

TAJNIKI JĘZYKA JAVA 7!

 PRENTICE  
HALL

# JAVA<sup>®</sup>

Techniki zaawansowane

WYDANIE IX



CAY S. HORSTMANN · GARY CORNELL

 **Helion**

Tytuł oryginału: Core Java, Volume II - Advanced Features (9th Edition)

Tłumaczenie: Jaromir Senczyk

ISBN: 978-83-246-7762-7

Authorized translation from the English language edition, entitled CORE JAVA, VOLUME II – ADVANCED FEATURES, Ninth Edition; ISBN 013708160X; 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.

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

Wydawnictwo HELION

ul. Kościuszki 1c, 44-100 GLIWICE

tel. 032 231 22 19, 032 230 98 63

e-mail: [helion@helion.pl](mailto: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/javtz9.zip>

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/javtz9>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzje.

Printed in Poland.

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

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

# Spis treści

<b>Przedmowa</b> .....	<b>11</b>
<b>Podziękowania</b> .....	<b>15</b>
<b>Rozdział 1. Strumienie i pliki</b> .....	<b>17</b>
1.1. Strumienie .....	17
1.1.1. Odczyt i zapis bajtów .....	18
1.1.2. Zoo pełne strumieni .....	20
1.1.3. Łączenie filtrów strumieni .....	24
1.2. Strumienie tekstowe .....	28
1.2.1. Zapisywanie tekstu .....	28
1.2.2. Wczytywanie tekstu .....	31
1.2.3. Zapis obiektów w formacie tekstowym .....	31
1.2.4. Zbiory znaków .....	34
1.3. Odczyt i zapis danych binarnych .....	39
1.3.1. Strumienie plików o swobodnym dostępie .....	42
1.4. Archiwa ZIP .....	46
1.5. Strumienie obiektów i serializacja .....	49
1.5.1. Format pliku serializacji obiektów .....	54
1.5.2. Modyfikowanie domyślnego mechanizmu serializacji .....	60
1.5.3. Serializacja singletonów i wyliczeń .....	62
1.5.4. Wersje .....	63
1.5.5. Serializacja w roli klonowania .....	65
1.6. Zarządzanie plikami .....	68
1.6.1. Ścieżki dostępu .....	68
1.6.2. Odczyt i zapis plików .....	70
1.6.3. Kopiowanie, przenoszenie i usuwanie plików .....	72
1.6.4. Tworzenie plików i katalogów .....	72
1.6.5. Informacje o plikach .....	74
1.6.6. Przeglądanie plików w katalogu .....	75
1.6.7. Systemy plików ZIP .....	78
1.7. Mapowanie plików w pamięci .....	79
1.7.1. Struktura bufora danych .....	86
1.7.2. Blokowanie plików .....	88
1.8. Wyrażenia regularne .....	90

