

class SizedFrame extends JFrame
{
    public SizedFrame()
    {
        // Sprawdzenie wymiarów ekranu.

        Toolkit kit = Toolkit.getDefaultToolkit();
        Dimension screenSize = kit.getScreenSize();
        int screenHeight = screenSize.height;
        int screenWidth = screenSize.width;

        // Ustawienie szerokości i wysokości ramki oraz polecenie systemowi, aby ustalił jej położenie.
```



```

setSize(screenWidth / 2, screenHeight / 2);
setLocationByPlatform(true);

// Ustawienie ikony i tytułu.

Image img = new ImageIcon("icon.gif").getImage();
setIconImage(img);
}
}

```

Jeszcze kilka dodatkowych wskazówek dotyczących obsługi ramek:

- Jeśli ramka zawiera tylko standardowe komponenty, jak przyciski i pola tekstowe, jej rozmiar można ustawić za pomocą metody `pack`. Rozmiar zostanie ustawiony na najmniejszy, w którym zmieszczą się wszystkie komponenty. Często rozmiar głównej ramki jest ustawiany na największą wartość. Od Java SE 1.4 ramkę można zmaksymalizować za pomocą następującego wywołania:


```
frame.setExtendedState(Frame.MAXIMIZED_BOTH);
```
- Dobrym pomysłem jest zapisanie położenia i rozmiaru ramki ustawionych przez użytkownika i zastosowanie tych ustawień przy ponownym uruchomieniu aplikacji. Jak to zrobić, dowiemy się w rozdziale 10., przy okazji omawiania API preferencji.
- Jeśli aplikacja wykorzystuje kilka ekranów, ich rozmiary można sprawdzić za pomocą metod `GraphicsEnvironment` i `GraphicsDevice`.
- Ponadto klasa `GraphicsDevice` umożliwia uruchomienie programu w trybie pełnoekranowym.

java.awt.Component **1.0**

- boolean `isVisible()`

- void `setVisible(boolean b)`

Sprawdza lub ustawia własność widoczności. Komponenty są początkowo widoczne z wyjątkiem komponentów najwyższego poziomu, jak `JFrame`.

- void `setSize(int width, int height)` **1.1**

Ustawia szerokość i wysokość komponentu.

- void `setLocation(int x, int y)` **1.1**

Przenosi komponent w inne miejsce. Współrzędne `x` i `y` są względne do kontenera lub ekranu, jeśli komponent jest komponentem najwyższego poziomu (np. `JFrame`).

- void `setBounds(int x, int y, int width, int height)` **1.1**

Przesuwa komponent i zmienia jego rozmiar.

- Dimension `getSize()` **1.1**

- void `setSize(Dimension d)` **1.1**

Pobiera lub ustawia własność `size` komponentu.

java.awt.Window 1.0

- void toFront()

Przenosi okno przed wszystkie pozostałe okna.

- void toBack()

Przenosi okno na sam dół stosu okien i odpowiednio przestawia pozostałe widoczne okna.

- boolean isLocationByPlatform() **5.0**

- void setLocationByPlatform(boolean b) **5.0**

Pobiera lub ustawia własność `locationByPlatform`. Jeśli zostanie ona ustawiona przed wyświetleniem okna, platforma wybierze odpowiednią lokalizację.

java.awt.Frame 1.0

- boolean isResizable()

- void setResizable(boolean b)

Pobiera lub ustawia własność `resizable`. Jeśli jest ona ustawiona, użytkownik może zmieniać rozmiar ramki.

- String getTitle()

- void setTitle(String s)

Pobiera lub ustawia własność `title` określającą tekst na pasku tytułu.

- Image getIconImage()

- void setIconImage(Image image)

Pobiera lub ustawia własność `iconImage`, która określa ikonę ramki. System może wyświetlić ikonę jako dodatek w ramce lub w innym miejscu.

- boolean isUndecorated() **1.4**

- void setUndecorated(boolean b) **1.4**

Pobiera lub ustawia własność `undecorated`. Jeśli ta własność jest ustawiona, ramka nie zawiera żadnych dodatków, jak pasek tytułu czy przycisk zamykający. Ta metoda musi być wywołana przed wyświetleniem ramki.

- int getExtendedState() **1.4**

- void setExtendedState(int state) **1.4**

Pobiera lub ustawia stan rozszerzonego okna. Możliwe stany to:

```
Frame.NORMAL  
Frame.ICONIFIED  
Frame.MAXIMIZED_HORIZ  
Frame.MAXIMIZED_VERT  
Frame.MAXIMIZED_BOTH
```

java.awt.Toolkit 1.0

- static Toolkit getDefaultToolkit()
Zwraca standardowy zestaw narzędzi.
- Dimension getScreenSize()
Pobiera rozmiar ekranu.

javax.swing.ImageIcon 1.2

- ImageIcon(String filename)
Tworzy ikonę, której obraz jest przechowywany w pliku.
- Image getImage()
Pobiera obraz ikony.

7.4. Wyświetlanie informacji w komponencie

W tym podrozdziale nauczymy się wyświetlać informacje w ramce. Zamiast przykładowego programu wyświetlającego tekst w konsoli, który widzieliśmy w rozdziale 3., zaprezentujemy program wyświetlający tekst w ramce, jak na rysunku 7.7.

Rysunek 7.7.
Ramka
wyświetlająca
tekst



Łańcuch tekstowy można wydrukować bezpośrednio na ramce, ale jest to uznawane za zły zwyczaj programistyczny. Ramki w Javie są z założenia kontenerami do przechowywania komponentów, takich jak pasek menu i inne elementy interfejsu użytkownika. Z reguły rysowanie odbywa się na innym dodanym do ramki komponencie.

Struktura ramki `JFrame` jest zadziwiająco skomplikowana. Rysunek 7.8 przedstawia schemat budowy takiej ramki. Jak widać, ramka `JFrame` składa się z czterech warstw. Trzy z nich — podstawowa (ang. *root pane*), warstwowa (ang. *layered pane*) i przezroczysta (ang. *glass pane*) — nie są dla nas interesujące. Ich przeznaczeniem jest organizacja paska menu i warstwy z treścią oraz implementacja stylu. Część interesująca programistów Swing to **warstwa treści** (ang. *content pane*). Podczas projektowania ramki komponenty dodaje się do warstwy treści, stosując kod podobny do poniższego:

```
Container contentPane = frame.getContentPane();
Component c = . . . ;
contentPane.add(c);
```

Do Java SE 1.4 metoda `add` klasy `JFrame` powodowała wyjątek wyświetlający komunikat „Do not use `JFrame.add()`. Use `JFrame.getContentPane().add()` instead”. W Java SE 5.0 zrezygnowano z takiej edukacji programistów, czego wyrazem jest zezwolenie na wywołanie metody `JFrame.add` na rzecz warstwy z treścią.

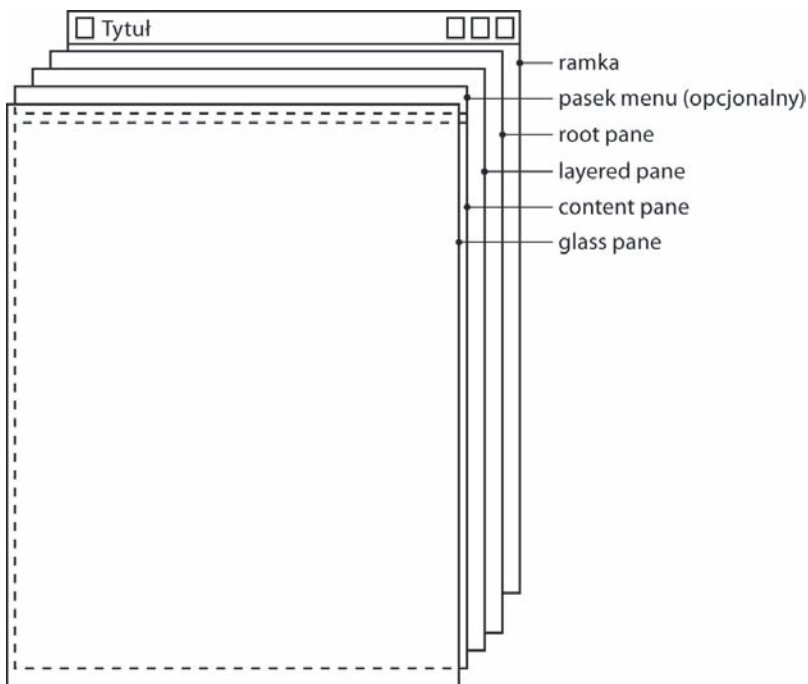
Dzięki temu od Java SE 5.0 można stosować krótki zapis:

```
frame.add(c);
```

Tym razem chcemy dodać do ramki jeden komponent, na którym narysujemy nasz komunikat. Aby móc rysować na komponencie, należy zdefiniować klasę rozszerzającą klasę `JComponent` i przesłonić w niej metodę `paintComponent` klasy nadrzędnej.

Metoda `paintComponent` przyjmuje jeden parametr typu `Graphics`. Obiekt typu `Graphics` zawiera ustawienia dotyczące rysowania obrazów i tekstu, jak czcionka czy aktualny kolor. Rysowanie w Javie zawsze odbywa się za pośrednictwem obiektu `Graphics`. Udostępnia on metody rysujące wzory, obrazy i tekst.

Rysunek 7.8.
Wewnętrzna
struktura
ramki `JFrame`



Parametr `Graphics` jest podobny do kontekstu urządzeń (ang. *device context*) w systemie Windows i kontekstu graficznego (ang. *graphics context*) w programowaniu dla systemu X11.

Poniższy fragment programu tworzy komponent, na którym można rysować:

```
class MyComponent extends JComponent
{
    public void paintComponent(Graphics g)
```

```

    {
        kod rysujący
    }
}

```

Za każdym razem, kiedy okno musi być ponownie narysowane, metoda obsługi zdarzeń informuje o tym komponent. Powoduje to uruchomienie metod `paintComponent` wszystkich komponentów.

Nigdy nie należy wywoływać metody `paintComponent` samodzielnie. Jest ona wywoływana automatycznie, gdy trzeba ponownie narysować jakąś część aplikacji, i nie należy zaburzać tego automatycznego procesu.

Jakiego rodzaju czynności uruchamiają tę automatyczną reakcję? Rysowanie jest konieczne, na przykład kiedy użytkownik zwiększy rozmiar okna albo je zminimalizuje, a następnie zmaksymalizuje. Jeśli zostanie otwarte nowe okno, które częściowo przykryje stare, to po zamknięciu tego nowego okna przykryta część starego zostanie zniszczona i trzeba ją będzie ponownie narysować (system graficzny nie zapisuje pikseli, które znajdują się pod spodem). Oczywiście przy pierwszym uruchomieniu okna musi ono przetworzyć kod określający sposób i miejsce rysowania początkowych elementów.



Aby wymusić ponowne rysowanie ekranu, należy użyć metody `repaint`. Wywoła ona metody `paintComponent` wszystkich komponentów, które mają prawidłowo skonfigurowany obiekt `Graphics`.

Z przedstawionego powyżej fragmentu kodu wnioskujemy, że metoda `paintComponent` przyjmuje tylko jeden parametr typu `Graphics`. Jednostką miary obiektów `Graphics` wyświetlanych na ekranie są piksele. Para współrzędnych (0, 0) określa lewy górny róg komponentu, na którego powierzchni odbywa się rysowanie.

Wyświetlanie tekstu jest specjalnym rodzajem rysowania. Klasa `Graphics` udostępnia metodę `drawString` o następującej składni:

```
g.drawString(text, x, y)
```

Chcemy narysować łańcuch: *To nie jest program „Witaj, świecie”* w oryginalnym oknie w odległości około jednej czwartej szerokości od lewej krawędzi i połowy wysokości od krawędzi górnej. Mimo że nie potrafimy jeszcze mierzyć długości łańcuchów, zaczniemy rysowanie w punkcie o współrzędnych (75, 100). Oznacza to, że pierwsza litera łańcucha jest przesunięta w prawo o 75 pikseli i w dół o 100 pikseli (w rzeczywistości o 100 pikseli w dół przesunięta jest podstawowa linia pisma — więcej na ten temat w dalszej części rozdziału). Kod opisanej metody `paintComponent` znajduje się poniżej:

```

class NotHelloWorldComponent extends JComponent
{
    public static final int MESSAGE_X = 75;
    public static final int MESSAGE_Y = 100;

    public void paintComponent(Graphics g)
    {
        g.drawString("To nie jest program „Witaj, świecie”", MESSAGE_X, MESSAGE_Y);
    }
}

```

```

    }
    . . .
}

```

W końcu komponent powinien informować użytkownika o tym, jaki ma być jego rozmiar. Przesłonimy metodę `getPreferredSize`, aby zwracała obiekt klasy `Dimension` z preferowaną szerokością i wysokością:

```

class NotHelloWorldComponent extends JComponent
{
    private static final int DEFAULT_WIDTH = 300;
    private static final int DEFAULT_HEIGHT = 200;
    . . .
    public Dimension getPreferredSize() { return new Dimension(DEFAULT_WIDTH,
DEFAULT_HEIGHT); }
}

```

Gdy umieścimy w ramce jakieś komponenty i będziemy chcieli użyć ich preferowanych rozmiarów, to zamiast `setSize` wywołamy metodę `pack`:

```

class NotHelloWorldFrame extends JFrame
{
    public NotHelloWorldFrame()
    {
        add(new NotHelloWorldComponent());
        pack();
    }
}

```

Listing 7.3 zawiera pełny kod programu.



Niektórzy programiści wolą rozszerzać klasę `JPanel` zamiast `JComponent`. Obiekt `JPanel` jest z założenia **kontenerem** na inne komponenty, ale można także na nim rysować. Jest jednak między nimi jedna różnica — panel jest **nieprzezroczysty**, czyli rysuje wszystkie piksele w swoim obrębie. Najprostszym sposobem na zrobienie tego jest narysowanie panelu z kolorowym tłem za pomocą wywołania metody `super.paintComponent` w metodzie `paintComponent` każdej podklasy panelu:

```

class NotHelloWorldPanel extends JPanel
{
    public void paintComponent(Graphics g)
    {
        super.paintComponent(g);
        procedury rysujące
    }
}

```

Listing 7.3. notHelloWorld/NotHelloWorld.java

```

package notHelloWorld;

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

/**
 * @version 1.32 2007-06-12

```

```

    *@author Cay Horstmann
    */
public class NotHelloWorld
{
    public static void main(String[] args)
    {
        EventQueue.invokeLater(new Runnable()
        {
            public void run()
            {
                JFrame frame = new NotHelloWorldFrame();
                frame.setTitle("NotHelloWorld");
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.setVisible(true);
            }
        });
    }
}

/**
 *Ramka zawierająca panel z komunikatem.
 */
class NotHelloWorldFrame extends JFrame
{
    public NotHelloWorldFrame()
    {
        add(new NotHelloWorldComponent());
        pack();
    }
}

/**
 *Panel wyświetlający komunikat.
 */
class NotHelloWorldPanel extends JComponent
{
    public static final int MESSAGE_X = 75;
    public static final int MESSAGE_Y = 100;

    private static final int DEFAULT_WIDTH = 300;
    private static final int DEFAULT_HEIGHT = 200;

    public void paintComponent(Graphics g)
    {
        g.drawString("To nie jest program „Witaj, świecie“.", MESSAGE_X, MESSAGE_Y);
    }

    public Dimension getPreferredSize() { return new Dimension(DEFAULT_WIDTH,
        DEFAULT_HEIGHT); }
}

```

javax.swing.JFrame **1.2**

■ Container getContentPane()

Zwraca obiekt ContentPane dla ramki JFrame.

- `Component add(Component c)`

Dodaje i zwraca dany komponent do warstwy treści ramki (przed Java SE 5.0 ta metoda powodowała wyjątek).

`java.awt.Component` **1.0**

- `void repaint()`

Powoduje ponowne jak najszybsze narysowanie komponentu.

- `Dimension getPreferredSize()`

Metoda, którą należy przesłonić, aby zwracała preferowany rozmiar komponentu.

`javax.swing.JComponent` **1.2**

- `void paintComponent(Graphics g)`

Metoda, którą należy przesłonić w celu zdefiniowania sposobu rysowania określonego komponentu.

`java.awt.Window` **1.0**

- `void pack()`

Zmienia rozmiar okna, biorąc pod uwagę preferowane rozmiary znajdujących się w nim komponentów.

7.5. Figury 2D

Od Java 1.0 klasa `Graphics` udostępnia metody rysujące linie, prostokąty, elipsy itd. Ich możliwości są jednak bardzo ograniczone. Nie ma na przykład możliwości ustawienia grubości linii ani obracania figur.

W Java 1.2 wprowadzono bibliotekę `Java2D` udostępniającą szeroki wachlarz metod graficznych. W tym rozdziale opisujemy tylko podstawy tej biblioteki — więcej bardziej zaawansowanych informacji na ten temat znajduje się w rozdziale 7. w drugim tomie.

Aby narysować figurę biblioteki `Java2D`, trzeba utworzyć obiekt klasy `Graphics2D`. Klasa ta jest podklasą klasy `Graphics`. Od Java SE 2 metody takie jak `paintComponent` automatycznie odbierają obiekty klasy `Graphics2D`. Wystarczy zastosować rzutowanie:

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


Figury geometryczne w bibliotece Java2D są obiektami. Istnieją klasy reprezentujące linie, prostokąty i elipsy:

```
Line2D
Rectangle2D
Ellipse2D
```

Wszystkie te klasy implementują interfejs Shape.



Biblioteka Java2D obsługuje także bardziej skomplikowane figury, jak łuki, krzywe drugiego i trzeciego stopnia oraz trajektorie. Więcej informacji na ten temat znajduje się w rozdziale 7. drugiego tomu.

Aby narysować figurę, należy najpierw utworzyć obiekt klasy implementującej interfejs Shape, a następnie wywołać metodę draw klasy Graphics2D. Na przykład:

```
Rectangle2D rect = . . . ;
g2.draw(rect);
```



Przed pojawieniem się biblioteki Java2D do rysowania figur używano metod klasy Graphics, np. drawRectangle. Na pierwszy rzut oka te stare wywołania wydają się prostsze, ale używając biblioteki Java2D, pozostawiamy sobie różne możliwości do wyboru — można później ulepszyć rysunki za pomocą rozmaitych narzędzi dostępnych w bibliotece Java2D.

Biblioteka Java2D nieco komplikuje programowanie. W przeciwieństwie do metod rysujących z wersji 1.0, w których współrzędne były liczbami całkowitymi, figury Java2D używają współrzędnych zmiennoprzecinkowych. W wielu przypadkach stanowi to duże ułatwienie dla programisty, ponieważ może on określać figury przy użyciu lepiej znanych mu jednostek (jak milimetry lub cale), które później są konwertowane na piksele. Biblioteka Java2D wykonuje obliczenia o pojedynczej precyzji na liczbach typu float w większości wykonywanych wewnętrznie działań. Pojedyncza precyzja w zupełności wystarcza — celem obliczeń geometrycznych jest przecież ustawienie pikseli na ekranie lub w drukarce. Dopóki błędy zaokrąglania mieszczą się w zakresie jednego piksela, rezultat wizualny nie cierpi. Ponadto obliczenia na liczbach typu float są na niektórych platformach szybsze, a dodatkowo wartości tego typu zajmują o połowę mniej miejsca niż wartości typu double.

Czasami jednak obliczenia na liczbach typu float bywają niewygodne, ponieważ Java niewzruszenie wymaga rzutowania, jeśli niezbędna jest konwersja wartości typu double na typ float. Przeanalizujmy na przykład poniższą instrukcję:

```
float f = 1.2; // błąd
```

Ta instrukcja spowoduje błąd kompilacji, ponieważ stała 1.2 jest typu double i kompilator obawia się utraty danych. Rozwiązaniem jest dodanie przyrostka F do stałej zmiennoprzecinkowej:

```
float f = 1.2F; // OK
```

Teraz przyjrzyjmy się poniższej instrukcji:

```
Rectangle2D r = . . .
float f = r.getWidth();           // błąd
```

Instrukcja ta spowoduje błąd kompilacji z tego samego powodu co poprzednia. Metoda `getWidth` zwraca wartość typu `double`. W tym przypadku rozwiązaniem jest rzutowanie:

```
float f = (float) r.getWidth();   // OK
```

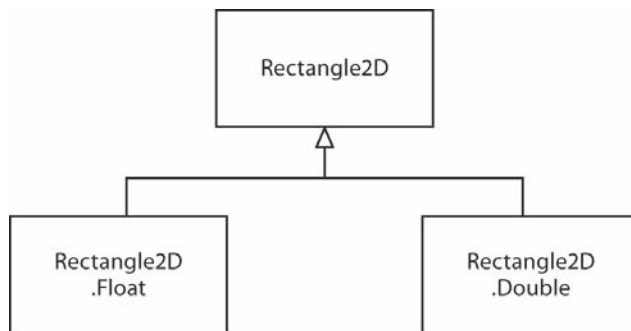
Ponieważ stosowanie przyrostków i rzutowania nie jest wygodne, projektanci biblioteki `Java2D` postanowili utworzyć **dwie wersje** każdej klasy reprezentującej figurę: jedną ze współrzędnymi typu `float` dla oszczędnych programistów i jedną ze współrzędnymi typu `double` dla leniwych (w tej książce zaliczamy się do tych drugich, czyli stosujemy współrzędne typu `double`, gdzie tylko możemy).

Projektanci biblioteki zastosowali ciekawą i początkowo wprowadzającą w błąd metodę pakowania obu wersji klas. Przyjrzyjmy się klasie `Rectangle2D`. Jest to abstrakcyjna klasa mająca dwie konkretne podklasy, które są dodatkowo statycznymi klasami wewnętrznymi:

```
Rectangle2D.Float
Rectangle2D.Double
```

Rysunek 7.9 przedstawia diagram dziedziczenia.

Rysunek 7.9.
Klasy
prostokątów 2D



Najlepiej ignorować fakt, że obie te konkretne klasy są statyczne i wewnętrzne — to tylko taki chwyt, który pozwala uniknąć nazw `FloatRectangle2D` i `DoubleRectangle2D` (więcej informacji na temat statycznych klas wewnętrznych znajduje się w rozdziale 6.).

Tworząc obiekt klasy `Rectangle2D.Float`, wartości określające współrzędne należy podawać jako typu `float`. W przypadku klasy `Rectangle2D.Double` współrzędne muszą być typu `double`.

```
Rectangle2D.Float floatRect = new Rectangle2D.Float(10.0F, 25.0F, 22.5F, 20.0F);
Rectangle2D.Double doubleRect = new Rectangle2D.Double(10.0, 25.0, 22.5, 20.0);
```

Ponieważ zarówno klasa `Rectangle2D.Float`, jak i `Rectangle2D.Double` rozszerzają wspólną klasę `Rectangle2D`, a metody w tych podklasach przesłaniają metody nadklasy, zapamiętywanie typu figury nie przynosi właściwie żadnych korzyści. Referencje do prostokątów można przechowywać w zmiennych typu `Rectangle2D`.

```
Rectangle2D floatRect = new Rectangle2D.Float(10.0F, 25.0F, 22.5F, 20.0F);
Rectangle2D doubleRect = new Rectangle2D.Double(10.0, 25.0, 22.5, 20.0);
```

Oznacza to, że użycie klas wewnętrznych jest konieczne tylko przy tworzeniu obiektów figur.

Parametry konstruktora określają lewy górny róg, szerokość i wysokość prostokąta.



Klasa `Rectangle2D.Float` zawiera jedną metodę, której nie dziedziczy po klasie `Rectangle2D`. Jest to metoda `setRect(float x, float y, float h, float w)`. Metody tej nie można użyć, jeśli referencja do obiektu typu `Rectangle2D.Float` jest przechowywana w zmiennej typu `Rectangle2D`. Nie jest to jednak duża strata — klasa `Rectangle2D` zawiera metodę `setRect` z parametrami typu `double`.

Metody klasy `Rectangle2D` przyjmują parametry i zwracają wartości typu `double`. Na przykład metoda `getWidth` zwraca wartość typu `double`, nawet jeśli szerokość jest zapisana w postaci liczby typu `float` w obiekcie typu `Rectangle2D.Float`.



Aby całkowicie pozbyć się wartości typu `float`, należy używać klas typu `Double`. Jednak w programach tworzących wiele tysięcy figur warto rozważyć użycie klas `Float` ze względu na oszczędność pamięci.

Wszystko, co napisaliśmy do tej pory na temat klas `Rectangle2D`, dotyczy również pozostałych klas reprezentujących figury. Dodatkowo istnieje klasa o nazwie `Point2D`, której podklasy to `Point2D.Float` i `Point2D.Double`. Poniższy fragment programu tworzy obiekt takiej klasy:

```
Point2D p = new Point2D.Double(10, 20);
```



Klasa `Point2D` jest niezwykle przydatna — zastosowanie obiektów `Point2D` jest znacznie bliższe idei programowania obiektowego niż używanie oddzielnych wartości `x` i `y`. Wiele konstruktorów przyjmuje parametry typu `Point2D`. Zalecamy stosowanie obiektów tej klasy, gdzie się da — dzięki nim obliczenia geometryczne są często dużo prostsze.

Klasy `Rectangle2D` i `Ellipse2D` dziedziczą po wspólnej nadklasie `RectangularShape`. Wprawdzie elipsa nie jest prostokątna, ale można na niej opisać prostokąt (zobacz rysunek 7.10).

Rysunek 7.10.

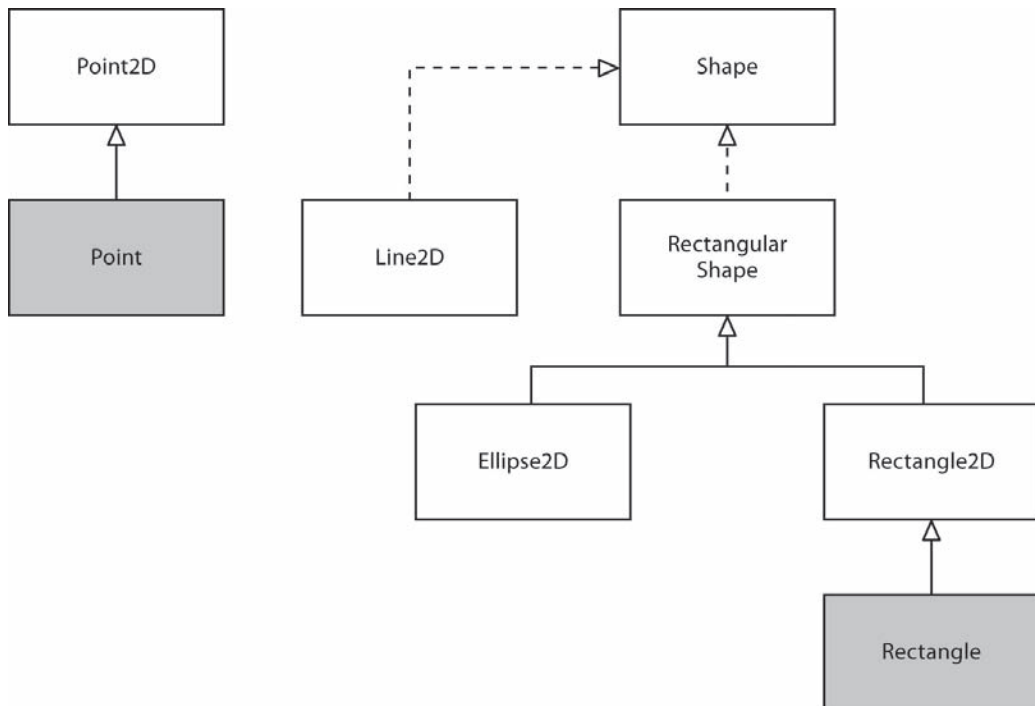
Prostokąt opisany na elipsie



Klasa `RectangularShape` definiuje ponad 20 metod wspólnych dla tych figur. Zaliczają się do nich metody `getWidth`, `getHeight`, `getCenterX` i `getCenterY` (niestety w czasie pisania tej książki nie było metody `getCenter` zwracającej obiekt typu `Point2D`).

Dodatkowo do hierarchii klas reprezentujących figury dodano kilka starszych klas z Java 1.0. Klasy `Rectangle` i `Point`, które przechowują prostokąt i punkt przy użyciu współrzędnych całkowitych, rozszerzają klasy `Rectangle2D` i `Point2D`.

Rysunek 7.11 przedstawia relacje pomiędzy klasami figur. Klasy `Double` i `Float` zostały pominięte, a klasy spadkowe wyróżniono szarym tłem.



Rysunek 7.11. Relacje między klasami figur

Tworzenie obiektów typu `Rectangle2D` i `Ellipse2D` jest prostym zadaniem. Należy podać:

- współrzędne x i y lewego górnego rogu,
- wysokość i szerokość.

W przypadku elipsy te wartości dotyczą opisanego na niej prostokąta. Na przykład instrukcja:

```
Ellipse2D e = new Ellipse2D.Double(150, 200, 100, 50);
```

utworzy elipsę wpisaną w prostokąt, którego lewy górny róg znajduje się w punkcie o współrzędnych (150, 200) o szerokości 100 i wysokości 50.

Czasami jednak współrzędne lewego górnego rogu nie są od razu dostępne. Często zdarza się, że dostępne są dwa punkty leżące naprzeciw siebie, ale nie są to rogi górny lewy i prawy dolny. Nie można utworzyć prostokąta w poniższy sposób:

```
Rectangle2D rect = new Rectangle2D.Double(px, py, qx - px, qy - py); // błąd
```

Jeśli p nie jest lewym górnym rogiem, jedna lub obie współrzędne będą miały wartości ujemne i prostokąt się nie pojawi. W takim przypadku należy najpierw utworzyć pusty prostokąt i użyć metody `setFrameFromDiagonal`:

```
Rectangle2D rect = new Rectangle2D.Double();
rect.setFrameFromDiagonal(px, py, qx, qy);
```

Jeszcze lepiej, jeśli p i q są punktami rogów reprezentowanymi przez obiekty typu `Point2D`:

```
rect.setFrameFromDiagonal(p, q);
```

Przy tworzeniu elipsy zazwyczaj znane są środek, szerokość i wysokość opisanego na niej prostokąta, a nie jego rogi (które nawet nie leżą na elipsie). Metoda `setFrameFromCenter` przyjmuje punkt środkowy, ale wymaga także jednego z czterech rogów. W związku z tym elipsę zazwyczaj tworzy się następująco:

```
Ellipse2D ellipse = new Ellipse2D.Double(centerX - width / 2, centerY - height / 2,
width, height);
```

Aby utworzyć linię, należy podać jej punkt początkowy i końcowy w postaci obiektów `Point2D` lub par liczb:

```
Line2D line = new Line2D.Double(start, end);
```

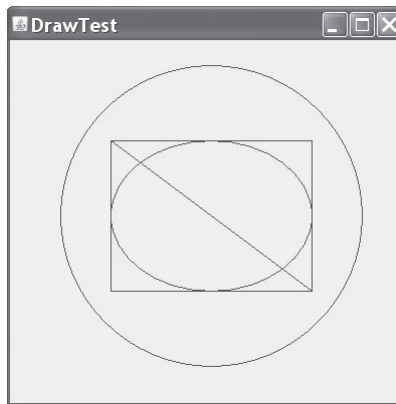
lub

```
Line2D line = new Line2D.Double(startX, startY, endX, endY);
```

Program przedstawiony na listingu 7.4 rysuje prostokąt, elipsę znajdującą się wewnątrz tego prostokąta, przekątną prostokąta oraz koło o takim samym środku jak prostokąt. Rysunek 7.12 przedstawia wynik działania tego programu.

Rysunek 7.12.

Rysowanie figur geometrycznych



Listing 7.4. draw/DrawTest.java

```
package draw;

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

/**
 * @version 1.32 2007-04-14
 * @author Cay Horstmann
 */
public class DrawTest
{
    public static void main(String[] args)
    {

```

```

       .EventQueue.invokeLater(new Runnable()
        {
            public void run()
            {
                JFrame frame = new DrawFrame();
                frame.setTitle("DrawTest");
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.setVisible(true);
            }
        });
    }
}

/**
 * Ramka zawierająca panel z rysunkami.
 */
class DrawFrame extends JFrame
{
    public DrawFrame()
    {
        add(new DrawComponent());
        pack();
    }
}

/**
 * Komponent wyświetlający prostokąty i elipsy.
 */
class DrawComponent extends JComponent
{
    private static final int DEFAULT_WIDTH = 400;
    private static final int DEFAULT_HEIGHT = 400;

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

        // Rysowanie prostokąta.

        double leftX = 100;
        double topY = 100;
        double width = 200;
        double height = 150;

        Rectangle2D rect = new Rectangle2D.Double(leftX, topY, width, height);
        g2.draw(rect);

        // Rysowanie elipsy.

        Ellipse2D ellipse = new Ellipse2D.Double();
        ellipse setFrame(rect);
        g2.draw(ellipse);

        // Rysowanie przekątnej.

        g2.draw(new Line2D.Double(leftX, topY, leftX + width, topY + height));
    }
}

```

```

// Rysowanie koła z takim samym środkiem.

double centerX = rect.getCenterX();
double centerY = rect.getCenterY();
double radius = 150;

Ellipse2D circle = new Ellipse2D.Double();
circle setFrameFromCenter(centerX, centerY, centerX + radius, centerY + radius);
g2.draw(circle);
}
public Dimension getPreferredSize() { return new Dimension(DEFAULT_WIDTH,
DEFAULT_HEIGHT); }
}

```

java.awt.geom.RectangularShape **1.2**

- double getCenterX()
- double getCenterY()
- double getMinX()
- double getMinY()
- double getMaxX()
- double getMaxY()

Zwraca współrzędną *x* lub *y* punktu środkowego, punktu o najmniejszych lub największych współrzędnych prostokąta.

- double getWidth()
- double getHeight()

Zwraca szerokość lub wysokość prostokąta.

- double getX()
- double getY()

Zwraca współrzędną *x* lub *y* lewego górnego rogu prostokąta.

java.awt.geom.Rectangle2D.Double **1.2**

- Rectangle2D.Double(double x, double y, double w, double h)

Tworzy prostokąt z lewym górnym rogiem w podanym miejscu i o podanej szerokości i długości.

java.awt.geom.Rectangle2D.Float **1.2**

- Rectangle2D.Float(float x, float y, float w, float h)

Tworzy prostokąt z lewym górnym rogiem w podanym miejscu i o podanej szerokości i długości.

```
java.awt.geom.Ellipse2D.Double 1.2
```

- `Ellipse2D.Double(double x, double y, double w, double h)`

Rysuje elipsę wpisaną w prostokąt, którego lewy górny róg znajduje się w podanym miejscu i który ma określone wysokość oraz szerokość.

```
java.awt.geom.Point2D.Double 1.2
```

- `Point2D.Double(double x, double y)`

Rysuje punkt o podanych współrzędnych.

```
java.awt.geom.Line2D.Double 1.2
```

- `Line2D.Double(Point2D start, Point2D end)`
- `Line2D.Double(double startX, double startY, double endX, double endY)`

Rysuje linię między dwoma podanymi punktami.

7.6. Kolory

Metoda `setPaint` z klasy `Graphics2D` ustawia kolor, który jest stosowany we wszystkich kolejnych rysunkach graficznych. Na przykład:

```
g2.setPaint(Color.RED);
g2.drawString("Uwaga!", 100, 100);
```

Figury zamknięte (np. prostokąt czy elipsa) można w takiej sytuacji wypełnić za pomocą metody `fill` (zamiast `draw`):

```
Rectangle2D rect = . . . ;
g2.setPaint(Color.RED);
g2.fill(rect); // Wypełnienie prostokąta rect kolorem czerwonym.
```

Aby zastosować kilka kolorów, należy wybrać kolor, zastosować metodę `draw` lub `fill`, a następnie wybrać inny kolor i ponownie zastosować metodę `draw` lub `fill`.



Metoda `fill` rysuje o jeden piksel mniej po prawej i na dole. Jeśli na przykład narysujemy prostokąt `new Rectangle2D.Double(0, 0, 10, 20)`, to rysunek będzie obejmował piksele o współrzędnych $x = 10$ i $y = 20$. Jeśli wypełnimy ten prostokąt kolorem, piksele te nie zostaną pokolorowane.

Do definiowania kolorów służy klasa `java.awt.Color`. W klasie tej dostępnych jest 13 następujących predefiniowanych stałych reprezentujących kolory:

```
BLACK, BLUE, CYAN, DARK_GRAY, GRAY, GREEN, LIGHT_GRAY, MAGENTA, ORANGE, PINK, RED,
↳ WHITE, YELLOW
```




Przed Java SE 1.4 stałe określające kolory były pisane małymi literami, np. `Color.red`. Było to sprzeczne z przyjętą konwencją pisania stałych wielkimi literami. Obecnie nazwy tych stałych można pisać wielkimi lub, ze względu na zgodność wsteczną, małymi literami.

Niestandardowy kolor można zdefiniować, tworząc obiekt klasy `Color` i podając wartości trzech składowych: czerwonego, zielonego i niebieskiego. Wartość każdego ze składników (zajmujących po jednym bajcie) musi należeć do zbioru 0 – 255. Poniższy kod przedstawia sposób wywołania konstruktora klasy `Color` z parametrami określającymi stopień czerwieni, niebieskiego i zieleni:

```
Color(int redness, int greenness, int blueness)
```

Poniżej znajduje się przykładowa procedura tworząca niestandardowy kolor:

```
g2.setPaint(new Color(0, 128, 128)); //niebieskozielony
g2.drawString("Witaj!", 75, 125);
```



Poza jednolitymi kolorami można także stosować bardziej skomplikowane ustawienia, jak różne odcienie czy obrazy. Więcej informacji na ten temat znajduje się w drugim tomie w rozdziale o zaawansowanych technikach AWT. Jeśli zamiast obiektu typu `Graphics2D` zostanie użyty obiekt `Graphics`, kolory należy ustawiać za pomocą metody `setColor`.

Do ustawiania **koloru tła** służy metoda `setBackground` z klasy `Component`, będącej nadklasą klasy `JComponent`.

```
MyComponent p = new MyComponent();
p.setBackground(Color.PINK);
```

Istnieje też metoda `setForeground`, która określa kolor elementów rysowanych na komponencie.



Metody `brighter()` i `darker()` — jak sama nazwa wskazuje — sprawiają, że aktualnie używany kolor staje się jaśniejszy bądź ciemniejszy. Ponadto metoda `brighter` jest dobrym sposobem na wyróżnienie wybranego elementu. W rzeczywistości metoda ta nieznacznie rozjaśnia kolor. Aby kolor był dużo jaśniejszy, można tę metodę zastosować trzy razy: `c.brighter().brighter().brighter()`.

Znacznie więcej predefiniowanych nazw kolorów znajduje się w klasie `SystemColor`. Stałe tej klasy określają kolory stosowane do rozmaitych elementów systemu użytkownika. Na przykład instrukcja:

```
p.setBackground(SystemColor.window)
```

ustawia kolor tła komponentu na domyślny dla wszystkich okien w systemie użytkownika (tło jest wstawiane przy każdym rysowaniu okna). Kolory zdefiniowane w klasie `SystemColor` są szczególnie przydatne, kiedy chcemy narysować elementy interfejsu użytkownika nieodbiegające kolorystyką od standardowych elementów w systemie. Tabela 7.1 zawiera nazwy kolorów systemowych oraz ich opisy.

Tabela 7.1. System kolorów

Nazwa	Zastosowanie
desktop	Kolor tła pulpitu
activeCaption	Kolor belki tytułowej aktywnego okna
activeCaptionText	Kolor tekstu na belce tytułowej
activeCaptionBorder	Kolor obramowania aktywnej belki
inactiveCaption	Kolor nieaktywnej belki
inactiveCaptionText	Kolor tekstu nieaktywnej belki
inactiveCaptionBorder	Kolor obramowania nieaktywnej belki
window	Tło okna
windowBorder	Kolor obramowania okna
windowText	Kolor tekstu w oknie
menu	Tło menu
menuText	Kolor tekstu w menu
text	Kolor tła tekstu
textText	Kolor tekstu
textInactiveText	Kolor tekstu nieaktywnych elementów sterujących
textHighlight	Kolor tła wyróżnionego tekstu
textHighlightText	Kolor wyróżnionego tekstu
control	Kolor tła elementów sterujących
controlText	Kolor tekstu w elementach sterujących
controlLtHighlight	Słabe wyróżnienie elementów sterujących
controlHighlight	Silne wyróżnienie elementów sterujących
controlShadow	Kolor cienia elementów sterujących
controlDkShadow	Ciemniejszy kolor cienia elementów sterujących
scrollbar	Kolor tła dla suwaków
info	Kolor tła dla tekstu pomocy
infoText	Kolor tekstu pomocy

java.awt.Color **1.0**

- Color(int r, int g, int b)

Tworzy obiekt reprezentujący kolor.

Parametry:

r	Wartość barwy czerwonej
g	Wartość barwy zielonej
b	Wartość barwy niebieskiej

java.awt.Graphics 1.0

- Color getColor()
- void setColor(Color c)

Pobiera lub ustawia kolor. Wszystkie następne rysunki będą miały ten kolor.

Parametry: c Nowy kolor

java.awt.Graphics2D 1.2

- Paint getPaint()
- void setPaint(Paint p)

Pobiera lub ustawia własność paint danego kontekstu graficznego. Klasa Color implementuje interfejs Paint. W związku z tym za pomocą tej metody można ustawić atrybut paint na jednolity kolor.

- void fill(shape s)

Wypełnia figurę aktualnym kolorem.

java.awt.Component 1.0

- Color getBackground()
- void setBackground(Color c)

Pobiera lub ustawia kolor tła.

Parametry: c Nowy kolor tła

- Color getForeground()
- void setForeground(Color c)

Pobiera lub ustawia kolor frontu.

Parametry: c Nowy kolor frontu

7.7. Czcionki

Program przedstawiony na początku tego rozdziału wyświetlał łańcuch tekstu pisany domyślną czcionką. Często jednak zdarza się, że tekst musi być napisany inną czcionką. Identyfikatorem czcionki jest jej **nazwa**. Nazwa czcionki składa się z **nazwy rodziny czcionek**, np. Helvetica, i opcjonalnego przyrostka, np. Bold. Na przykład nazwy Helvetica i Helvetica Bold należą do rodziny czcionek o nazwie Helvetica.

Aby sprawdzić, jakie czcionki są dostępne w danym komputerze, należy wywołać metodę `getAvailableFontFamilyNames` z klasy `GraphicsEnvironment`. Ta metoda zwraca tablicę nazw wszystkich dostępnych czcionek w postaci łańcuchów. Egzemplarz klasy `GraphicsEnvironment` reprezentujący środowisko graficzne systemu użytkownika można utworzyć za

pomocą statycznej metody `getLocalGraphicsEnvironment`. Poniższy program drukuje nazwy wszystkich czcionek znajdujących się w systemie:

```
import java.awt.*;

public class ListFonts
{
    public static void main(String[] args)
    {
        String[] fontNames = GraphicsEnvironment
            .getLocalGraphicsEnvironment()
            .getAvailableFontFamilyNames();
        for (String fontName : fontNames)
            System.out.println(fontName);
    }
}
```

W jednym z systemów początek tej listy wygląda następująco:

```
Abadi MT Condensed Light
Arial
Arial Black
Arial Narrow
Arioso
Baskerville
Binner Gothic
. . .
```

Lista ta zawiera ponad 70 pozycji.

Nazwy czcionek mogą być znakami towarowymi, a ich projekty mogą w niektórych jurysdykcjach podlegać prawom autorskim. W związku z tym dystrybucja czcionek często wiąże się z uiszczaniem opłat licencyjnych ich właścicielom. Oczywiście, podobnie jak są tanie podróbki drogich perfum, istnieją też podróbki czcionek imitujące oryginały. Na przykład imitacja czcionki Helvetica w systemie Windows nosi nazwę Arial.

Jako wspólny punkt odniesienia w bibliotece AWT zdefiniowano pięć **logicznych** nazw czcionek:

```
SansSerif
Serif
Monospaced
Dialog
DialogInput
```

Czcionki te są zawsze zamieniane na czcionki, które znajdują się w danym urządzeniu. Na przykład w systemie Windows czcionka SansSerif jest zastępowana czcionką Arial.

Dodatkowo pakiet SDK firmy Sun zawsze zawiera trzy rodziny czcionek: Lucida Sans, Lucida Bright i Lucida Sans Typewriter.

Aby narysować znak daną czcionką, najpierw trzeba utworzyć obiekt klasy `Font`. Konieczne jest podanie nazwy i stylu czcionki oraz rozmiaru w punktach drukarskich. Poniższa instrukcja tworzy obiekt klasy `Font`:

```
Font sansbold14 = new Font("SansSerif", Font.BOLD, 14);
```

Trzeci argument określa rozmiar w punktach. Jednostka ta jest powszechnie stosowana w typografii do określania rozmiaru czcionek. Jeden punkt jest równy 1/72 cala, czyli około 0,35 mm.

W konstruktorze klasy `Font` można użyć logicznej nazwy czcionki zamiast nazwy fizycznej. Styl (zwykły, **pogrubiony**, *kursywa* lub **pogrubiona kursywa**) określa drugi argument konstruktora `Font`, który może mieć jedną z poniższych wartości:

```
Font.PLAIN
Font.BOLD
Font.ITALIC
Font.BOLD + Font.ITALIC
```



Sposób odwzorowania logicznych nazw czcionek na fizyczne jest określony w pliku `fontconfig.properties` w katalogu `jre/lib` znajdującym się w folderze instalacji Javy. Informacje na temat tego pliku znajdują się pod adresem <http://docs.oracle.com/javase/7/docs/technotes/guides/intl/fontconfig.html>.

Pliki czcionek można wczytywać w formatach TrueType lub PostScript type 1. Potrzebny jest do tego strumień wejściowy dla danej czcionki — zazwyczaj z pliku lub adresu URL (więcej informacji na temat strumieni znajduje się w rozdziale 1. drugiego tomu). Następnie należy wywołać statyczną metodę `Font.createFont`:

```
URL url = new URL("http://www.fonts.com/Wingbats.ttf");
InputStream in = url.openStream();
Font f1 = Font.createFont(Font.TRUETYPE_FONT, in);
```

Zastosowana została zwykła czcionka o rozmiarze 1 punktu. Do określenia żadanego rozmiaru czcionki należy użyć metody `deriveFont`:

```
Font f = f1.deriveFont(14.0F);
```



Istnieją dwie przeciążone wersje metody `deriveFont`. Jedna z nich (przyjmująca parametr typu `float`) ustawia rozmiar czcionki, a druga (przyjmująca parametr typu `int`) ustawia styl czcionki. W związku z tym instrukcja `f1.deriveFont(14)` ustawia styl, a nie rozmiar czcionki! Styl czcionki będzie w tym przypadku kursywą, ponieważ binarna reprezentacja liczby 14 ustawia bit `ITALIC`.

Fonty Javy zawierają symbole i znaki ASCII. Na przykład znak `\u2297` fontu `Dialog` to znak \otimes . Dostępne są tylko te symbole, które zdefiniowano w zestawie znaków Unicode.

Poniższy fragment programu wyświetla napis *Witaj, świecie!* standardową czcionką bezszeryfową systemu z zastosowaniem pogrubienia i o rozmiarze 14 punktów:

```
Font sansbold14 = new Font("SansSerif", Font.BOLD, 14);
g2.setFont(sansbold14);
String message = "Witaj, świecie!";
g2.drawString(message, 75, 100);
```

Teraz **wypośredkujemy** nasz napis w zawierającym go komponencie. Do tego celu potrzebne są informacje o szerokości i wysokości łańcucha w pikselach. O wymiarach tych decydują trzy czynniki:

- czcionka (w tym przypadku jest to pogrubiona czcionka bezszeryfowa o rozmiarze 14 punktów),
- łańcuch (w tym przypadku *Witaj, świecie!*),
- urządzenie, na którym łańcuch będzie wyświetlany (w tym przypadku ekran monitora).

Obiekt reprezentujący własności czcionki urządzenia z ekranem tworzymy za pomocą metody `getFontRenderContext` z klasy `Graphics2D`. Zwraca ona obiekt klasy `FontRenderContext`. Obiekt ten należy przekazać metodzie `getStringBounds` z klasy `Font`:

```
FontRenderContext context = g2.getFontRenderContext();
Rectangle2D bounds = f.getStringBounds(message, context);
```

Metoda `getStringBounds` zwraca prostokąt, w którym mieści się łańcuch.

Do interpretacji wymiarów tego prostokąta potrzebna jest znajomość podstawowych pojęć z zakresu składu tekstów (zobacz rysunek 7.13). **Linia bazowa** (ang. *baseline*) to teoretyczna linia, na której opiera się dolna część litery, np. *e*. **Wydlużenie górne** (ang. *ascent*) to odstęp dzielący linię bazową i **linią górną pisma** (ang. *ascender*), która określa górną granicę liter, takich jak *b* lub *k* czy też wielkich liter. **Wydlużenie dolne** (ang. *descent*) to odległość pomiędzy linią bazową a **linią dolną pisma** (ang. *descender*), która stanowi granicę dolnej części takich liter jak *p* lub *g*.



Rysunek 7.13. Pojęcia z zakresu składu tekstów

Interlinia (ang. *leading*) to odstęp pomiędzy wydlużeniem dolnym jednej linii a wydlużeniem górnym następnego wiersza (termin pochodzi od pasków ołowiu używanych przez zecerów do oddzielania linii). **Wysokość** (ang. *height*) czcionki to odległość pomiędzy następującymi po sobie liniami bazowymi i jest równa sumie wydlużenia dolnego, *leadingu* i wydlużenia górnego.

Szerokość prostokąta zwracanego przez metodę `getStringBounds` określa szerokość tekstu. Wysokość natomiast jest równa sumie wydlużenia dolnego, *leadingu* i wydlużenia górnego. Prostokąt ma swój początek na linii bazowej łańcucha. Górna współrzędna *y* prostokąta ma wartość ujemną. W związku z tym szerokość, wysokość i wydlużenie górne łańcucha można sprawdzić następująco:

```
double stringWidth = bounds.getWidth();
double stringHeight = bounds.getHeight();
double ascent = -bounds.getY();
```

Aby sprawdzić wydlużenie dolne lub *leading*, należy użyć metody `getLineMetrics` klasy `Font`. Zwraca ona obiekt klasy `LineMetrics`, dysponujący metodami do sprawdzania wydlużenia dolnego i *leadingu*:

```
LineMetrics metrics = f.getLineMetrics(message, context);
float descent = metrics.getDescent();
float leading = metrics.getLeading();
```

W poniższym fragmencie programu wykorzystano wszystkie opisane powyżej informacje do umieszczenia łańcucha na środku zawierającego go komponentu:

```
FontRenderContext context = g2.getFontRenderContext();
Rectangle2D bounds = f.getStringBounds(message, context);

// (x, y) = lewy górny róg tekstu
double x = (getWidth() - bounds.getWidth()) / 2;
double y = (getHeight() - bounds.getHeight()) / 2;

// Dodanie wydłużenia górnego do y w celu sięgnięcia do linii bazowej.
double ascent = -bounds.getY();
double baseY = y + ascent;
g2.drawString(message, (int) x, (int) baseY);
```

Aby ułatwić sobie zrozumienie techniki wyśrodkowywania tekstu, warto sobie uzmysłowić, że metoda `getWidth()` zwraca szerokość komponentu. Pewna część tej przestrzeni, `bounds.getWidth()`, jest zajmowana przez tekst. Reszta powinna być podzielona na dwie równe części, rozmieszczone po obu stronach tekstu. Ten sam sposób rozumowania dotyczy wysokości.



Kiedy konieczne jest obliczenie wymiarów układu bez użycia metody `paintComponent`, nie można uzyskać obiektu obrazowania czcionki typu `Graphics2D`. W zamian należy wywołać metodę `getFontMetrics` klasy `JComponent`, a następnie metodę `getFontRenderContext`.

```
FontRenderContext context = getFontMetrics(f).getFontRenderContext();
```

Przykładowy program przedstawiony poniżej nie tylko drukuje napis, ale także linię bazową i prostokąt otaczający napis. Rysunek 7.14 przedstawia wynik działania tego programu. Listing 7.5 zawiera jego kod.

Rysunek 7.14.
Linia bazowa
i prostokąt
otaczający
łańcuch



Listing 7.5. font/FontTest.java

```
package font;

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

/**
 * @version 1.33 2007-04-14
```

```

    *@author Cay Horstmann
    */
public class FontTest
{
    public static void main(String[] args)
    {
        EventQueue.invokeLater(new Runnable()
        {
            public void run()
            {
                JFrame frame = new FontFrame();
                frame.setTitle("FontTest");
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.setVisible(true);
            }
        });
    }
}

/**
 * Ramka z komponentem zawierającym tekst.
 */
class FontFrame extends JFrame
{
    public FontFrame()
    {
        add(new FontComponent());
        pack();
    }
}

/**
 * Komponent z tekstem w ramce na środku.
 */
class FontComponent extends JComponent
{
    private static final int DEFAULT_WIDTH = 300;
    private static final int DEFAULT_HEIGHT = 200;

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

        String message = "Witaj, świecie!";

        Font f = new Font("Serif", Font.BOLD, 36);
        g2.setFont(f);

        // Sprawdzenie rozmiaru tekstu.

        FontRenderContext context = g2.getFontRenderContext();
        Rectangle2D bounds = f.getStringBounds(message, context);

        // set (x, y) = lewy górny róg tekstu

        double x = (getWidth() - bounds.getWidth()) / 2;
        double y = (getHeight() - bounds.getHeight()) / 2;
    }
}

```



```

// Dodanie wydłużenia górnego do y w celu sięgnięcia do linii bazowej.
double ascent = -bounds.getY();
double baseY = y + ascent;

// Rysowanie komunikatu.
g2.drawString(message, (int) x, (int) baseY);

g2.setPaint(Color.LIGHT_GRAY);

// Rysowanie linii bazowej.
g2.draw(new Line2D.Double(x, baseY, x + bounds.getWidth(), baseY));

// Rysowanie otaczającego tekst prostokąta.
Rectangle2D rect = new Rectangle2D.Double(x, y, bounds.getWidth(),
↳ bounds.getHeight());
g2.draw(rect);
}
public Dimension getPreferredSize() { return new Dimension(DEFAULT_WIDTH,
DEFAULT_HEIGHT); }
}

```

java.awt.Font **1.0**

- Font(String name, int style, int size)

Tworzy obiekt reprezentujący czcionkę.

Parametry:	name	Nazwa czcionki — może być nazwa typu Helvetica Bold lub logiczna nazwa typu Serif lub SansSerif
	style	Styl: Font.PLAIN, Font.BOLD, Font.ITALIC lub Font.BOLD + Font.ITALIC
	size	Rozmiar w punktach (na przykład 12)

- String getFontName()

Pobiera nazwę czcionki (typu Helvetica Bold).

- String getFamily()

Pobiera nazwę rodziny czcionek (np. Helvetica).

- String getName()

Pobiera nazwę logiczną (np. SansSerif), jeśli czcionka została utworzona z nazwy logicznej. W przeciwnym przypadku zwraca nazwę czcionki.

- Rectangle 2D getStringBounds(String s, FontRenderContext context) **1.2**

Zwraca prostokąt otaczający łańcuch. Prostokąt ma swój początek na linii bazowej łańcucha. Górna współrzędna y prostokąta ma wartość równą odwrotności wydłużenia górnego. Wysokość prostokąta jest równa sumie wydłużenia górnego, dolnego i leadingu. Szerokość jest równa szerokości tekstu.

- `LineMetrics getLineMetrics(String s, FontRenderContext context)` **1.2**

Zwraca ona obiekt klasy `LineMetrics` dysponujący metodami do sprawdzania wydłużenia dolnego i leadingu.

- `Font deriveFont(int style)` **1.2**
- `Font deriveFont(float size)` **1.2**
- `Font deriveFont(int style, float size)` **1.2**

Zwraca nową czcionkę różniącą się od aktualnej tylko rozmiarem podanym jako argument.

`java.awt.font.LineMetrics` **1.2**

- `float getAscent()`
Pobiera wydłużenie górne czcionki — odległość linii bazowej od wierzchołków wielkich liter.
- `float getDescent()`
Pobiera wydłużenie dolne czcionki — odległość linii bazowej od podstaw liter sięgających dolnej linii pisma.
- `float getLeading()`
Pobiera leading czcionki — odstęp pomiędzy spodem jednej linii tekstu a wierzchołkiem następnej.
- `float getHeight()`
Pobiera całkowitą wysokość czcionki — odległość pomiędzy dwiema liniami bazowymi tekstu (wydłużenie dolne + leading + wydłużenie górne).

`java.awt.Graphics` **1.0**

- `Font getFont()`
- `void setFont(Font font)`
Pobiera lub ustawia czcionkę. Czcionka ta będzie stosowana w kolejnych operacjach rysowania tekstu.
Parametry: `font` Czcionka
- `void drawString(String str, int x, int y)`
Rysuje łańcuch przy użyciu aktualnej czcionki i koloru.
Parametry: `str` Łańcuch
 `x` Współrzędna x początku łańcucha
 `y` Współrzędna y linii bazowej łańcucha

```
java.awt.Graphics2D 1.2
```

- `FontRenderContext getFontRenderContext()`

Pobiera kontekst wizualizacji czcionki, który określa cechy czcionki w kontekście graficznym.

- `void drawString(String str, float x, float y)`

Rysuje łańcuch przy zastosowaniu aktualnej czcionki i koloru.

Parametry:	<code>str</code>	Łańcuch
	<code>x</code>	Współrzędna x początku łańcucha
	<code>y</code>	Współrzędna y linii bazowej łańcucha

```
javax.swing.JComponent 1.2
```

- `FontMetrics getFontMetrics(Font f)` 5.0

Pobiera cechy czcionki. Klasa `FontMetrics` jest prekursorem klasy `LineMetrics`.