<b>Rozdział 2. Język XML .....</b>	<b>101</b>
2.1. Wprowadzenie do języka XML .....	102
2.1.1. Struktura dokumentu XML .....	104
2.2. Parsowanie dokumentów XML .....	107
2.3. Kontrola poprawności dokumentów XML .....	118
2.3.1. Definicje typów dokumentów .....	119
2.3.2. XML Schema .....	126
2.3.3. Praktyczny przykład .....	129
2.4. Wyszukiwanie informacji i XPath .....	142
2.5. Przestrzenie nazw .....	148
2.6. Parsery strumieniowe .....	150
2.6.1. Wykorzystanie parsera SAX .....	151
2.6.2. Wykorzystanie parsera StAX .....	156
2.7. Tworzenie dokumentów XML .....	160
2.7.1. Dokumenty bez przestrzeni nazw .....	160
2.7.2. Dokumenty z przestrzenią nazw .....	160
2.7.3. Zapisywanie dokumentu .....	161
2.7.4. Przykład: tworzenie pliku SVG .....	162
2.7.5. Tworzenie dokumentu XML za pomocą parsera StAX .....	165
2.8. Przekształcenia XSL .....	172
<b>Rozdział 3. Programowanie aplikacji sieciowych .....</b>	<b>183</b>
3.1. Połączenia z serwerem .....	183
3.1.1. Limity czasu gniazd .....	187
3.1.2. Adresy internetowe .....	189
3.2. Implementacja serwerów .....	191
3.2.1. Obsługa wielu klientów .....	194
3.2.2. Połączenia częściowo zamknięte .....	196
3.3. Przerywanie działania gniazd sieciowych .....	198
3.4. Połączenia wykorzystujące URL .....	204
3.4.1. URL i URI .....	205
3.4.2. Zastosowanie klasy URLConnection do pobierania informacji .....	207
3.4.3. Wysyłanie danych do formularzy .....	216
3.5. Wysyłanie poczty elektronicznej .....	222
<b>Rozdział 4. Programowanie baz danych: JDBC .....</b>	<b>227</b>
4.1. Architektura JDBC .....	228
4.1.1. Typy sterowników JDBC .....	228
4.1.2. Typowe zastosowania JDBC .....	229
4.2. Język SQL .....	231
4.3. Instalacja JDBC .....	235
4.3.1. Adresy URL baz danych .....	237
4.3.2. Pliki JAR zawierające sterownik .....	237
4.3.3. Uruchamianie bazy danych .....	237
4.3.4. Rejestracja klasy sterownika .....	238
4.3.5. Nawiązywanie połączenia z bazą danych .....	239
4.4. Wykonywanie poleceń języka SQL .....	242
4.4.1. Zarządzanie połączeniami, poleceniami i zbiorami wyników .....	245
4.4.2. Analiza wyjątków SQL .....	246
4.4.3. Wypełnianie bazy danych .....	248

4.5.	Wykonywanie zapytań .....	252
4.5.1.	Polecenia przygotowane .....	252
4.5.2.	Odczyt i zapis dużych obiektów .....	258
4.5.3.	Sekwencje sterujące .....	260
4.5.4.	Zapytania o wielu zbiorach wyników .....	261
4.5.5.	Pobieranie wartości kluczy wygenerowanych automatycznie .....	262
4.6.	Przewijalne i aktualizowalne zbiory wyników zapytań .....	263
4.6.1.	Przewijalne zbiory wyników .....	263
4.6.2.	Aktualizowalne zbiory rekordów .....	265
4.7.	Zbiory rekordów .....	270
4.7.1.	Tworzenie zbiorów rekordów .....	270
4.7.2.	Buforowane zbiory rekordów .....	271
4.8.	Metadane .....	274
4.9.	Transakcje .....	283
4.9.1.	Punkty kontrolne .....	284
4.9.2.	Aktualizacje wsadowe .....	284
4.9.3.	Zaawansowane typy języka SQL .....	287
4.10.	Zaawansowane zarządzanie połączeniami .....	288

## **Rozdział 5. Internacjonalizacja ..... 291**

5.1.	Lokalizatory .....	292
5.2.	Formaty liczb .....	297
5.2.1.	Waluty .....	302
5.3.	Data i czas .....	304
5.4.	Porządek alfabetyczny .....	311
5.4.1.	Moc uporządkowania .....	312
5.4.2.	Rozkład .....	313
5.5.	Formatowanie komunikatów .....	318
5.5.1.	Formatowanie z wariantami .....	320
5.6.	Pliki tekstowe i zbiory znaków .....	322
5.6.1.	Internacjonalizacja a pliki źródłowe programów .....	322
5.7.	Komplety zasobów .....	324
5.7.1.	Wyszukiwanie kompletów zasobów .....	324
5.7.2.	Pliki właściwości .....	325
5.7.3.	Klasy kompletów zasobów .....	326
5.8.	Kompletny przykład .....	328