```
java.awt.FontMetrics 1.0
```

- `FontRenderContext getFontRenderContext()` 1.2

Pobiera kontekst wizualizacji czcionki.

7.8. Wyświetlanie obrazów

Poznaliśmy techniki tworzenia prostych rysunków składających się z linii i figur geometrycznych. Bardziej złożone obrazy, jak zdjęcia, mają zazwyczaj inne pochodzenie, np. przenosi się je do komputera za pomocą skanera lub wytwarza w wyspecjalizowanym do tego celu oprogramowaniu. W drugim tomie nauczymy się tworzyć obrazy złożone z pojedynczych pikseli zapisanych w tablicy — technika ta jest często używana na przykład podczas tworzenia obrazów fraktalnych.

Obrazy zapisane w postaci plików na dysku lub w internecie można wczytać do aplikacji w Javie i wyświetlić na obiektach `Graphics`. Obrazy można wczytywać na wiele sposobów. W poniższym przykładzie użyta jest znana nam już klasa `ImageIcon`:

```
Image image = new ImageIcon(filename).getImage();
```

Zmienna `image` zawiera referencję do obiektu opakowującego obraz. Można go wyświetlić za pomocą metody `drawImage` z klasy `Graphics`:

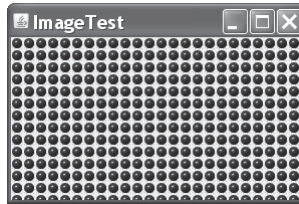
```
public void paintComponent(Graphics g)
{
    ...
    g.drawImage(image, x, y, null);
}
```

Program z listingu 7.6 robi nawet więcej, ponieważ wypełnia całe okno wieloma obrazami. Rezultat tego widać na rysunku 7.15. Za to kaskadowe wypełnienie odpowiedzialna jest metoda `paintComponent`. Najpierw rysujemy jeden obraz w lewym górnym rogu, a następnie zapełniamy całe okno za pomocą metody `copyArea`:

```
for (int i = 0; i * imageWidth <= getWidth(); i++)
    for (int j = 0; j * imageHeight <= getHeight(); j++)
        if (i + j > 0)
            g.copyArea(0, 0, imageWidth, imageHeight, i * imageWidth, j * imageHeight);
```

Rysunek 7.15.

Okno wypełnione kopiami jednego obrazu



Listing 7.6 przedstawia pełny kod źródłowy opisywanego program.

Listing 7.6. image/ImageTest.java

```
package image;

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

/**
 * @version 1.33 2007-04-14
 * @author Cay Horstmann
 */
public class ImageTest
{
    public static void main(String[] args)
    {
        EventQueue.invokeLater(new Runnable()
        {
            public void run()
            {
                JFrame frame = new ImageFrame();
                frame.setTitle("ImageTest");
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.setVisible(true);
            }
        });
    }
}

/**
 * Ramka zawierająca komponent obrazu.
 */
class ImageFrame extends JFrame
{
    public ImageFrame()
    {
        add(new ImageComponent());
    }
}
```

```

        pack();
    }
}

/**
 *Komponent wyświetlający powielony obraz.
 */
class ImageComponent extends JComponent
{
    private static final int DEFAULT_WIDTH = 300;
    private static final int DEFAULT_HEIGHT = 200;

    private Image image;

    public ImageComponent()
    {
        image = new ImageIcon("blue-ball.gif").getImage();
    }

    public void paintComponent(Graphics g)
    {
        if (image == null) return;

        int imageWidth = image.getWidth(this);
        int imageHeight = image.getHeight(this);

        // Rysowanie obrazu w lewym górnym rogu.

        g.drawImage(image, 0, 0, null);
        // Powielenie obrazu w obrębie komponentu.

        for (int i = 0; i * imageWidth <= getWidth(); i++)
            for (int j = 0; j * imageHeight <= getHeight(); j++)
                if (i + j > 0) g.copyArea(0, 0, imageWidth, imageHeight, i * imageWidth, j
                    * imageHeight);
    }
    public Dimension getPreferredSize() { return new Dimension(DEFAULT_WIDTH,
        ↪DEFAULT_HEIGHT); }
}

```

```
java.awt.Graphics 1.0
```

- `boolean drawImage(Image img, int x, int y, ImageObserver observer)`

Rysuje obraz w naturalnym rozmiarze. Uwaga: to wywołanie może zwrócić wartość przed narysowaniem obrazu.

Parametry:	<code>img</code>	Obraz do narysowania
	<code>x</code>	Współrzędna x lewego górnego rogu
	<code>y</code>	Współrzędna y lewego górnego rogu
	<code>observer</code>	Obiekt powiadamiający o postępie procesu wizualizacji (może być wartość <code>null</code>)

- `boolean drawImage(Image img, int x, int y, int width, int height, ImageObserver observer)`

Rysuje obraz o zmienionych wymiarach. System dopasowuje rozmiar obrazu do obszaru o podanej szerokości i wysokości. Uwaga: to wywołanie może zwrócić wartość przed narysowaniem obrazu.

Parametry:	<code>img</code>	Obraz do narysowania
	<code>x</code>	Współrzędna <code>x</code> lewego górnego rogu
	<code>y</code>	Współrzędna <code>y</code> lewego górnego rogu
	<code>width</code>	Szerokość obrazu
	<code>height</code>	Wysokość obrazu
	<code>observer</code>	Obiekt powiadamiający o postępie procesu wizualizacji (może być wartość <code>null</code>)

- `void copyArea(int x, int y, int width, int height, int dx, int dy)`

Kopiuje obszar ekranu.

Parametry:	<code>x</code>	Współrzędna <code>x</code> lewego górnego rogu obszaru źródłowego
	<code>y</code>	Współrzędna <code>y</code> lewego górnego rogu obszaru źródłowego
	<code>width</code>	Szerokość obszaru źródłowego
	<code>height</code>	Wysokość obszaru źródłowego
	<code>dx</code>	Odległość w poziomie od obszaru źródłowego do obszaru docelowego
	<code>dy</code>	Odległość w pionie od obszaru źródłowego do obszaru docelowego

Na tym zakończymy wprowadzenie do grafiki w Javie. Bardziej zaawansowane techniki, takie jak grafika 2D i obróbka obrazów, zostały opisane w drugim tomie. W kolejnym rozdziale dowiemy się, jak programy reagują na dane wprowadzane przez użytkownika.

Skorowidz

A

- abstrakcja, 214
- ActiveX, 26, 35, 540
- adnotacja, 111, 640
 - @SafeVarargs, 643
 - @SuppressWarnings, 643, 648
- adres URL, 527, 546
- agregacja, 135
- akcelerator, 439
- akcesorium podglądu, 499
- akcesory, 142
- akcje, 355, 373
- aktualizacje, updates, 39
- aktualizowanie preferencji, 558
- aktywność komponentu, 375
- algorytm, 132, 718
 - binarySearch, 723
 - obliczania kodu mieszającego, 226
 - QuickSort, 121, 721
 - znajdujący największy element, 718
- algorytmy
 - sortujące, 720
 - w klasie Collections, 724
- alokacja listy tablicowej, 235
- analiza
 - funkcjonalności klasy, 252
 - MVC, 397
 - obiektów w czasie działania programu, 257
- animacja piłki, 736–742
- animowane gify, 546
- anonimowe klasy wewnętrzne, 289, 300, 363
- API
 - Javy, 33
 - JNLP, 525
 - Logging, 591
 - Preferences, 555, 595
 - String, 83
- aplet, 28, 34, 53, 54
- aplet, 511, 533
 - Jmol, 29
 - WelcomeApplet, 53
- aplety
 - implementacja, 533
 - komunikacja, 547
 - konwersja programów, 536
 - obrazy, 546
 - pliki audio, 546
 - środowisko działania, 547
 - uruchamianie, 535
- aplikacja
 - ImageViewer, 51
 - Java Web Start, 519
 - WebStartCalculator, 528
- aplikacje
 - graficzne, 50
 - serwerowe, 29
- architektura, framework, 706
 - kolekcji, 706, 710
 - model-widok-kontroler, 396
- argument, 61
- ASCII, 65
- asercje, 587
 - wyłączanie, 588
 - zastosowania, 589, 590
- asocjacja, 135
- atak, 25
- atrybut
 - classid, 540
 - codebase, 541
 - codetype, 540

atrybuty
 pozycjonujące, 540
 znacznika applet, 537–540
 znacznika param, 541, 542
autoboxing, 32
automatyczna konwersja typów, 32
automatyczne opakowywanie, 241
AWT, Abstract Window Toolkit, 314

B

bariera cykliczna, 814
bariery, 814
bazowy katalog drzewa pakietu, 187
bezpieczeństwo, 25, 35
 typów, 639, 649
 wątków, 775
biała księga Javy, 22
biblioteka, 33
 AWT, 314, 387
 fdlibm, 74
 IFC, 314
 Java2D, 332, 333
 JFC, 314
 kolekcji, 666, 707
 refleksyjna, 247
 STL, 666
 Swing, 315, 815
biblioteki
 struktur danych, 666
 zabezpieczeń, 25
bit, 729
blok, 98
 inicjujący, 175
 try-catch, 250, 578, 585
 try-finally, 578
blokada, 762, 769
 jawna, 770
 odczytu, 783
 uczciwa, 764
 wewnętrzna, 770
 zapisu, 784
bloki synchronizowane, 774
blokowanie po stronie klienta, 774, 775
błąd
 AssertionError, 587
 pomyłki o jeden, 719
 ThreadDeath, 752
 typu, 650
 typu cannot read, 46
błędy
 danych wejściowych, 564
 kompilacji, 49, 153, 181, 629, 646
 programisty, 568

przydzielania pamięci, 25
urządzeń, 564
w Eclipse, 49
w kodzie, 565
wejścia-wyjścia, 565
wewnętrzne, 568
wykonawcze, 644
zabezpieczeń, 25
zaokrąglania, 64, 333

C

catch, 250
cechy
 języka, 22, 33
 komponentu, 393
certyfikat
 bezpieczeństwa, 524
 niebezpieczny, 525
 własny, 524
chwytywanie typu wieloznacznego, 655
ciało metody, 60
czasochłonne zadania, 816
czcionki, 343
 bezseryfowe, 346
 nazwy, 343
 nazwy logiczne, 344
 PostScript type 1, 345
 styl, 345
 TrueType, 345
 wysokość, 346

D

dane, 195
 binarne, 26
 wejściowe, 89
 wyjściowe, 91
debuger, 28, 621
debuger JSwat, 623
debugowanie, 609
debugowanie aplikacji z GUI, 614
definiowanie
 klasy, 148
 klasy ogólnej, 630
 kolorów, 340
 słuchacza, 356
 stałej klasowej, 69
 wątku, 745
 wyjątków kontrolowanych, 567
 zmiennej, 66, 68
 zmiennej obiektowej, 138
 zmiennej tablicowej, 117

- dekompilowanie pliku, 761
- dekrementacja, 71
- delegacja, 265
- delegacja zdarzeń, 356
- demony, 753
- dezaktywacja elementów menu, 441
- diagram
 - dziedziczenia klasy, 215, 534
 - dziedziczenia zdarzeń AWT, 387
 - hierarchii wyjątków, 565
 - klas, 136
 - przepływu sterowania, 100–106, 110
- dodawanie
 - akcji do menu, 375
 - elementu do listy powiązanej, 678
 - elementu do mapy, 702
 - ikony, 435
 - klasy do pakietu, 182
 - klauzuli throws, 96
 - komponentów, 401
- dokumentacja, 194
 - API, 85–87
 - JSR, 627
 - założeń, 590
- dopasowywanie typów, 659
- dopełnienie, 459
 - wewnętrzne, 452
 - zewnętrzne, 452
- dostęp
 - chroniony, 219
 - do apletu, 539
 - do elementów kolekcji, 672, 708
 - do elementów listy tablicowej, 236
 - do elementu tablicy, 117
 - do formatera, 781
 - do komponentów, 473
 - do pakietów, 518
 - do plików lokalizacyjnych, 516
 - do pliku, 96
 - do pliku JNLP, 521
 - do pól, 155
 - do pól generycznych, 637
 - do prywatnych pól nadklasy, 202
 - do sekcji krytycznej, 762
 - do stanu obiektu, 289
 - do usługi, 526
 - do wartości, 243
 - do węzła drzewa, 555
 - do zasobów lokalnych, 525
 - do zmiennej warunkowej, 775
 - do zmiennych finalnych, 297
 - jednoczesny wątek, 761
 - swobodny, 719, 723
 - wątków do struktury danych, 757
- drukowanie, 31
- drukowanie informacji o klasie, 252
- drzewo katalogów, 42
- duże liczby, big numbers, 62
- dymki, tooltips, 446
- dyrektywa
 - #include, 182
 - import, 90
- działanie
 - kontrolera, 395
 - metody transfer, 761
- dziedziczenia klasy Applet, 534
- dziedziczenie, inheritance, 133, 199, 268
 - hierarchia, 206
 - klasy abstrakcyjne, 214
 - klasy finalne, 211
 - metody finalne, 211
 - ochrona dostępu, 219
 - polimorfizm, 207
 - pomiędzy klasami par, 649
 - rzutowanie, 212
 - typów ogólnych, 649
 - wiązanie dynamiczne, 209
 - wielokrotne, 279
- dzielenie
 - całkowitoliczbowe, 69
 - moduło, 69
 - zmiennoprzecinkowe, 69
- dzienniki, 591
- dzienniki rotacyjne, 599

E

- Eclipse, 44, 47
- edycja
 - kodu źródłowego, 48
 - ścieżki dostępu, 39
- edytor tekstowy
 - Emacs, 44
 - JEdit, 44
 - TextPad, 44
- edytowalna lista rozwijalna, 423
- EE, Enterprise Edition, 38
- egzemplarz klasy, 132
- elementy menu, 432
 - aktywowanie, 440
 - dezaktywowanie, 440
 - ikony, 435
 - poła wyboru, 436
 - przełączniki, 436
- elementy tablicy, 116
- eliminacja wywołań funkcji, 27
- elipsa, 335

etykiety, 459
 HTML, 409
 komponentów, 408
 ewolucja Javy, 32

F

figury
 2D, 332
 geometryczne, 333
 filtr
 plików, 496, 497, 505
 rekordów, 600
 firma
 Oracle, 32, 34
 Sun Fellow, 30
 Sun Microsystems, 25, 34
 format
 binarny liczby, 63
 JNLP, 519
 Unicode, 26
 XML, 556
 formatery, 600
 formatowanie
 danych wyjściowych, 91
 daty, 95
 funkcja unexpected, 569
 funkcje
 czysto wirtualne, 216
 matematyczne, 73
 sieciowe, 24
 składowe, 60

G

GC, Garbage Collector, 702
 generowanie
 dokumentacji, 194
 obiektów klas ogólnych, 646
 generyczne
 klasy, 629
 listy tablicowe, 233
 generyczny kod tablicowy, 261
 graficzny interfejs użytkownika, GUI, 28, 57, 313,
 391, 614
 grafika, 313
 grupa, 754
 przycisków radiowych, 416
 wątków, 755
 zadań, 808
 GTK, 316

H

harmonogram
 wykonywania wątków, 750
 zadań, 696
 hasło, 90, 410
 hermetyzacja, 133, 155
 hierarchia
 dziedziczenia, 206
 dziedziczenia interfejsu Type, 660
 dziedziczenia klasy Component, 400
 dziedziczenia klasy JFrame, 321
 interfejsów, 278
 wyjątków, 566, 586
 zdarzeń, 387
 historia Javy, 30
 HTML, 33

I

IDE, 39, 47
 identyfikacja klas, 134
 IFC, Internet Foundation Classes, 314
 ikony, 435
 ikony komunikatów, 475
 implementacja
 apletów, 533
 ArrayList, 644
 interfejsu, 271, 273
 klasy Bank, 770
 klasy ogólnej, 629
 kolejki, 667
 import
 klas, 180
 statyczny, 182
 indeks, 123
 indeks argumentu, 95
 informacje
 o klasie, 252
 o typach, 28
 o typach czasu wykonywania, 248
 o typach generycznych, 659
 o typach obiektów, 248
 o uruchomionym programie, 613
 o zdarzeniach, 602
 inicjalizacja
 pól, 172
 pól statycznych, 176
 pól wartościami domyślnymi, 171
 tablic, 118
 z podwójną klamrą, 302
 zmiennej obiektowej, 138
 zmiennych, 68

- inkrementacja, 71
- instalacja
 - bibliotek, 41
 - dokumentacji, 41
 - filtru, 600
 - JDK, 38, 39
 - programów, 42
- instrukcja
 - break, 111–114
 - break z etykietą, 112
 - case, 111
 - continue, 113
 - do-while, 104
 - for, 77
 - goto, 111
 - if, 100, 113
 - if-else, 100
 - if-else if, 102
 - import, 180, 182
 - lock, 762
 - return, 578
 - switch, 109–111
 - try, 580, 581
 - while, 103
- instrukcje
 - sterujące, 98
 - warunkowe, 98, 109, 113
 - złożone, 99
- interfejs
 - Action, 373, 379, 433
 - ActionListener, 286, 300, 357, 364, 373, 389
 - AdjustmentListener, 389
 - AppletContext, 547
 - AutoCloseable, 580
 - BasicService, 527
 - BlockingDeque<E>, 793
 - BlockingQueue<E>, 792
 - ButtonModel, 397, 398, 417
 - Callable, 797
 - Callable<V>, 801
 - Cloneable, 282
 - Collection, 668–672, 679, 707, 711
 - Collection<E>, 672
 - Comparable, 272, 279, 308, 634, 654, 689
 - Comparator<T>, 690, 693
 - Condition, 769–771
 - Delayed, 792
 - Deque<E>, 695
 - Enumeration, 670, 727
 - ExecutorService, 803, 808
 - FileFilter, 497
 - Filter, 600
 - FocusListener, 389
 - Formattable, 92
 - Future, 798
 - Future<V>, 801
 - GenericArrayType, 660, 664
 - InvocationHandler, 307, 311
 - ItemListener, 389
 - Iterable, 117, 669
 - Iterator, 669, 677, 680
 - Iterator<E>, 674
 - klasy, 146
 - LayoutManager, 469
 - LayoutManager2, 469
 - LinkedList<E>, 683
 - List, 684, 708, 711
 - List<E>, 682
 - ListIterator, 677, 708
 - ListIterator<E>, 683
 - Lock, 764, 769, 771, 783
 - KeyListener, 389
 - Map, 707
 - Map<K, V>, 700
 - MouseListener, 441
 - MouseListener, 381, 383, 389
 - MouseMotionListener, 381, 383, 389
 - MouseWheelListener, 389
 - nasłuchu, 356
 - NavigableMap, 709
 - NavigableMap<K, V>, 716
 - NavigableSet, 709, 712
 - NavigableSet<E>, 694, 716
 - ParameterizedType, 660, 664
 - PersistenceService, 527
 - PersistentService, 528
 - Powered, 278
 - Queue, 666, 667
 - Queue<E>, 695
 - RandomAccess, 708, 721
 - Runnable, 797
 - ScheduledExecutorService, 807
 - Set, 708
 - Shape, 333
 - SortedMap, 709
 - SortedMap<K, V>, 716
 - SortedSet, 709, 712
 - SortedSet<E>, 693, 716
 - SwingConstants, 278, 408
 - Thread.UncaughtExceptionHandler, 754
 - TransferQueue, 788
 - TransferQueue<E>, 793
 - Type, 660
 - TypeVariable, 660, 664
 - WildcardType, 660, 664
 - WindowFocusListener, 389
 - WindowListener, 369, 372, 389
 - WindowStateListener, 372, 389

interfejsu, 219, 265
 hierarchia, 278
 implementacja, 271, 273
 metody, 278
 sprzężenie zwrotne, 286
 zmienne, 277

interfejsy
 architektury kolekcji, 707
 kolekcyjne, 665, 707, 719
 nasłuchowe, 389
 nasłuchujące AWT, 389
 przenośne, 27
 użytkownika, 34, 391
 znacznikowe, 282

interlinia, 346

interpreter, 27, 184

iterator jako parametr, 727

iteratory, 669, 670

J

JAR, Java Archive, 512

Java look and feel, 315

Java Micro Edition, 23

Java Plug-in, 540

Java Runtime System, 26

Java Web Start, 511, 519

JavaBeans, 247

JavaFX, 317

jawna inicjalizacja pól, 172

JDK, Java Development Kit, 37

jednostki kodowe, 66, 81

język

Algol, 165

C, 25

C#, 28, 34

C++, 23, 24

HTML, 33

J#, 28

J++, 28

Java, 22

JavaScript, 35

UML, 136

Visual Basic, 23, 137

języki

interpretowane, 34

obiektywne, 24

proceduralne, 24

JFC, Java Foundation Classes, 314

JIT, just-in-time compiler, 27

JNLP, Java Network Launch Protocol, 519

JRE, Java Runtime Environment, 38

JSR, Java Specification Requests, 627

K

kalendarz, 139

kalkulator, 403, 521

karta

HSB, 506

RGB, 506

Swatches, 506

katalog

bazowy drzewa pakietu, 187

bin, 39

com, 183

gutenberg, 820

src, 43

klas

coupling, 135

diagramy, 136

dziedziczenie, 133, 200

identyfikacja, 134

implementacja interfejsu, 271

komentarze, 191

konstruktory, 137, 152

metody, 133

metody prywatne, 157

metody statyczne, 160

nadklasy, 200

plik źródłowy, 189

podklasy, 200

pola statyczne, 159

predefiniowanie, 137

projektowanie, 195

relacje, 135

rozszerzanie, 133

pola stałe, 158

ścieżka, 187

klasa, 58, 132

ExampleFileView, 499

AbstractAction, 374, 377

AbstractButton, 419, 434–436, 440

AbstractCollection, 672

AbstractList, 726

AbstractQueue, 668

AbstractSequentialList, 723

AbstractSet, 223

AccessibleObject, 257, 261

ActionMap, 376

AnonymousInnerClassTest, 301

Applet, 533, 537, 545, 549

AppletContext, 540, 548, 549

Array, 262

ArrayAlg, 663

ArrayBlockingQueue<E>, 791

ArrayDeque, 694

ArrayDeque<E>, 696

- ArrayDequeue, 668
- ArrayList, 234–236, 240, 628, 642, 674, 684
- ArrayList<T>, 650
- ArrayListTest, 238
- Arrays, 118, 121, 123
- AtomicInteger, 777
- AWTEvent, 387
- BallRunnable, 742
- BasicButtonUI, 398
- BasicService, 528, 531
- bazowa Object, 133, 220
- BigDecimal, 64, 114, 116
- BigInteger, 114, 115
- BigIntegerTest, 115
- BitSet, 726, 729
- BlockingQueueTest, 789
- BorderFactory, 419, 421
- BorderLayout, 401, 402
- BounceFrame, 737
- BuggyButtonTest, 622
- ButtonFrame, 360
- ButtonGroup, 417, 418
- ButtonModel, 417, 418
- ButtonUIListener, 398
- Calendar, 140
- CalendarTest, 145
- CheckBoxTest, 415
- CircularArrayQueue, 668
- Class, 233, 248–251, 255, 261, 658
- Class<T>, 658, 663
- CloneTest, 284
- Collections, 679, 711, 715, 724
- Color, 340, 342, 343
- ColorAction, 360, 362
- Component, 322, 325, 332, 343, 399, 408
- ConcurrentHashMap, 794
- ConcurrentHashMap<K, V> 5.0, 795
- ConcurrentLinkedQueue<E>, 795
- ConcurrentSkipListMap, 794
- ConcurrentSkipListSet<E>, 795
- Console, 90, 91
- ConsoleHandler, 600, 607
- Constructor, 250–252, 256, 658
- ConstructorTest, 177
- Container, 362, 399
- CopyOfTest, 263
- CountDownLatch, 813
- Cursor, 381
- CyclicBarrier, 814
- Date, 139, 140
- DateFormatSymbols, 144, 148
- DateInterval, 638
- Dimension, 330
- Double, 276
- DrawTest, 337
- Ellipse2D, 335
- Ellipse2D.Double, 340
- Employee, 148, 151, 163, 184, 639
- EmployeeSortTest, 275
- EmployeeTest, 149
- Enum, 246
- EnumMap, 704
- EnumMap<K extends Enum<K>, V>, 706
- EnumSet, 704
- EnumSet<E extends Enum<E>>, 706
- EnumTest, 246
- EOFException, 570
- EqualsTest, 230
- Error, 565
- EventHandler, 364, 365
- EventObject, 363, 387
- EventTracer, 616
- Exception, 251, 565, 583
- Exchanger, 814
- Executors, 802
- ExtendedService, 527
- Field, 252, 256, 258, 261, 265
- File, 497
- FileContents, 531
- FileFilter, 504
- FileHandler, 600, 607
- FileInputStream, 567
- FileNameExtensionFilter, 505
- FileOpenService, 526, 532
- FileSaveService, 532
- FileView, 497, 505
- Filter, 609
- FlowLayout, 400
- Font, 345, 349
- FontMetrics, 351
- FontParamApplet, 541
- FontRenderContext, 346
- FontTest, 348
- ForkJoinTest, 811
- Formatter, 600, 609
- Frame, 318, 326
- FutureTask<V>, 802
- FutureTest, 799
- GenericReflectionTest, 661
- Graphics, 332, 343, 350, 353
- Graphics2D, 332, 333, 343, 351
- GraphicsDevice, 325, 617
- GraphicsEnvironment, 344, 620
- GregorianCalendar, 139–143, 146, 654
- GridBagConstraints, 454, 458
- GridLayout, 399, 405
- GroupLayout, 459, 463, 466
- Handler, 598, 607

klasa

- HashMap<K, V>, 701
- HashSet, 684, 686
- HashSet<E>, 687
- Hashtable, 709, 726
- IdentityHashMap, 705
- IdentityHashMap<K, V>, 706
- ImageIcon, 327, 351
- ImagePreviewer, 499
- ImageTest, 352
- InnerClassTest, 292
- InputEvent, 386
- InputMap, 376
- Integer, 242, 276, 308
- Iterator, 290
- JApplet, 533
- JButton, 361, 397
- JCheckBox, 415
- JCheckBoxMenuItem, 436
- JColorChooser, 505, 509
- JComboBox, 425, 725
- JComponent, 328, 347, 351, 379, 408, 419, 438
- JDialog, 484, 488
- JEditorPane, 412
- JFileChooser, 495, 503
- JFrame, 318, 321, 331, 434, 484
- klasa JLabel, 408, 499
- klasa JList, 425
- JMenu, 433
- JMenuBar, 432
- JMenuItem, 434, 440
- JOptionPane, 288, 474, 481, 484
- JPanel, 330
- JPasswordField, 406, 410
- JPopupMenu, 437
- JRadioButton, 418
- JRadioButtonMenuItem, 436
- JScrollPane, 413
- JSlider, 426, 431, 640
- JTextArea, 406, 410, 412
- JTextComponent, 406
- JTextField, 406, 408
- JToolBar, 445, 447
- KeyStroke, 375, 379
- LayoutManager, 472
- Line2D.Double, 340
- LineBorder, 422
- LineMetrics, 347, 350
- LinkedBlockingQueue<E>, 792
- LinkedHashMap, 702, 704
- LinkedHashMap<K, V>, 705
- LinkedHashSet, 702
- LinkedHashSet<E>, 705