## **Rozdział 6. Zaawansowane możliwości pakietu Swing ..... 343**

6.1.	Listy .....	343
6.1.1.	Komponent JList .....	344
6.1.2.	Modele list .....	349
6.1.3.	Wstawianie i usuwanie .....	354
6.1.4.	Odrysowywanie zawartości listy .....	355
6.2.	Tabele .....	359
6.2.1.	Najprostsze tabele .....	359
6.2.2.	Modele tabel .....	363
6.2.3.	Wiersze i kolumny .....	367
6.2.4.	Rysowanie i edycja komórek .....	383
6.3.	Drzewa .....	394
6.3.1.	Najprostsze drzewa .....	395
6.3.2.	Przeglądanie węzłów .....	410

6.3.3.	Rysowanie węzłów .....	412
6.3.4.	Nasłuchiwanie zdarzeń w drzewach .....	415
6.3.5.	Własne modele drzew .....	421
6.4.	Komponenty tekstowe .....	429
6.4.1.	Śledzenie zmian zawartości komponentów tekstowych .....	430
6.4.2.	Sformatowane pola wejściowe .....	433
6.4.3.	Komponent JSpinner .....	449
6.4.4.	Prezentacja HTML za pomocą JEditorPane .....	457
6.5.	Wskaźniki postępu .....	463
6.5.1.	Paski postępu .....	463
6.5.2.	Monitory postępu .....	466
6.5.3.	Monitorowanie postępu strumieni wejścia .....	469
6.6.	Organizatory komponentów i dekoratory .....	474
6.6.1.	Panele dzielone .....	475
6.6.2.	Panele z zakładkami .....	478
6.6.3.	Panele pulpitu i ramki wewnętrzne .....	483
6.6.4.	Rozmieszczenie kaskadowe i sąsiadujące .....	487
6.6.5.	Zgłaszanie weta do zmiany właściwości .....	495
<b>Rozdział 7. Zaawansowane możliwości biblioteki AWT .....</b>		<b>505</b>
7.1.	Potokowe tworzenie grafiki .....	506
7.2.	Figury .....	508
7.2.1.	Wykorzystanie klas obiektów graficznych .....	511
7.3.	Pola .....	523
7.4.	Ślad pędzla .....	524
7.5.	Wypełnienia .....	532
7.6.	Przekształcenia układu współrzędnych .....	534
7.7.	Przycinanie .....	539
7.8.	Przezroczystość i składanie obrazów .....	541
7.9.	Wskazówki operacji graficznych .....	549
7.10.	Czytanie i zapisywanie plików graficznych .....	555
7.10.1.	Wykorzystanie obiektów zapisu i odczytu plików graficznych .....	555
7.10.2.	Odczyt i zapis plików zawierających sekwencje obrazów .....	560
7.11.	Operacje na obrazach .....	565
7.11.1.	Dostęp do danych obrazu .....	565
7.11.2.	Filtrowanie obrazów .....	571
7.12.	Drukowanie .....	580
7.12.1.	Drukowanie grafiki .....	580
7.12.2.	Drukowanie wielu stron .....	589
7.12.3.	Podgląd wydruku .....	591
7.12.4.	Usługi drukowania .....	599
7.12.5.	Usługi drukowania za pośrednictwem strumieni .....	603
7.12.6.	Atrybuty drukowania .....	604
7.13.	Schówek .....	610
7.13.1.	Klasy i interfejsy umożliwiające przekazywanie danych .....	611
7.13.2.	Przekazywanie tekstu .....	612
7.13.3.	Interfejs Transferable i formaty danych .....	615
7.13.4.	Przekazywanie obrazów za pomocą schowka .....	617
7.13.5.	Wykorzystanie schowka systemowego do przekazywania obiektów Java .....	621
7.13.6.	Zastosowanie lokalnego schowka do przekazywania referencji obiektów .....	624