- LinkedList, 290, 668, 675, 684, 694, 713
- LinkedListQueue, 668
- LinkedListTest, 681
- LinkedTransferQueue, 788
- ListIterator, 679
- Lock, 762
- Logger, 605
- LogManager, 595
- LogRecord, 609
- LookAndFeelInfo, 368
- Manager, 232
- ManagerTest, 205
- MapTest, 699
- Math, 73, 74
- MouseListener, 441
- Method, 252, 265, 663
- MethodTableTest, 266
- Modifier, 252, 256
- MouseEvent, 380, 386
- MouseHandler, 383
- MouseMotionHandler, 383
- MouseMotionListener, 381
- Object, 220
- ObjectAnalyzer, 258
- ObjectAnalyzerTest, 259
- PackageTest, 183, 184
- Pair, 303, 637, 641, 648, 655
- Pair<T>, 659
- PairTest1, 631
- PairTest2, 635
- PairTest3, 656
- ParallelGroup, 468
- ParamTest, 169
- PasswordChooser, 490
- Paths, 97
- PersistenceService, 532
- PersonTest, 217
- PlafFrame, 368
- Point2D, 335
- Point2D.Double, 340
- Preferences, 556, 560
- PreferencesFrame, 558
- PrintWriter, 97
- PriorityBlockingQueue<E>, 792
- PriorityQueue, 697
- PriorityQueueTest, 697
- Properties, 550–555, 709, 728
- PropertiesTest, 551
- Proxy, 307, 311
- ProxyTest, 309
- Rectangle2D, 334, 335
- Rectangle2D.Double, 334, 339
- Rectangle2D.Float, 334, 339
- RectangularShape, 335, 339
- RecursiveTask<T>, 810

- ReentrantLock, 762–764, 783
- ReentrantReadWriteLock, 783
- ReflectionTest, 253
- ResourceBundle, 596
- ResourceTest, 517
- Robot, 617, 618
- Runnable, 746
- RuntimeException, 566–568, 583
- Scanner, 89, 90, 97
- SequentialGroup, 468
- ServiceManager, 531
- SetTest, 686
- ShuffleTest, 721
- SimpleDateFormat, 781
- SimpleFrame, 319
- SimpleFrameTest, 318
- Singleton, 645
- SizedFrameTest, 324
- SoftBevelBorder, 421, 422
- SortedMap<K, V>, 701
- SQLException, 576
- Stack, 709, 729
- StackTraceElement, 581, 583
- StackTraceTest, 582
- StaticInnerClassTest, 305
- StaticTest, 162
- StreamHandler, 598
- StrictMath, 74
- String, 77, 83, 86, 211
- StringBuilder, 86, 88
- SwingThreadTest, 818
- SwingUtilities, 494
- SwingWorker, 820, 824, 826
- SwingWorkerTest, 821
- SynchronousQueue, 815
- System, 43, 554
- SystemColor, 341
- TalkingClock, 289, 295–297
- Thread, 647, 746, 749–753
- ThreadGroup, 754, 755
- ThreadLocal, 781
- ThreadLocal<T>, 782
- ThreadPoolTest, 804
- Throwable, 251, 565, 570, 581, 583
- TimePrinter, 291–295
- Timer, 286–288
- TimerTest, 287
- ToolBarTest, 447
- Toolkit, 288, 323, 327, 386
- TraceHandler, 307
- TransferRunnable, 780
- TreeMap<K, V>, 701
- TreeSet, 688, 691
- TreeSet<E>, 688, 694
- TreeSetTest, 691
- UIManager, 368
- Vector, 234, 684, 709, 726
- WeakHashMap, 702
- WeakHashMap<K, V>, 705
- Window, 322, 326
- WindowAdapter, 370
- WindowEvent, 372
- klasy
 - abstrakcyjne, 214, 216, 279
 - adaptacyjne, adapter class, 369
 - anonimowe, 300
 - bazowe, 200, 279
 - blokad, 783
 - finalne, 211
 - generyczne, 234
 - graniczne, 636
 - kolekcyjne, 627, 672, 674, 709, 726
 - kontenerowe, 709
 - macierzyste, 200
 - modelowe, 397
 - niezmiennie, 158
 - ogólne, generic class, 629
 - osłonowe, 241
 - pochodne, 200
 - podpisywane cyfrowo, 25
 - pomocnicze, 453, 781
 - potomne, 200
 - proxy, 306, 311
 - publiczne, 180
 - specjalne, 702
 - statyczne, 303
 - surowe, 237
 - szablonowe, 632
 - w architekturze kolekcji, 710
 - wewnętrzne, 271, 289
 - anonimowe, 300
 - bezpieczeństwo, 296
 - dostęp do stanu obiektu, 289
 - dostęp do zmiennych finalnych, 297
 - lokalne, 296
 - obsługa zdarzeń, 362
 - prawa dostępu, 296
 - referencja do klasy zewnętrznej, 293
 - referencja do obiektu zewnętrznego, 291
 - reguły składniowe, 293
 - składnia, 289
 - statyczne, 303
 - wyjątków, 570
 - wyliczeniowe, 245
 - zagnieżdżone, 290
 - zdarzeniowe AWT, 387
- klasyfikacja wyjątków, 565

- klauzula
 - catch, 572, 748
 - finally, 576, 578
 - throws, 96, 567
 - try, 572
- klawiatura, 375
- klawisze specjalne, 380
- klonowanie obiektów, 280, 285
- klucz URL, 528
- klucze, 698
- klucze należące do węzła, 560
- kod
 - bajtowy, 26
 - błędu, 570
 - kod generyczny, 630
 - kod maszynowy, 26
 - kod mieszający, hash code, 225, 684, 687
 - kod ogólny, 635
 - kod wyjścia, exit code, 60
- kodowanie
 - Unicode, 65, 67
 - UTF-16, 66, 82
- kolejka, queue, 666
 - ArrayBlockingQueue, 788
 - DelayQueue, 788
 - Deque, 694
 - LinkedBlockingQueue, 787
 - PriorityBlockingQueue, 788
 - Queue, 694
- kolejki
 - blokujące, 786
 - dostępu, 472
 - priorytetowe, 696
 - synchroniczne, 815
- kolejność
 - dostępu, 703
 - dostępu do komponentów, 473
 - ograniczeń, 637
- kolekcja, 117, 665
 - par, 698
 - wątków, 754
- kolekcje
 - bezpieczne wątkowo, 794, 796
 - ograniczone, 668
 - uporządkowane, 677, 708
 - w bibliotece, 675
- kolizja nazw, 180
- kolizje, 685
- kolor, 340
- kolor tła, 341, 507
- komentarze, 61
 - do klas, 191
 - do metod, 191
 - do pakietów, 194
 - do pól, 192
 - dokumentacyjne, 190
 - ogólne, 192
- komparator, 690
- kompilacja
 - programu, 44
 - w czasie rzeczywistym, 26
- kompilator, 25, 45, 776
 - czasu rzeczywistego, 34
 - javac, 188
 - JIT, 27, 212
- komponenty
 - Swing, 391–510
 - tekstowe, 411
- kompresja ZIP, 512
- komunikacja
 - między apletami, 540, 547
 - międzyprocesowa, 736
- komunikat, 592
 - o błędzie, 46, 50, 569
 - o wyjątkach, 816
- konektor UML, 136
- konfiguracja
 - komponentów, 319
 - menedżera dzienników, 598
 - projektu, 48
- konflikt metod, 648
- konkatenacja, 78
- konsola, 44
- konstruktor, 137, 152
 - bezargumentowy, 172
 - domyślny, 172
 - kopiujący, 140
 - przeciążony, 172
 - wirtualny, 250
- konstruktory
 - klasy FileHandler, 607
 - klasy HashSet<E>, 687
 - klasy TreeMap<K, V>, 701
 - klasy TreeSet<E>, 694
- kontekst
 - graficzny, 328
 - urządzenia, 328
- kontener, 330, 399
- kontrola
 - dostępu, 290
 - nazw, 290
 - typów, 627
- kontroler, controller, 394
- konwersja
 - łańcucha na liczbę, 242
 - między kolekcjami a tablicami, 718
 - programów na aplety, 536
 - tablic, 718
 - typów, 650
 - typów numerycznych, 74

kończenie działania programu, 60
 kopie

- łańcucha, 80
- obrazu, 352

 kopiowanie

- głębokie, 281
- obiektów, 280
- plytkie, 281
- tablicy, 119
- zmiennej tablicowej, 119

 koszty uzyskania certyfikatu, 524
 kowariantne typy zwrotne, 209, 639
 kubek, bucket, 685
 kursory, 382
 kwalifikator `.this`, 294

L

licencja GPL, 34
 liczba

- kliknięć, 381
- parametrów, 244

 liczby

- całkowite, 62
- zmiennoprzecinkowe, 64

 linia

- bazowa, 346, 347
- dolna pisma, 346
- górną pisma, 346

 lista

- ArrayList, 117
- kluczy, 556, 698
- modyfikowalna, 721
- rozwijalna, combo box, 423
- wątków, 779

 listy

- cykliczne, 666, 668
- dwukierunkowe, 25, 238, 674
- powiązane, 668, 674, 680
- tablicowe, 234, 236, 684
- surowe, 239
- z typem, 240

 lokalizacja, 143, 596

- komunikatów, 596
- pliku, 546

 lokalne klasy wewnętrzne, 289, 296

Ł

ładowanie

- klas, 307
- pliku w osobnym wątku, 821
- zasobów, 516

łańcuch

- `_blank`, 548
- dziedziczenia, 206
- null, 81
- prompt, 91
- pusty, 81
- testowy, 407
- wyjątków, 575

 łańcuchy, 77

- łączenie, 78
- modyfikowanie, 79
- porównywanie, 79
- składanie, 86
- współdzielenie, 79
- zmienialne, mutable, 80

 łączenie narastające, 27

M

makro assert, 588
 manifest, 512
 mapa, 697

- akcji, 376
- HashMap, 698
- haszowa, 795
- wejścia, 376
- własności, property map, 550, 553, 728

 mapy klawiaturowe, 376
 maska bitowa, 380
 maszyna wirtualna, JVM, 26, 514

- opcja `-verbose`, 612
- opcja `-Xlint`, 612
- opcja `-Xprof`, 614

 ME, Micro Edition, 38
 menedżer

- dzienników, 595
- zabezpieczeń, 522

 menu, 432
 menu podręczne, pop-up menu, 437
 metadane, 32
 metoda, 133

- accept, 504
- acquire, 812
- actionPerformed, 286, 291, 357, 368, 414, 433
- add, 142, 237, 399, 433, 629, 677
- addActionListener, 364
- addAll, 629, 672, 682
- addBall, 737, 742
- addChangeListener, 426
- addChoosableFileFilter, 504
- addComponent, 467
- addContainerGap, 468
- addFirst, 683
- addGap, 467

- metoda
 - addGroup, 467
 - addItem, 423, 426
 - addLast, 683
 - addPropertyChangeListener, 373
 - addSeparator, 434, 447
 - addSuppressed, 583
 - addWindowListener, 370
 - akcesora get, 141
 - and, 730
 - andNot, 730
 - append, 87, 413
 - appendCodePoint, 88
 - Arrays.hashCode, 227
 - Arrays.toString, 230
 - asList, 711
 - await, 766, 769, 783, 814
 - awaitUninterruptibly, 783
 - beep, 289
 - binarySearch, 123, 308, 722
 - BorderFactory, 421
 - brighter, 341
 - call, 801
 - cancel, 802
 - canRead, 532
 - canWrite, 532
 - cast, 658
 - ceiling, 694
 - charAt, 83
 - checkedCollection, 713
 - clear, 672, 730
 - clone, 280–284
 - close, 179, 580, 600, 607, 762
 - codePointAt, 83
 - codePointCount, 84
 - compare, 276, 690, 693
 - compareTo, 83, 246, 272–274, 299, 634, 653, 693, 720
 - Component.show, 320
 - config, 605
 - console, 91
 - contains, 672, 680, 686
 - containsAll, 672, 673
 - containsKey, 700
 - containsValue, 700
 - copy, 724
 - copyArea, 352, 354
 - copyOf, 123
 - countdown, 813
 - create, 365
 - createCompoundBorder, 422
 - createCustomCursor, 382, 386
 - createEmptyBorder, 421
 - createEtchedBorder, 421
 - createFont, 345
 - createLineBorder, 421
 - createLoweredBevelBorder, 421
 - createMatteBorder, 421
 - createParallelGroup, 467
 - createRaisedBevelBorder, 421
 - createScreenCapture, 618
 - createTitledBorder, 422
 - darker, 341
 - decrementAndGet, 777
 - metoda delay, 621
 - metoda delete, 88
 - deriveFont, 345, 350
 - destroy, 537
 - disjoint, 725
 - divide, 115
 - doInBackground, 824–826
 - draw, 333, 340
 - drawImage, 353
 - drawString, 329, 341, 347, 351
 - elements, 728
 - endsWith, 83
 - ensureCapacity, 236
 - entering, 600, 605
 - entrySet, 700
 - equals, 79, 83, 124, 221, 242, 648, 705
 - equalsIgnoreCase, 83
 - execute, 827
 - exiting, 600, 605
 - fill, 340, 343, 724
 - fillMenu, 725
 - finalize, 179
 - fine, 605
 - finer, 605
 - finest, 605
 - firstKey, 701
 - floor, 694
 - flush, 600, 607
 - Font.createFont, 345
 - format, 609
 - formatMessage, 609
 - formatTo, 92
 - forName, 248, 251
 - frame.setUndecorated, 320
 - frequency, 725
 - get, 142, 236, 261, 264, 628, 682
 - getActionCommand, 363, 388, 417
 - getActionMap, 379
 - getActualTypeArguments, 664
 - getAllItems, 726
 - getAncestorOfClass, 494
 - getApplet, 540
 - getAppletContext, 547, 549
 - getAppletInfo, 545

getApplets, 548, 549
getAudioClip, 547
getAutoCreateContainerGaps, 467
getAutoCreateGaps, 467
getAvailableFontFamilyNames, 344
getBackground, 343
getBounds, 664
getCause, 583
getCenterX, 339
getClass, 222, 248, 641
getClassName, 368, 584
getClickCount, 380, 386
getCodeBase, 528, 531
getColor, 343, 510
getColumns, 408
getComponentPopupMenu, 438
getConstructor, 658, 659
getConstructors, 252, 255
getContentPane, 327, 331
getDeclareFields, 261
getDeclaredConstructor, 658
getDeclaredConstructors, 252, 256
getDeclaredField, 261
getDeclaredFields, 252, 255, 258
getDeclaredMethods, 252, 255
getDeclaringClass, 256
getDefaultScreenDevice, 621
getDefaultToolkit, 288, 323, 327
getDelay, 788, 792
getDescent, 350
getDescription, 505
getDocumentBase, 546
getDouble, 258, 610
getEnumConstants, 658
getExceptionTypes, 256
getExtendedState, 326
getFamily, 349
getField, 261
getFields, 252, 255, 261
getFileName, 583
getFilter, 607
getFirst, 637, 652, 683
getFirstDayOfWeek, 143
getFont, 350, 408
getFontMetrics, 347, 351
getFontName, 349
getFontRenderContext, 346, 351
getForeground, 343
getFormatter, 607
getGenericComponentType, 664
getGenericInterfaces, 663
getGenericParameterTypes, 663
getGenericReturnType, 663
getGenericSuperclass, 663
getHandlers, 606
getHead, 609
getHeight, 347
getHonorsVisibility, 467
getIcon, 409, 505
getIconImage, 326
getImage, 327, 547
getInheritsPopupMenu, 438
getInputMap, 376, 379
getInputStream, 526, 531
getInstalledLookAndFeel, 368
getKey, 701
getKeyStroke, 375
getLast, 683
getLeading, 350
getLength, 262, 264
getLevel, 606, 607
getLineMetrics, 347, 350
getLineNumber, 584
getLocalGraphicsEnvironment, 344
getLogger, 605
getLoggerName, 608
getLowerBounds, 664
getMaxX, 339
getMessage, 571, 608
getMethodName, 584
getMethods, 255
getMethodsI, 252
getMillis, 608
getMinX, 339
getModifiers, 252, 256
getModifiersEx, 381, 386
getModifiersExText, 386
getMonths, 148
getName, 150, 249, 349, 505, 664
getNewState, 372
getOldState, 372
getOutputStream, 526, 532
getOwnerType, 664
getPaint, 343
getParameter, 541, 545
getParameterInfo, 546
getParameters, 608
getParameterTypes, 256
getParent, 606
getPassword, 410
getPoint, 386
getPredefinedCursor, 381
getPreferredSize, 330
getProperties, 550, 554
getProperty, 553, 728
getProxyClass, 311
getRawType, 664
getResource, 516

metoda

- getResourceBundle, 608
- getResourceBundleName, 608
- getReturnType, 256
- getRootPane, 494
- getSalary, 214, 265
- getScreenSize, 323, 327
- getSelectedFile, 504
- getSelectedItem, 423–426
- getSelectedObjects, 417
- getSelection, 417, 418
- getSequenceNumber, 609
- getServiceNames, 531
- getShortMonths, 148
- getShortWeekdays, 144, 148
- getSource, 388, 424
- getSourceClassName, 608
- getSourceMethodName, 608
- getStackTrace, 581, 583
- getState, 752
- getStringBounds, 346
- getSuperclass, 659
- getSuppressed, 581
- getTail, 609
- getText, 406, 409
- getThreadID, 609
- getThrown, 608
- getTitle, 322, 326
- getTotalBalance, 764
- getType, 252
- getTypeDescription, 505
- getTypeParameters, 663
- getUpperBounds, 664
- getUseParentHandlers, 607
- getValue, 373, 379, 701
- getWeekdays, 148
- getWidth, 334, 335, 339, 346
- getX, 339, 386
- getY, 386
- hashCode, 225, 684, 706
- hasMoreElements, 670, 727
- hasNext, 91, 669, 674, 677
- hasNextDouble, 91
- hasNextInt, 91
- headMap, 712, 716
- headSet, 712, 716
- higher, 694
- IconImage, 522
- in.close, 580
- incrementAndGet, 777
- indexOf, 84
- indexOfSubList, 724
- info, 605
- init, 536
- initCause, 583
- initialize, 782
- initialValue, 781
- insert, 88, 434
- insertItemAt, 426
- InsertItemAt, 424
- insertSeparator, 434
- interrupt, 746, 749
- interrupted, 748, 749
- intValue, 243
- invoke, 265–267, 311
- invokeAll, 808, 810
- invokeAndWait, 820
- invokeAny, 808
- invokeLater, 817, 820
- isAbstract, 256
- isAccessible, 261
- isDispatchThread, 820
- isDone, 777
- isEditable, 406, 425
- isEmpty, 672, 673
- isEnabled, 373, 379
- isFinal, 252, 256
- isInterface, 256
- isInterrupted, 746–749
- isJavaIdentifierPart, 67
- isJavaIdentifierStart, 67
- isLocationByPlatform, 323, 326
- isLoggable, 600, 609
- isNative, 256
- isNativeMethod, 584
- isPopupTrigger, 438
- isPrivate, 252, 256
- isProtected, 256
- isProxyClass, 311, 312
- isPublic, 252, 256
- isResizable, 326
- isSelected, 415, 436
- isStatic, 256
- isStrict, 257
- isSynchronized, 257
- isTraversable, 498, 505
- isUndecorated, 326
- isVisible, 325
- isVolatile, 257
- isWebBrowserSupported, 531
- itemComparator, 720
- iterator, 668, 672
- join, 752, 810
- JTextField, 406
- keyPress, 621
- keyRelease, 621
- keySet, 700
- KeyStroke, 379

lastIndexOf, 84
lastIndexOfSubList, 724
lastKey, 701
layoutContainer, 472
length, 84, 730
linkSize, 466
listFiles, 497
listIterator, 677, 682
load, 554, 728
lock, 762, 764, 782
lockInterruptibly, 783
log, 593, 606
logp, 606
logrb, 606
lookup, 531
lower, 694
main, 59, 148, 162, 249
makeButton, 362
makePair, 644
Math.random, 122
Math.round, 75
menuCanceled, 441
menuDeselected, 441
menuSelected, 441
minimumLayoutSize, 472
mod, 115
modifiers, 256
mouseClicked, 380, 381
mouseDragged, 382
mouseEntered, 383
mouseExited, 383
mouseMove, 621
mouseMoved, 381, 382
mousePress, 621
mousePressed, 380, 381
mouseRelease, 621
mouseReleased, 380
move, 736
multiply, 115, 116
mutatora set, 142
newCondition, 765, 769
newFixedThreadPool, 803
newInstance, 249–251, 262, 658
newProxyInstance, 307, 311
newScheduledThreadPool, 807
newSingleThreadScheduledExecutor, 807
next, 669, 679
nextDouble, 89, 91, 610
nextElement, 670, 727
nextInt, 89, 91
nextLine, 89, 91
node, 560
notify, 773
notifyAll, 773
Object.clone, 282
Objects.hashCode, 227
offer, 787, 793
offerFirst, 793
offerLast, 793
offsetByCodePoints, 83
openFileDialog, 526, 532
openMultiFileDialog, 532
or, 730
ordinal, 247
pack, 330, 332
paintComponent, 328, 347, 475, 569
parse, 244
parseInt, 242, 243
peek, 729, 787
play, 546
poll, 787, 793, 809
pollFirst, 694, 793
pollLast, 694, 793
pop, 729
preferredLayoutSize, 472
previous, 677–680
previousIndex, 680
print, 96
printBuddies, 651
printf, 92, 244
println, 60, 96, 229, 310
printStack, 251
printStackTrace, 251, 581, 611
process, 826
publish, 599, 607, 825
push, 729
put, 556, 787, 792
putFirst, 793
putIfAbsent, 794
putLast, 793
putValue, 373, 379
raiseSalary, 284
readConfiguration, 595
readLine, 91
readLock, 784
readPassword, 91
release, 812
remove, 434, 669–673, 678, 682
removeAll, 672, 673, 681
removeAllItems, 426
removeEldestEntry, 705
removeFirst, 683
removeHandler, 607
removeItem, 424, 426
removeItemAt, 424, 426
removeLast, 683
removeLayoutComponent, 472
removePropertyChangeListener, 373

- metoda
 - repaint, 329, 332, 827
 - replace, 84, 796
 - replaceAll, 724
 - res.close, 580
 - resetChoosableFileFilters, 504
 - resize, 537
 - resume, 752
 - retainAll, 672, 717
 - revalidate, 407, 408
 - reverse, 724
 - reverseOrder, 722
 - rotate, 724
 - run, 647, 754
 - Runtime.addShutdownHook, 179
 - saveAsFileDialog, 532
 - saveFileDialog, 526
 - schedule, 807
 - scheduleAtFixedRate, 807
 - scheduleWithFixedDelay, 808
 - ServiceManager, 526
 - set, 142, 236, 261, 679, 682
 - setAccelerator, 440
 - setAccessible, 257, 261
 - setAccessory, 504
 - setAction, 434
 - setActionCommand, 363, 417, 419
 - setAutoCreateContainerGaps, 467
 - setAutoCreateGaps, 467
 - setBackground, 341, 343