7.14.	Mechanizm „przeciągnij i upuść” .....	625
7.14.1.	Przekazywanie danych pomiędzy komponentami Swing .....	627
7.14.2.	Źródła przeciąganych danych .....	631
7.14.3.	Cele upuszczanych danych .....	633
7.15.	Integracja z macierzystą platformą .....	641
7.15.1.	Ekran powitalny .....	641
7.15.2.	Uruchamianie macierzystych aplikacji pulpitu .....	646
7.15.3.	Zasobnik systemowy .....	651

## **Rozdział 8. JavaBeans ..... 657**

8.1.	Dlaczego ziarnka? .....	658
8.2.	Proces tworzenia ziarek JavaBeans .....	660
8.3.	Wykorzystanie ziarek do tworzenia aplikacji .....	662
8.3.1.	Umieszczanie ziarek w plikach JAR .....	663
8.3.2.	Korzystanie z ziarek .....	664
8.4.	Wzorce nazw właściwości ziarek i zdarzeń .....	669
8.5.	Typy właściwości ziarek .....	673
8.5.1.	Właściwości proste .....	673
8.5.2.	Właściwości indeksowane .....	674
8.5.3.	Właściwości powiązane .....	674
8.5.4.	Właściwości ograniczone .....	676
8.6.	Klasa informacyjna ziarnka .....	683
8.7.	Edytory właściwości .....	687
8.7.1.	Implementacja edytora właściwości .....	690
8.8.	Indywidualizacja ziarnka .....	697
8.8.1.	Implementacja klasy indywidualizacji .....	699
8.9.	Trwałość ziarek JavaBeans .....	705
8.9.1.	Zastosowanie mechanizmu trwałości JavaBeans dla dowolnych danych .....	709
8.9.2.	Kompletny przykład zastosowania trwałości JavaBeans .....	715

## **Rozdział 9. Bezpieczeństwo ..... 727**

9.1.	Ładowanie klas .....	728
9.1.1.	Hierarchia klas ładowania .....	730
9.1.2.	Zastosowanie procedur ładujących w roli przestrzeni nazw .....	732
9.1.3.	Implementacja własnej procedury ładującej .....	733
9.2.	Weryfikacja kodu maszyny wirtualnej .....	738
9.3.	Menedżery bezpieczeństwa i pozwolenia .....	742
9.3.1.	Bezpieczeństwo na platformie Java .....	744
9.3.2.	Pliki polityki bezpieczeństwa .....	747
9.3.3.	Tworzenie własnych klas pozwoleń .....	755
9.3.4.	Implementacja klasy pozwoleń .....	756
9.4.	Uwierzelnianie użytkowników .....	762
9.4.1.	Moduły JAAS .....	767
9.5.	Podpis cyfrowy .....	776
9.5.1.	Skróty wiadomości .....	777
9.5.2.	Podpisywanie wiadomości .....	779
9.5.3.	Weryfikacja podpisu .....	781
9.5.4.	Problem uwierzelniania .....	784
9.5.5.	Podpisywanie certyfikatów .....	786
9.5.6.	Żądania certyfikatu .....	787

9.6.	Podpisywanie kodu .....	788
9.6.1.	Podpisywanie plików JAR .....	789
9.6.2.	Certyfikaty twórców oprogramowania .....	793
9.7.	Szyfrowanie .....	795
9.7.1.	Szyfrowanie symetryczne .....	795
9.7.2.	Generowanie klucza .....	797
9.7.3.	Strumienie szyfrujące .....	801
9.7.4.	Szyfrowanie kluczem publicznym .....	803
<b>Rozdział 10. Skrypty, kompilacja i adnotacje .....</b>		<b>807</b>
10.1.	Skrypty na platformie Java .....	807
10.1.1.	Wybór silnika skryptów .....	808
10.1.2.	Wykonywanie skryptów i wiązania zmiennych .....	809
10.1.3.	Przekierowanie wejścia i wyjścia .....	811
10.1.4.	Wywoływanie funkcji i metod skryptów .....	812
10.1.5.	Kompilacja skryptu .....	814
10.1.6.	Przykład: skrypty i graficzny interfejs użytkownika .....	815
10.2.	Interfejs kompilatora .....	819
10.2.1.	Kompilacja w najprostszy sposób .....	820
10.2.2.	Stosowanie zadań kompilacji .....	820
10.2.3.	Przykład: dynamiczne tworzenie kodu w języku Java .....	826
10.3.	Stosowanie adnotacji .....	830
10.3.1.	Przykład: adnotacje obsługi zdarzeń .....	832
10.4.	Składnia adnotacji .....	837
10.5.	Adnotacje standardowe .....	841
10.5.1.	Adnotacje kompilacji .....	842
10.5.2.	Adnotacje zarządzania zasobami .....	842
10.5.3.	Metaadnotacje .....	843
10.6.	Przetwarzanie adnotacji w kodzie źródłowym .....	845
10.7.	Inżynieria kodu bajtowego .....	851
10.7.1.	Modyfikacja kodu bajtowego podczas ładowania .....	857
<b>Rozdział 11. Obiekty rozproszone .....</b>		<b>861</b>
11.1.	Role klienta i serwera .....	862
11.2.	Wywołania zdalnych metod .....	864
11.2.1.	Namiastka i szeregowanie parametrów .....	864
11.3.	Model programowania RMI .....	866
11.3.1.	Interfejsy i implementacje .....	866
11.3.2.	Rejestr RMI .....	868
11.3.3.	Przygotowanie wdrożenia .....	871
11.3.4.	Rejestrowanie aktywności RMI .....	874
11.4.	Parametry zdalnych metod i wartości zwracane .....	876
11.4.1.	Przekazywanie obiektów zdalnych .....	876
11.4.2.	Przekazywanie obiektów, które nie są zdalne .....	876
11.4.3.	Dynamiczne ładowanie klas .....	878
11.4.4.	Zdalne referencje obiektów o wielu interfejsach .....	883
11.4.5.	Zdalne obiekty i metody equals, hashCode oraz clone .....	884
11.5.	Aktywacja zdalnych obiektów .....	884



<b>Rozdział 12. Metody macierzyste .....</b>	<b>891</b>
12.1. Wywołania funkcji języka C z programów w języku Java .....	892
12.2. Numeryczne parametry metod i wartości zwracane .....	898
12.2.1. Wykorzystanie funkcji printf do formatowania liczb .....	899
12.3. Łańcuchy znaków jako parametry .....	900
12.4. Dostęp do składowych obiektu .....	906
12.4.1. Dostęp do pól instancji .....	906
12.4.2. Dostęp do pól statycznych .....	910
12.5. Sygnatury .....	911
12.6. Wywoływanie metod języka Java .....	912
12.6.1. Wywoływanie metod obiektów .....	912
12.6.2. Wywoływanie metod statycznych .....	916
12.6.3. Konstruktory .....	917
12.6.4. Alternatywne sposoby wywoływania metod .....	917
12.7. Tablice .....	919
12.8. Obsługa błędów .....	923
12.9. Interfejs programowy wywołań języka Java .....	927
12.10. Kompletny przykład: dostęp do rejestru systemu Windows .....	932
12.10.1. Rejestr systemu Windows .....	933
12.10.2. Interfejs dostępu do rejestru na platformie Java .....	934
12.10.3. Implementacja dostępu do rejestru za pomocą metod macierzystych ....	935
<b>Skorowidz .....</b>	<b>949</b>



# 7

## Zaawansowane możliwości biblioteki AWT

W tym rozdziale:

- Potokowe tworzenie grafiki.
- Figury.
- Pola.
- Ślad pędzla.
- Wypełnianie.
- Przekształcenia układu współrzędnych.
- Przycinanie.
- Przezroczystość i składanie obrazów.
- Wskazówki operacji graficznych.
- Czytanie i zapisywanie plików graficznych.
- Operacje na obrazach.
- Drukowanie.
- Schowek.
- Mechanizm „przeciągnij i upuść”.
- Integracja z macierzystą platformą.