 - setBorder, 419, 423
 - setBounds, 321, 322, 325, 468
 - setCharAt, 88
 - setColor, 343, 510, 625
 - setColumns, 408, 410, 413
 - setComponentPopupMenu, 438
 - setCursor, 386
 - setDaemon, 754
 - setDebugGraphicsOptions, 615
 - setDefaultButton, 494
 - setDefaultCloseOperation, 320, 536
 - setDefaultUncaughtExceptionHandler, 611
 - setDone, 777
 - setEditable, 406, 423, 425
 - setEnabled, 373, 379, 441
 - setExtendedState, 325, 326
 - setFileFilter, 497, 504
 - setFileSelectionMode, 503
 - setFileView, 499, 504
 - setFilter, 600, 607
 - setFirst, 652
 - setFont, 350, 408
 - setForeground, 341, 343
 - setFormatter, 601, 607
 - setFrameFromCenter, 337
 - setFrameFromDiagonal, 336
 - setHonorsVisibility, 467
 - setHorizontalGroup, 466
 - setHorizontalTextPosition, 435
 - setIcon, 409, 435
 - setIconImage, 321, 326
 - setInheritsPopupMenu, 438
 - setJMenuBar, 432, 434
 - setLabelTable, 428, 431, 640
 - setLayout, 399
 - setLevel, 606, 607
 - setLineWrap, 411, 413
 - setLocation, 321, 325
 - setLocationByPlatform, 322, 326
 - setLookAndFeel, 368
 - setMajorTickSpacing, 431
 - setMinorTickSpacing, 431
 - setMnemonic, 440
 - setModel, 424
 - setMultiSelectionEnabled, 503
 - setPaint, 340, 341, 343
 - setPaintLabels, 428, 431
 - setPaintTicks, 428, 431
 - setPaintTrack, 432
 - setParent, 606
 - setPriority, 753
 - setRect, 335
 - setResizable, 321, 326
 - setRows, 410, 413
 - setSecond, 638
 - setSelected, 414, 436
 - setSelectedFiles, 503
 - setSize, 325
 - setSnapToTicks, 432
 - setSource, 363
 - setText, 406, 407, 409
 - setTitle, 321, 326, 537
 - setToolTip, 446
 - setToolTipText, 447
 - setUncaughtExceptionHandler, 754
 - setUndecorated, 326
 - setUseParentHandlers, 607
 - setValue, 701
 - setVerticalGroup, 466
 - setVisible, 320, 325, 489, 537, 827
 - setWrapStyleWord, 413
 - severe, 605
 - show, 320, 437
 - showConfirmDialog, 474, 476, 482
 - showDialog, 490
 - showDocument, 531, 548, 549
 - showInputDialog, 475, 476, 483
 - showInternalConfirmDialog, 482

- showInternalInputDialog, 484
 - showInternalMessageDialog, 482
 - showMessageDialog, 288, 474, 481, 530
 - showOptionDialog, 474, 476
 - showSaveDialog, 495
 - showStatus, 548, 549
 - shuffle, 721, 722
 - shutdown, 803
 - shutdownNow, 804
 - signal, 769, 780
 - signalAll, 766–769
 - size, 236, 672, 673
 - sleep, 737, 742
 - sort, 121, 274, 720
 - start, 288
 - startsWith, 84
 - stateChanged, 426
 - stop, 288, 752, 784
 - store, 550, 554, 728
 - subList, 716
 - subMap, 712, 716
 - submit, 803, 809
 - subSet, 712, 716
 - substring, 78, 84, 711
 - subtract, 115, 116
 - super.clone, 282
 - super.paintComponent, 330
 - suspend, 752, 785
 - swap, 168, 724
 - swapHelper, 656
 - SwingUtilities.updateComponentTreeUI, 366
 - synchronizedCollection, 713, 797
 - synchronizedList, 797
 - synchronizedMap, 713, 797
 - synchronizedSet, 797
 - synchronizedSortedMap, 797
 - synchronizedSortedSet, 797
 - System.exit, 60, 320
 - System.out.println, 89
 - System.runFinalizersOnExit, 179
 - systemNodeForPackage, 560
 - systemRoot, 560
 - tailMap, 712, 716
 - tailSet, 712, 716
 - take, 793
 - takeFirst, 793
 - takeLast, 793
 - text, 91
 - Thread.getAllStackTraces, 581
 - ThreadLocalRandom.current, 781
 - throwing, 594, 606
 - toArray, 237, 645, 672, 718
 - toBack, 326
 - toFront, 322, 326
 - toString, 88, 118, 228, 259, 310, 424, 584, 610
 - toUpperCase, 84
 - transfer, 756, 761, 770, 793
 - trim, 85, 407
 - trimToSize, 235, 236
 - tryLock, 782, 783
 - tryTransfer, 793
 - UIManager.setLookAndFeel, 366
 - uncaughtException, 754, 755
 - unlock, 762, 764
 - unmodifiableCollection, 713
 - unmodifiableList, 713
 - unmodifiableSet, 713
 - update, 240
 - userNodeForPackage, 560
 - userRoot, 560
 - validate, 407, 408
 - valueOf, 114, 243, 246
 - wait, 773
 - warning, 605
 - windowActivated, 369, 372
 - windowClosed, 369, 372
 - windowClosing, 369, 370, 372
 - windowDeactivated, 369, 372
 - windowDeiconified, 369, 372
 - windowIconified, 369, 372
 - windowOpened, 369, 372
 - windowStateChanged, 372
 - writeLock, 784
 - xor, 730
- metody
- abstrakcyjne, 215
 - akcesora, 155
 - fabryczne, 161, 704, 803
 - finalne, 211, 770
 - graficzne, 332
 - interfejsu
 - Action, 373
 - BlockingDeque<E>, 793
 - BlockingQueue<E>, 792
 - Collection, 672
 - Collection<E>, 672
 - Deque<E>, 695
 - Future<V>, 801
 - GenericArrayType, 664
 - Iterator<E>, 674
 - LinkedList<E>, 683
 - List<E>, 682
 - ListIterator<E>, 683
 - Map<K, V>, 700
 - NavigableSet<E>, 694
 - ParameterizedType, 664
 - Queue<E>, 695
 - SortedSet, 712

metody

interfejsu

- TypeVariable, 664
- WildcardType, 664
- WindowListener, 369, 372

klas wewnętrznych, 289

klasy

- AccessibleObject, 261
- Applet, 537, 545, 546
- Array, 264
- Arrays, 123
- BigDecimal, 116
- BigInteger, 115
- BitSet, 730
- BorderFactory, 421
- Class, 252, 255
- Class<T>, 658
- Collections, 712, 715, 722, 724
- Component, 325
- Console, 91
- Constructor, 256
- Date, 141
- Employee, 151
- Executors, 803
- FileView, 498, 505
- Font, 349
- Frame, 326
- Graphics, 350
- Graphics2D, 351
- GregorianCalendar, 147
- GroupLayout, 466
- Integer, 243
- JComboBox, 425
- JFileChooser, 503
- JFrame, 321
- JMenu, 433
- JOptionPane, 481
- JSlider, 431
- JTextArea, 412
- JTextComponent, 406
- LayoutManager, 472
- LineMetrics, 350
- Logger, 605
- Modifier, 256
- Object, 773
- Preferences, 560
- Properties, 553
- RectangularShape, 335, 339
- Robot, 621
- Scanner, 90
- String, 83, 87
- StringBuilder, 88
- Thread, 749–755

ThreadLocal<T>, 782

Throwable, 583

Timer, 288

Toolkit, 327

Window, 326

kolejek blokujących, 787, 788

komentarze, 191

monitorowe, 775

o zmiennej liczbie parametrów, 244

odradzane, 141

ogólne, 632

pomostowe, 638, 649

prywatne, 157

przeciążanie, 171

przesłaniające, 201

publiczne, 59

rejestrujące, 594

rodzime, 160

statyczne, 73, 160, 645

statyczne ze zmiennymi typowymi, 645

sygnatura, 171, 209

synchronizowane, 771

tworzące niemodyfikowalne widoki, 712

udostępniające, 141

uogólnione, 671

wstawiane, 154

z parametrami typowymi, 632

zmieniające wartość elementu, 141

mieszanie współrzędnych, 691

mnemoniki, 438

modalność, 485

model, 394, 395

pola tekstowego, 394

wskaźnikowy, 24

moduł ładujący klasy, 588

modyfikacja

parametru obiektowego, 167

zbioru EnumSet, 704

modyfikator

dostępu, 58, 220

dostępu private, 186

dostępu public, 185

final, 158, 211, 777

volatile, 777

zdarzenia, 386

modyfikowanie elementów zbioru, 687

monitor, 775

motyw Ocean, 316

mutatory, 142

MVC, Model-View-Controller, 392

mysz, 380

N

nadklasa, superclass, 200
 nadtypy, 652, 654
 NaN, 64, 70
 narzędzia
 graficzne, 317
 wiersza poleceń, 44
 narzędzie
 appletviewer, 53, 535
 ButtonTest, 618
 ImageViewer, 51, 500
 jar, 512, 514
 Jar Bundler, 515
 javac, 45
 javadoc, 190–195
 javap, 294
 jconsole, 595, 613, 779
 jmap, 614
 Matisse, 460, 461
 OptionDialogTest, 477
 ReflectionTest, 295, 296
 Swing graphics debugger, 614
 natywna biblioteka interfejsowa, 33
 nawiasy
 klamrowe, 59, 300
 kwadratowe, 116
 ostre, 632
 puste, 234
 nazwa
 akcji, 377
 klasy, 45, 58, 134, 197
 konstruktora, 137
 parametru, 174
 pliku, 45
 pliku dziennika, 598
 rejestratora, 595
 zasobu, 516
 zmiennej, 67
 zmiennej typowej, 630
 nazwy
 klas komponentów Swing, 318
 klas proxy, 311
 logiczne czcionek, 344
 metod, 197
 rodziny czcionek, 343
 NetBeans, 38, 44, 459
 niezależność od architektury, 26
 niezawodność, 24
 niezmiennalność łańcuchów, 79
 niszczenie obiektów, 179
 notacja wielbłądzia, 58

O

obiekt, 24, 132
 Action, 446
 ActionEvent, 357
 AssertionError, 587
 BasicButtonUI, 398
 builder, 86
 ButtonGroup, 415
 Callable, 803
 ColorAction, 360
 Comparator, 693
 Console, 90
 Date, 138
 DefaultButtonModel, 398
 ExecutorCompletionService, 808
 FutureTask, 798
 Graphics, 328, 341
 GridLayout, 406
 Handler, 595, 598, 608
 Iterator, 672
 JButton, 397
 JPanel, 402
 JRadioButton, 415
 PaintEvent, 388
 Path, 97
 PrintWriter, 96, 97
 Properties, 553
 Runnable, 759, 803
 Scanner, 96
 System.out, 60
 WeakReference, 702
 obiektów, 133
 autoboxing, 241
 hermetyzacja, 133
 klonowanie, 280
 kopiowanie, 280
 metody, 133
 metody prywatne, 157
 niszczenie, 179
 polimorfizm, 204
 porównywanie, 221
 składowe, 133, 155
 stan, 133, 134
 tożsamość, 134
 właściwości, 133
 zachowanie, 133
 obiekty
 blokady, 762
 funkcyjne, 690
 klasy Class, 249
 klasy Lock, 762
 klasy ogólnej, 646
 nasłuchujące zdarzeń, listener objects, 287, 371

- obiekty
 - obsługujące wywołanie, 307
 - opakowujące kolekcje, 711
 - proxy, 308
 - typu wyliczeniowego, 548
 - warunków, 765
 - zdarzeń, event objects, 356
- obliczanie kodu mieszającego, 226
- obramowanie, 419
- obrazy, 351
- obsługa
 - apletów, 533
 - błędów, 564
 - kliknięcia przycisku, 357
 - nieprzechwyconych wyjątków, 754
 - przycisku OK, 486
 - ramek, 325
 - wyjątków, 249, 564, 646
 - zdarzeń, 329, 355
 - zdarzeń AWT, 389, 390
 - zdarzeń myszy, 380, 383
- obszar
 - surogatów, surrogates area, 66
 - tekstowy, text area, 406, 410
- ochrona bloku kodu, 762
- odczyt plików, 96
- odmierzenie czasu, 782
- odpakowywanie, 242
- odrzucone metody, 141
- odwołanie, 193
- odwzorowanie nazw czcionek, 345
- ograniczenia
 - blokad wewnętrznych, 771
 - nadtypów, 652, 653
 - typów generycznych, 240
 - widoczności metody, 219
 - widoków, 714
 - zmiennych typowych, 633
- okna dialogowe
 - modalne, 474
 - niemodalne, 474
- okno, 369, 372
 - dialogowe
 - opcji, 474, 483
 - potwierdzenia, 482
 - przyjmowania danych, 484
 - typu O programie, 485
 - wyboru kolorów, 505
 - wyboru plików, 495
 - z komunikatem, 482
 - z polem hasła, 489
 - dokumentacji API, 85
 - konsoli, 44
 - przeglądarki apletów, 53
 - wyboru plików, 498, 558
- OOP, Object Oriented Programming, 132
- opakowywanie, autoboxing, 241
- opakowywanie wyjątków, 648
- opcja
 - Step Into, 623
 - Step Over, 623
 - Terminate, 625
- opcje
 - debugera, 623
 - narzędzia jar, 513
- operacje
 - niepodzielne, 760
 - opcjonalne, 714
 - zbiorcze, 717
- operator
 - ::, 202
 - [], 120
 - ==, 705
 - dekrementacji, 71
 - inkrementacji, 71
 - instanceof, 213, 222, 278
 - new, 86, 137, 153, 276
 - przecinka, 77
- operatory
 - arytmetyczne, 69
 - bitowe, 72
 - logiczne, 71
 - relacyjne, 71
- opis
 - danych, 33
 - elementu, 692
 - klasy, 86, 192
 - metod, 87
 - struktury stron, 33
 - zmiennej, 192
- optymalizacja, 27
- osłona obiektów, 241
- ostrzeżenie, 643
- otwieranie menu, 438
- overloading resolution, 171
- oznaczenia relacji, 136

P

- pakiet
 - com.horstmann.corejava, 183
 - com.mycompany.mylib, 588
 - com.mycompany.util, 518
 - com.sun.java, 366
 - domyślny, 183
 - java.awt, 186
 - java.awt.event, 286, 388
 - java.lang, 83, 90
 - java.lang.reflect, 252, 660
 - java.math, 114

- java.sql, 181
- java.util, 90, 181, 387
- java.util.concurrent, 762, 771, 797
- java.util.concurrent.atomic, 777
- java.util.concurrent.locks, 783
- javax.swing, 286, 319
- javax.swing.event, 390
- JDK, 38
- org.omg.CORBA, 243
- Swing, 314
- pakiety, packages, 180
 - dodawanie klasy, 182
 - komentarze, 194
 - lokalizacyjne, 596
 - zasięg, 185
- panel
 - JPanel, 398
 - przewijany, scroll pane, 411, 413
 - z przyciskami, 359, 398
- para klucz – wartość, 556, 561, 697
- parametr, 61
 - anchor, 452
 - fill, 452
 - gridheight, 451, 452
 - gridwidth, 451, 452
 - gridx, 451, 452
 - gridy, 451, 452
- parametry
 - jawne, 153
 - konfiguracyjne obiektu Handler, 598
 - łańcuchowe, 96
 - metod, 164
 - niejawne, 153, 174
 - obiektowe, 167
 - określające cel, 549
 - typowe, 628, 641
 - wiersza poleceń, 120
- pasek
 - menu, 432
 - narzędzi, toolbar, 444
- pętla
 - do-while, 102
 - for, 98, 106–108
 - for each, 32, 98, 117, 126
 - w stylu for each, 669
 - while, 101, 102
- piaskownica, sandbox, 519, 522
- pieczętowanie pakietów, 186, 518
- plik
 - AboutDialog.java, 487
 - ActionFrame.java, 377
 - AnonymousInnerClassTest.java, 301
 - ArrayListTest.java, 238
 - Ball.java, 739
 - BallComponent.java, 740
 - Bank.class, 761
 - Bank.java, 758, 767, 772
 - BigIntegerTest.java, 115
 - BlockingQueueTest.java, 789
 - BorderFrame.java, 420
 - Bounce.java, 737
 - BounceThread.java, 743
 - BuggyButtonTest.java, 622
 - ButtonFrame.java, 360
 - ButtonPanel.java, 481
 - Calculator.jnlp, 519
 - CalculatorFrame.java, 528
 - CalculatorPanel.java, 403
 - CalendarTest.java, 144
 - Chart.java, 543
 - CheckBoxTest.java, 414
 - CircleLayout.java, 469
 - CircleLayoutFrame.java, 472
 - CloneTest.java, 284
 - ColorChooserPanel.java, 508
 - ComboBoxFrame.java, 424
 - CompoundInterest.java, 125
 - ConstructorTest.java, 177
 - CopyOfTest.java, 263
 - DataExchangeFrame.java, 491
 - deskryptora, 519
 - DialogFrame.java, 486
 - doc-files, 191
 - DrawTest.java, 337
 - Employee.java, 151, 184, 205, 231, 275, 284
 - EmployeeSortTest.java, 275
 - EmployeeTest.java, 149, 151
 - en.properties, 596
 - EnumTest.java, 246
 - EqualsTest.java, 230
 - EventTracer.java, 615
 - FileIconView.java, 503
 - fontconfig.properties, 345
 - FontFrame.java, 454, 463
 - FontTest.java, 347
 - forkJoinTest.java, 810
 - FutureTest.java, 799
 - GBC.java, 456
 - GenericReflectionTest.java, 661
 - ImagePreviewer.java, 502
 - ImageTest.java, 352
 - ImageViewer.java, 51
 - ImageViewerFrame.java, 500
 - InnerClassTest.java, 292
 - InputTest.java, 89
 - Item.java, 692
 - java.nlog, 597
 - javaws.jar, 526

plik

- jogging.properties, 594
- LinkedListTest.java, 681
- LoggingImageViewer.java, 602
- LotteryArray.java, 128
- LotteryDrawing.java, 121
- LotteryOdds.java, 108
- Manager.java, 206, 232
- ManagerTest.java, 204
- MANIFEST.MF, 512
- manifestu
 - klasa główna, 514
 - sekcja główna, 512
 - wstawianie sekcji, 518
 - zmienianie zawartości, 513
- MapTest.java, 699
- MenuFrame.java, 442
- MethodTableTest.java, 266
- MouseComponent.java, 383
- MouseFrame.java, 383
- NotHelloWorld.java, 330, 534
- ObjectAnalyzerTest.java, 259
- OptionDialogFrame.java, 477
- overview.html, 194
- PackageTest.java, 184
- PairTest1.java, 631
- PairTest2.java, 634
- PairTest3.java, 656
- ParamTest.java, 169
- PasswordChooser.java, 492
- Person.java, 217
- PersonTest.java, 217
- PlafFrame.java, 367
- PreferencesTest.java, 557
- PriorityQueueTest.java, 696
- program.properties, 551
- PropertiesTest.java, 551
- ProxyTest.java, 309
- RadioButtonFrame.java, 417
- ReflectionTest.java, 253
- ResourceTest.java, 517
- Retirement.java, 103
- Retirement2.java, 105
- RobotTest.java, 618
- rt.jar, 187
- SetTest.java, 686
- ShuffleTest.java, 721
- Sieve.cpp, 731
- Sieve.java, 731
- SimpleFrameTest.java, 318
- SizedFrameTest.java, 324
- SliderFrame.java, 428
- src.zip, 41, 42
- StackTraceTest.java, 582
- StaticInnerClassTest.java, 305
- StaticTest.java, 162
- Student.java, 218
- swing.properties, 366
- SwingThreadTest.java, 817
- SwingWorkerTest.java, 821
- System.java, 43
- TalkingClock\$TimePrinter.class, 294
- TextComponentFrame.java, 411
- ThreadPoolTest.java, 804
- TimerTest.java, 287
- ToolBarTest.java, 446
- TransferRunnable.java, 759
- TreeSetTest.java, 691
- UnsynchBankTest.java, 757
- Welcome.java, 45
- WelcomeApplet.class, 53
- WelcomeApplet.html, 53
- WelcomeApplet.java, 55

pliki

- .class, 514
 - .exe, 514
 - .java, 58
 - .jnlp, 519
 - audio, 546
 - cookie, 527
 - graficzne, 546
 - JAR, 187, 512, 514
 - obrazów, 515
 - podpisane cyfrowo, 523
 - tekstowe, 515
 - XML, 600
 - z danymi binarnymi, 515
 - zasobów, 516
 - źródłowe, 151, 189
- pobieranie
- hasła, 90
 - właściwości systemowych, 554
- podklasa, subclass, 200
- Properties, 726
 - Stack, 726
 - podklasa, subclass, 200
- podkradanie pracy, 812
- podłańcuchy, 78
- podmenu, 432
- podpakiety, 180
- podpis cyfrowy, 26, 523
- podpisywanie kodu, 523
- podtyp typu granicznego, 634
- podziałka, 427
- pola
- finalne, 211
 - hasel, 406, 410
 - klasowe, 159

- niestatyczne, 159
- prywatne, 155
- publiczne, 155
- statyczne, 159, 176
- tekstowe, 394, 406
- ulotne, 776
- weight, 451
- wyboru, 413
- pole value, 243
- polecenie
 - cmd, 41
 - dir, 46
 - jcontrol, 536
- polimorfizm, 204, 207, 269
- połączenie na poziomie gniazd, 24
- położenie przycisków, 402
- porównywanie
 - elementów tablic, 225
 - łańcuchów, 79, 81
 - obiektów, 221, 689
 - obiektów osłonowych, 242
 - w pętli, 107
 - w podklasach, 277
- powiadamanie o zdarzeniach, 358
- powiązana tablica mieszająca, 703
- poziom
 - FINE, 595, 597, 601
 - rejestracji obiektu Handler, 597
- poziomy
 - rejestracji, 595
 - ważności komunikatów, 592
- pozycjonowanie
 - bezwzględne, 468
 - ramki, 321
- preferencje
 - użytkownika, 549, 555
 - węzła, 561
- priorytet
 - informacji, 592
 - wątków, 750
 - wątku, 752, 753
 - operatorów, 76
- procedura
 - obsługi błędów, 565
 - obsługi zdarzeń, 355
- proces, 736
- proces inliningu, 212
- program, *Patrz* narzędzie
- programowanie
 - interfejsów graficznych, 391
 - interfejsu użytkownika, 317
 - obiektywne, 24, 132
 - ogólne, generic programming, 627
 - pasków narzędzi, 445
 - proceduralne, 133
 - sieciowe, 33
 - wielowątkowe, 28
- programy
 - jednowątkowe, 736
 - refleksyjne, 247
 - wielowątkowe, 735
- projektant formy, 398
- projektowanie klas, 195
- prostokąt, 334
- protokoły sieciowe, 24
- prywatne pola, 155
- przechwytywanie
 - strumienia błędów, 611
 - wielu typów wyjątków, 574
 - wyjątków, 250, 571, 747
- przeciąganie paska narzędzi, 445
- przeciążanie
 - konstruktorów, 172
 - metod, 171
- przedział, 712
- przeglądarka
 - apletów, 535
 - HotJava, 31
 - pamięci podręcznej, 521
- przejmowanie blokady, 774
- przekazywanie
 - obiektu, 138
 - przez wartość, 167