Do tworzenia prostych rysunków możemy wykorzystać metody klasy `Graphics`. Metody te są wystarczające w przypadku podstawowych apletów i aplikacji, ale nie nadają się do zastosowań wymagających tworzenia skomplikowanych rysunków lub potrzebujących daleko idącej kontroli nad tworzoną grafiką. Do takich zastosowań przeznaczona jest biblioteka Java 2D, której możliwości przedstawimy w bieżącym rozdziale.

Zajmiemy się także omówieniem implementacji drukowania w aplikacjach Java.

Omówimy też dwa sposoby przekazywania danych pomiędzy programami: schowek systemowy oraz mechanizm „przeciagnij i upuść”. Mogą one być wykorzystywane do przekazywania danych między dwiema aplikacjami Java lub pomiędzy aplikacją Java i programem rodzimym danej platformy. Na koniec przedstawimy techniki pozwalające zbliżyć zachowanie aplikacji tworzonych w języku Java do aplikacji macierzystych platformy: tworzenie ekranu powitania i korzystanie z zasobnika systemowego.

## 7.1. Potokowe tworzenie grafiki

JDK 1.0 dysponował bardzo prostym mechanizmem tworzenia grafiki. Wystarczyło wybrać jedynie kolor i tryb rysowania, a następnie wywołać odpowiednią metodę klasy `Graphics`, na przykład `drawRect` lub `fillOval`. Tworzenie grafiki za pomocą Java 2D udostępnia wiele więcej możliwości:

- tworzenie różnorodnych *figur*,
- kontrolę nad sposobem tworzenia *obrysu* figur,
- *wypełnianie* figur różnymi kolorami i ich odcieniami, a także wzorami wypełnień,
- metody *przekształceń* figur — przesunięcia, skalowania, obrotu i zmian proporcji,
- *przycinanie* figur do dowolnie wybranych obszarów,
- wybór *zasad składania* obrazów opisujących sposób tworzenia kombinacji pikseli nowej figury z istniejącymi już pikselami tła,
- *wskazówki tworzenia grafiki* umożliwiające uzyskanie kompromisu pomiędzy szybkością tworzenia grafiki a jej jakością.

Aby narysować figurę, należy kolejno wykonać następujące kroki.

1. Musimy uzyskać obiekt klasy `Graphics2D`, która jest klasą pochodną klasy `Graphics`. Począwszy od Java SE 1.2 metody, takie jak `paint` czy `paintComponent`, automatycznie otrzymują jako parametr obiekty klasy `Graphics2D`. Wystarczy więc wykonać następujące rzutowanie:

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

2. Korzystamy z metody `setRenderingHints` w celu ustalenia kompromisu między szybkością tworzenia grafiki a jej jakością.

```
RenderingHints hints = . . . ;
g2.setRenderingHints(hints);
```

- 3.** Wywołujemy metodę `setStroke` w celu określenia rodzaju *ślada pędzla*, który zostanie użyty do narysowania obrysu figury. Musimy wybrać odpowiednią grubość śladu pędzla oraz sposób jego pozostawiania (ciągły, przerywany).

```
Stroke stroke = . . . ;
g2.setStroke(stroke);
```

- 4.** Wywołujemy metodę `setPaint` w celu określenia sposobu *wypełnienia* obszaru ograniczonego obrysem. Należy wybrać wypełnienie stałym kolorem, kolorem o zmieniających się odcieniach lub wzorem wypełnienia.

```
Paint paint = . . . ;
g2.setPaint(paint);
```

- 5.** Korzystamy z metody `clip` w celu określenia *obszaru przycięcia*.

```
Shape clip = . . . ;
g2.clip(clip);
```

- 6.** Wywołujemy metodę `transform`, aby *przekształcić* współrzędne użytkownika na współrzędne urządzenia. Przekształcenie takie stosujemy, jeśli w programie łatwiej nam posługiwać się własnym układem współrzędnych niż współrzędnymi pikseli.

```
AffineTransform transform = . . . ;
g2.transform(transform);
```

- 7.** Wywołujemy metodę `setComposite` dla określenia *zasady składania* obrazów opisującej tworzenie kombinacji nowych pikseli z już istniejącymi.

```
Composite composite = . . . ;
g2.setComposite(composite);
```

- 8.** Tworzymy figurę. Java 2D udostępnia w tym celu wiele obiektów i metod umożliwiających tworzenie kombinacji figur.

```
Shape shape = . . . ;
```

- 9.** Rysujemy i (lub) wypełniamy figurę. Jeśli narysujemy figurę, to powstanie jedynie jej obrys, który następnie możemy wypełnić.

```
g2.draw(shape);
g2.fill(shape);
```

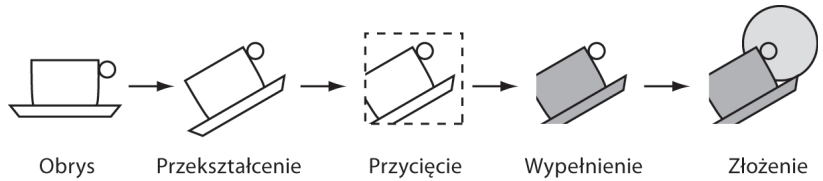
Oczywiście nie zawsze musimy realizować wszystkie wymienione etapy. Parametry kontekstu graficznego posiadają wartości domyślne wystarczające w wielu przypadkach.

W kolejnych podrozdziałach przedstawimy sposób tworzenia figur, definiowania śladów pędzla i wypełnień, przekształcenia i zasady składania obrazów.

Metody `set` nie wywołują żadnych operacji graficznych, a powodują jedynie zmiany stanu kontekstu graficznego. Podobnie utworzenie obiektu reprezentującego figurę nie powoduje narysowania jej reprezentacji. Operacje graficzne wykonywane są jedynie na skutek wywołania metod `draw` lub `fill`. W ich rezultacie reprezentacja graficzna figury tworzona jest w *potoku rysowania* (patrz rysunek 7.1).

**Rysunek 7.1.**

Potok rysowania



W potoku tym wykonywane są następujące operacje prowadzące do utworzenia reprezentacji graficznej obiektu.

1. Tworzony jest obrys figury.
2. Obrys figury jest przekształcany.
3. Obrys figury jest przycinany. Jeśli figura i obszar przycięcia nie mają części wspólnej, to przetwarzanie w potoku kończy się na tym etapie.
4. Wypełniany jest obszar figury powstały po przycięciu.
5. Piksele wypełnionej figury składane są z istniejącymi pikselami tła. (Na rysunku 7.1 koło stanowi fragment tła, na które nakładana jest figura filiżanki).

W kolejnym podrozdziale przedstawimy sposób definiowania figur, a następnie przejdziemy do omówienia kontekstów graficznych.

**API** java.awt.Graphics2D 1.2

- void draw(Shape s) rysuje obrys figury zgodnie z bieżącymi parametrami kontekstu graficznego.
- void fill(Shape s) wypełnia obrys figury zgodnie z bieżącymi parametrami kontekstu graficznego.