 - wyjątków, 586
- przełącznik, radio button, 413
- przełączniki, 415
- przełączniki w elementach menu, 436
- przenośność, 26, 70
- przepływ sterowania, 98, 111, 298
- przerywanie
 - działania pętli, 111
 - procesu ładowania, 737
 - wątków, 746
- przesłanianie metod, 202, 225, 647
- przestrzenie numeracyjne, 66
- przesuwanie iteratora, 671
- przeszukiwanie liniowe, 723
- przetwarzanie XML, 33
- przycisk, 358, 397
 - domyślny, 490
 - JButton, 398
 - OK, 486
 - położenie, 402
 - rozmiar, 402
 - w rozkładzie brzegowym, 401
- przyciski radiowe, 415
- przywileje klasowe, 157

- publiczne metody
 - akcesora, 155
 - mutatora, 155
 - publiczne pola, 155
 - pule wątków, 802–804
 - punkt wstrzymania, 623, 624
 - pusta mapa własności, 553
- R**
- ramka, frame, 318
 - nadrzędna, 485, 490
 - wyświetlająca tekst, 327
 - ramki
 - pozycjonowanie, 321
 - rozmiar, 323
 - warstwy, 327
 - własności, 322
 - wyświetlanie, 320
 - reaktywacja wątków, 766
 - referencja
 - do elementów typu Object, 628
 - do klasy zewnętrznej, 291, 293
 - do obiektu, 138
 - do parametru niejawnego, 203
 - null, 250, 565
 - refleksja, reflection, 199, 247, 270, 658
 - analiza
 - funkcjonalności klasy, 252
 - obiektów w czasie działania programu, 257
 - generyczny kod tablicowy, 261
 - rejestr, 556
 - rejestr zdarzeń, 615
 - rejestratory, logger, 591, 597
 - rejestrujący obiekt pośredni, 610
 - rekordy dziennika, 597
 - relacje
 - agregacja, 135
 - dziedziczenie, 136
 - zależność, 135
 - reorganizacja tablicy mieszającej, 686
 - repozytorium Preferences, 555
 - robot, 618
 - rodzaje
 - ataków, 25
 - modalności, 485
 - obramowań, 419
 - suwaków, 429
 - rozkład
 - brzegowy, 400, 402, 448
 - ciągły, 448
 - GridBagLayout, 448–453
 - grupowy, 459, 461
 - komponentów, 407
 - siatkowy, 402, 448
 - sprężynowy, 449
 - SpringLayout, 449
 - rozmiar
 - apletu, 537
 - ekranu, 323
 - ikon, 522
 - interpretera, 23
 - pola tekstowego, 407
 - przycisków, 402
 - ramki, 323
 - tablicy, 117
 - rozmieszczanie komponentów, 398
 - rozstrzyganie przeciążania, 171, 209
 - rozszerzanie
 - klasy, 133, 216
 - klasy Throwable, 646
 - programów, 211
 - stylu, 317
 - rysowanie, 314
 - figur, 333, 337
 - na komponencie, 328
 - obrazu, 354
 - wykresu, 543
 - rzutowanie, casting, 75, 212, 637, 644, 718
- S**
- sandbox, 519, 522
 - scalanie list, 681
 - SDK, Software Development Kit, 38
 - SE, Standard Edition, 38
 - semafory, 812
 - separator, 445
 - serializacja, 539
 - serwer pochodzenia, 523
 - serwer Tomcat, 519, 520
 - siatka, 450
 - sito Eratostenesa, 730
 - skaner, 89
 - skład tekstów, 346
 - składanie łańcuchów, 86
 - składnia
 - diamantowa, diamond syntax, 234
 - Javy, 23
 - klas wewnętrznych, 289
 - wewnętrznych klas anonimowych, 371
 - składowe, 133, 155
 - składowe chronione, 220
 - skrótów klawiszowe, 439
 - słabe referencje, 702
 - słowo kluczowe, 829
 - abstract, 215
 - assert, 587
 - catch, 250
 - class, 58

- extends, 200, 634
- final, 68, 158, 211, 299
- implements, 273, 634
- import, 180
- instanceof, 213
- interface, 272
- package, 183, 186
- private, 152, 158, 220
- protected, 190, 219
- public, 58, 152, 220
- static, 69, 160, 304
- strictfp, 70
- super, 202
- synchronized, 762, 769, 775
- this, 154, 160, 174
- throws, 569
- try, 250
- void, 60
- volatile, 776
- sluchacz
 - akcji, 414, 416
 - przycisku, 359
 - z źródłami zdarzeń, 373
 - zdarzeń, event listener, 356, 364, 388
- sortowanie, 275, 299, 720
 - kluczy, 701
 - listy elementów, 720
 - tablicy, 121
- specyfikacja, 59
- specyfikacja wyjątku, 568
- specyfikator
 - dostępu, 150
 - formatu, 92, 96
 - throws, 283, 569, 573
- sprawdzanie
 - parametrów, 589
 - pól obiektu, 265
 - typów, 641
 - typów pól, 257
 - zakresu, 120
- sprzężenie zwrotne, 286
- stała, 68
 - ACCELERATOR_KEY, 374
 - ACTION_COMMAND_KEY, 374
 - BorderLayout.SOUTH, 401
 - DEFAULT, 374
 - Double.NaN, 64
 - Double.NEGATIVE_INFINITY, 64
 - Double.POSITIVE_INFINITY, 64
 - LONG_DESCRIPTION, 374
 - MNEMONIC_KEY, 374
 - NAME, 374
 - SHORT_DESCRIPTION, 374
 - SMALL_ICON, 374
- stałe
 - interfejsu Action, 374
 - interfejsu SwingConstants, 409
 - klasowe, 69
 - klasy BorderLayout, 401
 - łańcuchowe, 80
 - matematyczne, 73
 - statyczne, 159
- stan
 - obiektu, 133, 134
 - okna, 372
 - wątków, 749, 751
- standard
 - ECMA-262, 35
 - IEEE 754, 64
 - ISO/ANSI, 80
 - wyrażania czasu, 139
- status przerwania wątku, 746
- statyczne
 - funkcje składowe, 60
 - klasy wewnętrzne, 303
- sterta, heap, 120, 140, 696
- STL, Standard Template Library, 666
- stopień powiązań między klasami, 135
- stopy oprocentowania, 126
- stos, stack, 120, 729
- stos wywołań, 251, 582
- stosowanie
 - blokady, 765
 - dziedziczenia, 269
 - refleksji, 270
 - warunków, 769
 - wyjątków, 584
- struktura
 - katalogów, 43
 - ramki JFrame, 328
- struktury danych, 665
- strumień
 - ByteArrayInputStream, 526
 - ByteArrayOutputStream, 526
 - InputStream, 526
 - PrintStream, 526
 - wejściowy, 89
- styl
 - GTK, 316
 - Metal, 315, 366
 - Nimbus, 317
 - Synth, 317
- style
 - obramowań, 419
 - projektowania, 196
- suwak, slider, 413, 426, 430
- suwak z podziałką, 427
- SWT, 317

sygnatura metody, 171, 209
 symbole zastępcze, 65
 symulacja banku, 756, 759, 768
 synchronizacja, 756
 synchronizatory, 812, 813
 synchronizowane wątki, 763
 system kolorów, 342
 szablon

- bitset, 729
- vector, 235

 szerokość kolumny, 407
 szkielet rozgałęzienie-złączenie, 809, 812

Ś

ścieżka

- dostępu, 39
- klas, 187, 189

 ścieżki

- bezwzględne, 97
- względne, 97

 ścisła kontrola typów, 62, 274
 śledzenie

- przepływu wykonywania, 593
- stosu, 251, 581, 755

 środowisko działania apletu, 547
 środowisko programistyczne

- Eclipse, 44, 47
- NetBeans, 44

T

tabela metod, 210
 tablic, 116

- inicjowanie, 118
- kopiowanie, 119
- numerowanie, 116
- przeglądanie, 117
- sortowanie, 121

 tablica

- accounts, 760
- args, 120
- arrayToFill, 673
- par klucz – wartość, 556
- referencji, 628
- result, 123
- trójkątna, 129

 tablice

- anonimowe, 118
- generyczne, 644
- kopiowane przy zapisie, 796
- mieszające, 226, 428, 684, 703
- postrzępione, 127

typów ogólnych, 642
 typów wieloznacznych, 642
 wielowymiarowe, 124
 tasowanie, 720
 tasowanie elementów listy, 721
 technologia Flash, 29
 tekst, 406, 410
 terminal, 35
 test wydajności, 730
 testowanie

- apletu, 53
- blokad, 782
- mechanizmu własności, 551

 Tomcat, 519
 tożsamość obiektu, 134
 translacja

- metod ogólnych, 637
- poprzez wymazywanie typów, 648
- typów ogólnych, 639
- wyrażen generycznych, 637

 tryb pełnoekranowy, 325
 tworzenie

- akceleratora, 439
- apletów, 29, 53
- dziennika, 601
- egzemplarza klasy, 132
- etykiety, 409
- klas wyjątków, 570
- konstruktorów, 152
- listy cyklicznej, 668
- listy powiązanej, 668
- menu, 432
- obiektów, 96, 132, 137, 171
- obiektu proxy, 307
- obiektu typu Class, 249
- obramowań, 421
- ogólnych tablic, 643
- okien dialogowych, 484
- okna komunikatu, 477
- osobnego wątku, 741, 745
- plików JAR, 512
- pól wyboru, 413
- przycisku, 358, 362
- puli wątków, 802
- ramki, 318
- roboota, 617
- słuchacza akcji, 363
- tablic, 129, 203
 - postrzępionych, 128
 - generycznych, 644
 - trójkątnych, 129
 - z kolekcji, 718
- widoku mapy, 698

typ
 boolean, 66
 byte, 62
 char, 65
 Date, 92
 double, 64
 float, 63
 graniczny, 634
 int, 26, 62
 Integer, 243
 long, 63
 MIME, 519, 520
 osłony, 265
 parametryzowany, 650
 short, 62
 short int, 26
 surowy, 636, 641, 650
 wbudowany, 137
 wieloznaczny, 629, 634
 wieloznaczny z ograniczeniem nadtypów, 654
 wyliczeniowy, 77
 zwrotny, 639

typy
 całkowite, 62
 interfejsowe, 667
 kolekcji, 675
 komunikatów, 475
 konwersji, 74
 kursorów, 382
 listowe, 651
 ogólne, 627, 658
 parametryzowane, 627
 podstawowe, 221
 sparametryzowane, generic types, 32
 surowe, 234
 tablicowe, 221
 wieloznaczne, 650, 653
 bez ograniczeń, 655
 mechanizm chwywania, 656
 wyliczeniowe, 245, 704
 zmiennoprzecinkowe, 63
 zwrotne kowariantne, 639

U

ukrywanie danych, 133
 UML, Unified Modeling Language, 136
 Unicode, 65
 uruchamianie
 apletu, 53, 535
 aplikacji graficznej, 50
 aplikacji Java Web Start, 519, 520
 osobnego wątku, 741

programów w konsoli, 45
 programu, 44, 49
 uruchomienie kilku wątków, 743
 usługi
 API, 526
 JNLP, 526
 ustawianie
 preferencji, 557
 ścieżki klas, 189
 usuwanie
 elementów, 670
 elementów z kolekcji, 673
 elementu z listy powiązanej, 676
 elementu z tablicy, 675
 nieużytków, garbage collecting, 22, 702
 przedziału, 712
 UTC, Coordinated Universal Time, 139
 UTF-16, 66
 utrata wyjątku, 579

V

varargs, 244

W

warstwa
 implementacji, 666
 interfejsów, 666
 ramki, 327
 wartości
 graniczne przedziału, 712
 zwrotne okna potwierdzenia, 476
 wartość
 NaN, 64, 70
 null, 77, 81, 139, 172
 skrótu, hash value, 227
 warunek wstępny, 590
 warunki, 765, 769
 wątek, thread, 735, 815
 dystrybucji zdarzeń, 319, 741, 785, 819, 828
 roboczy, 824
 sterowania, thread of control, 735
 TransferRunnable, 785
 wyliczeniowy, 791
 zablokowany, 747
 zamknięty, 747
 wątki
 kończenie działania, 746
 oczekujące, 766
 niesynchronizowane, 763
 priorytet, 752
 stan

- wątki
 - stan
 - BLOCKED, 749
 - NEW, 749
 - RUNNABLE, 749
 - TERMINATED, 749
 - TIMED WAITING, 749
 - WAITING, 749
 - synchronizowane, 763
 - usunięte z kolejki, 766
 - wyłączanie, 760
 - zamienianie w demona, 753
 - zamknięcie, 752
- wczytywanie zasobów, 516
- wejście, 89
- wejście System.in, 96
- wersje Javy, 32
- węzeł, 556, 560
- wiązanie
 - dynamiczne, 204, 209–211
 - statyczne, 209
- widoczność metod, 211
- widok, view, 394, 710
 - kolekcji, 715
 - listowy elementów, 716
 - mapy, 698
 - podprzedziału, 717
 - poła tekstowego, 394
- widoki
 - kontrolowane, 714
 - niemodyfikowalne, 712
 - przedziałowe, 711
 - synchronizowane, 713
- wielkie liczby, 114
- wielkość liter, 58
- wielowątkowość, 28, 33, 735
- wielozadaniowość, 735
- wiersz poleceń, 44, 120
- wizualne budowanie interfejsów, 51
- własne typy wyjątków, 571
- własności
 - czcionki, 450, 454
 - interfejsów, 276
 - interfejsu ButtonModel, 397
 - klas proxy, 311
 - metody equals, 222
 - monitorów, 775
 - ramek, 322, 551
 - wątków, 752
- własność, property, 322
- właściwości
 - list, 721
 - systemowe, 554
- włączanie
 - asercji, 588
 - zegara, 293
- wnioskowanie o typie, 632
- wprowadzanie tekstu, 406
- wskazniki, 25
 - do funkcji, 286
 - do metod, 264
 - do obiektów, 140
- współczynnik zapamiętania tablicy, 686
- współrzędne
 - figur, 333–336
 - kodowe znaków, 66, 81
 - siatki, 451
- wstawianie komentarzy, 190
- wybór
 - kolorów, 505
 - plików, 495
- wyciszanie wyjątków, 586
- wydajność, 27
- wydłużenie
 - dolne, 346
 - górne, 346
- wyjątek
 - ArrayIndexOutOfBoundsException, 117
 - ArrayIndexOutOfBoundsException, 566, 816
 - ArrayStoreException, 642, 650
 - BadCastException, 658
 - Class.forName, 250
 - ClassCastException, 213, 262, 650, 714
 - CloneNotSupportedException, 283
 - ConcurrentModificationException, 679, 797
 - EmptyStackException, 584, 586
 - FileNotFoundException, 97, 528, 569, 648
 - IllegalAccessException, 257
 - IllegalMonitorStateException, 773
 - IllegalStateException, 670, 674, 683
 - InterruptedException, 737, 742, 748
 - IOException, 569, 572
 - NoSuchElementException, 674, 695
 - NullPointerException, 566, 586, 590
 - RuntimeException, 565, 566
 - ServletException, 575
 - ThreadDeath, 785
 - typu RuntimeException, 583
 - UnavailableServiceException, 526
 - UnsupportedOperationException, 711–715
- wyjątki
 - kontrolowane, 249, 567, 646
 - konwersji, 629
 - niekontrolowane, 250, 567, 647
 - typu IOException, 569
 - typu SQLException, 576
 - zabezpieczeń, 554
 - wykonawcze, 565

wyjątków, 563
 deklarowanie, 567
 powtórne generowanie, 575
 przechwytywanie, 250, 571
 przechwytywanie wielu typów, 574
 przekazywanie, 586
 własne typy, 571
 zgłaszanie, 569

wyjście, 89

wyjście System.out, 96

wykres, 543

wykres słupkowy, 542

wyliczenia, 727

wyłączenie

- asercji, 588
- dziedziczenia, 211
- sprawdzania wyjątków, 646

wymazywanie typów, 637–648

wymiana danych, 489

wymuszanie rysowania, 329

wypełnianie figur, 340

wrażenia generyczne, 637

wyrównywanie etykiety i pól, 462

WYSIWYG, 395

wysyłanie

- rekordów, 597
- zdarzeń, 319

wyszukiwanie binarne, 722

wyścig, 756, 760

wyświetlanie

- elementów w przeglądarce, 548
- informacji, 327
- komponentów, 614
- obrazów, 52, 351
- ramki, 320
- rekordów dziennika, 604
- tekstu, 327, 329
- zasobu, 515

wyłączanie wątku, 750, 760

wywołanie

- dowolnych metod, 264
- innego konstruktora, 174
- przez nazwę, 165
- przez referencję, 164
- przez wartość, 164
- setFirst(null), 655

wzorce

- nazw plików dziennika, 599
- projektowe, 392

wzorzec

- Composite, 393
- Decorator, 393
- MVC, 392–396
- Strategy, 393

X

XML, 33

Z

zachowanie obiektu, 133

zagnieżdżanie

- bloków instrukcji, 98
- pętli, 113

zakleszczenie, deadlock, 766, 778

zależność, 135

zamiana parametrów obiektowych, 168

zamykanie

- aplikacji, 320
- ramki, 320
- wątków, 752

zapewnianie tablicy, 237

zapis

- błędów w pliku, 611
- danych w repozytorium, 556
- do dziennika, 592–594
- plików, 96
- preferencji użytkownika, 549

zarządca rozkładu, layout manager, 391, 400, 448

- CircleLayout, 469
- brzegowego, 400
- ciągłego, 398
- FlowLayout, 402
- grupowego, 449
- niestandardowy, 469
- siatkowego, 402

zasada

- jednego wątku, 816, 827
- uczciwości, 764
- zamienialności, 207

zasięg

- blokowy, 98
- pakietów, 185
- zmiennych, 98

zasoby, resources, 515

zastosowanie

- asercji, 589, 590
- klas abstrakcyjnych, 279
- klas wewnętrznych, 294
- kolejek priorytetowych, 696
- parametrów Class<T>, 659
- refleksji, 252
- typów wyliczeniowych, 246

zawartość pól danych, 257

zawijanie wierszy, 411

zbiór, set, 686, 697

- HashSet, 684
- TreeSet, 688–691
- uporządkowany, 688

- zdarzenia, 355, 389
 - interfejs nasłuchu, 356
 - myszy, 380, 438
 - niskiego poziomu, 388
 - okna, 370
 - semantyczne, 388
 - słuchacz, 356
 - źródło, 356
- zdarzenie, 287
 - FocusEvent, 388
 - KeyEvent, 388
 - MouseEvent, 388
 - MouseEvent, 388
 - MouseEvent, 388
 - WindowEvent, 369, 388
- zdjęcie blokady, 766
- zegar, 286, 293
- zezwolenie, permit, 812
- zgłaszanie wyjątków, 569
- zintegrowane środowisko programistyczne, IDE, 39, 47
- zmiana
 - koloru, 374
 - koloru tła, 507
 - rozmiaru tablicy, 117
 - stanu okna, 372, 388
 - stylu, 366, 368
 - typu wyjątków, 647
 - własności czcionek, 454
- zmienna, 66
- zmienna środowiskowa
 - CLASSPATH, 46, 189
 - Path, 40, 41
- zmienne
 - atomowe, 777
 - finalne, 299, 777
 - finalne puste, 299
 - interfejsowe, 277
 - lokalne, 153
 - lokalne wątków, 781
 - obiektywne, 137–140, 216
 - parametryczne, 174
 - polimorficzne, 207
 - statyczne, 159
 - tablicowe, 116, 119
 - typowe, 630, 633, 643, 645
 - warunkowe, 765
- znacznik
 - @author, 192
 - @deprecated, 193
 - @link, 194
 - @Override, 225
 - @param, 192
 - @see, 193
 - @since, 193
 - @version, 192
 - append, 599
 - applet, 54, 537
 - object, 540
 - param, 541, 542
- znaczniki
 - dokumentacyjne, 190
 - polecenia printf, 93
- znak
 - \$, 67
 - /, 516, 560
 - ampersand, 634
 - konwersji, 92
 - końca pliku, 565
 - końca wiersza, 514
 - powrotu karetki, 60
 - równości, 68
 - trzykropka, 244
- znaki
 - dodatkowe, 66
 - echa, 410
 - konwersji Date i Time, 94, 95
 - konwersji polecenia printf, 93
 - nowego wiersza, 477
 - specjalne, 65
- zniszczenie danych, 757
- zoptymalizowane wywoływanie metod, 212
- zwalnianie blokady, 764
- zwracanie referencji, 157

Ż

- źródła zdarzeń biblioteki AWT, 389
- źródło zdarzeń, event sources, 356, 389

Ż

- żądanie zamknięcia wątku, 746, 747

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**

WYKORZYSTAJ SIĘ OBIEKTÓW. ZACZNIJ PROGRAMOWAĆ OBIEKTOWO W JĘZYKU JAVA!

Pomimo zaawansowanego wieku Java wciąż jest na topie. Ten język programowania oraz narzędzia z nim powiązane są najczęściej wybierane do tworzenia rozbudowanych systemów informatycznych. Skąd ta popularność? Przejrzysta składnia, obsługa nowoczesnych technik przesyłania informacji, automatyczne czyszczenie pamięci to tylko niektóre z atutów Javy. Jeżeli dołożymy do tego ogromną rzeszę użytkowników chętnych do pomocy, wszystko staje się jasne. Java jeszcze długo będzie na świeczniku!

Kolejne wydanie tej cenionej książki zostało zaktualizowane o nowości, które pojawiły się w wersji 7 platformy Java Standard Edition. W trakcie lektury poznasz składnię języka oraz wszystkie istotne kwestie związane z programowaniem w Javie. Zrozumiesz założenia programowania obiektowego, nauczysz się korzystać z interfejsów oraz obsługiwać wyjątki. Przekonasz się również, jakie ułatwienia w tym zakresie oferuje Java 7 — obsługa wielu wyjątków w ramach jednego bloku catch to tylko jedno z wielu udogodnień. Książka ta jest idealną pozycją dla wszystkich osób chcących poznać język Java. Sprawdzi się ona również w rękach doświadczonych programistów — jako źródło informacji na temat nowości w Java Standard Edition 7.

POZNAJ:

- podstawy języka Java
- zasady programowania obiektowego
- zastosowanie interfejsów
- nowości wprowadzone w ostatniej wersji Javy

 PRENTICE HALL
PEARSON EDUCATION

helion.pl
księgarnia
internetowa

Nr katalogowy: 14912

 Księgarnia internetowa
<http://helion.pl>

 Zamówienia telefoniczne:
0 801 339900
 **0 601 339900**



Helion

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

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

Informatyka w najlepszym wydaniu



cena: 99,00 zł

ISBN 978-83-246-7758-0



9 788324 677580