## 7.2. Figury

Klasa Graphics udostępnia szereg metod umożliwiających rysowanie figur:

```
drawLine
drawRectangle
drawRoundRect
draw3DRect
drawPolygon
drawPolyline
drawOval
drawArc
```

Posiada także odpowiadające im metody fill tworzenia *wypełnień* figur. Wszystkie te metody dostępne są w klasie Graphics już od wersji JDK 1.0. Natomiast Java 2D proponuje inne, bardziej obiektowe podejście. Zamiast metod udostępnia odpowiednie klasy:

```
Line2D
Rectangle2D
RoundRectangle2D
```

```

Ellipse2D
Arc2D
QuadCurve2D
CubicCurve2D
GeneralPath

```

Wszystkie wymienione klasy implementują interfejs `Shape`.

Zdefiniowana jest także klasa `Point2D`, która choć reprezentuje punkt o współrzędnych  $x$  i  $y$ , a nie figurę, to używana jest podczas definiowania figur.

Aby uzyskać reprezentację graficzną figury, musimy więc najpierw utworzyć obiekt klasy implementującej interfejs `Shape`, a następnie wywołać metodę `draw` klasy `Graphics2D`.

Jak łatwo zauważyć, klasy `Line2D`, `Rectangle2D`, `RoundRectangle2D`, `Ellipse2D` i `Arc2D` odpowiadają metodom `drawLine`, `drawRectangle`, `drawRoundRectangle`, `drawOval` i `drawArc` klasy `Graphics`. (Koncept „prostokąta w trzech wymiarach” zarzucono i dlatego metoda `draw3DRect` nie posiada swego odpowiednika w postaci klasy). Java 2D udostępnia dodatkowo klasy `QuadCurve2D` i `CubicCurve2D` reprezentujące krzywe drugiego i trzeciego stopnia. Omówimy je w dalszej części tego podrozdziału. Nie istnieje natomiast bezpośredni odpowiednik metody `drawPolygon`. Możemy jednak do tworzenia wielokątów wykorzystać klasę `GeneralPath`, która opisuje ścieżki złożone z odcinków, a także krzywych drugiego i trzeciego stopnia.

#### Klasy

```

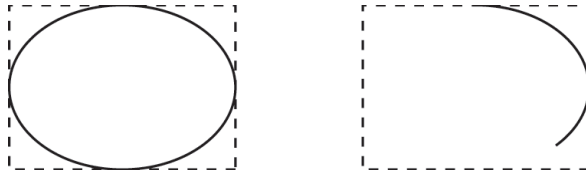
Rectangle2D
RoundRectangle2D
Ellipse2D
Arc2D

```

są pochodnymi klasy `RectangularShape`. Choć elipsy i łuki nie są prostokątami, to jednak podczas ich definiowania wykorzystywany jest *prostokąt ograniczający* (patrz rysunek 7.2).

#### Rysunek 7.2.

*Prostokąt ograniczający elipsę i łuk*



Każda z klas, której nazwa zakończona jest przyrostkiem 2D, posiada dwie klasy pochodne umożliwiające określanie współrzędnych za pomocą wartości typu `float` lub `double`. W książce *Java. Podstawy* spotkaliśmy już klasy `Rectangle2D.Float` i `Rectangle2D.Double`.

Ten sam schemat stosowany jest w przypadku pozostałych klas, na przykład `Arc2D.Float` i `Arc2D.Double`.

Implementacja wszystkich klas graficznych wykorzystuje w rzeczywistości współrzędne typu `float`, ponieważ reprezentacja tego typu zajmuje mniej miejsca i posiada wystarczającą dokładność dla operacji graficznych. Ponieważ jednak operowanie wartościami typu `float` jest w języku Java mało wygodne, to większość metod klas graficznych posiada parametry i zwraca wartości typu `double`. Zwykle więc jedynie podczas tworzenia obiektu klasy graficznej wybierac możemy pomiędzy konstruktorem o parametrach typu `float` i konstruktorem o parametrach typu `double`, na przykład:

```

Rectangle2D floatRect
    = new Rectangle2D.Float(5F, 10F, 7.5F, 15F);
Rectangle2D doubleRect
    = new Rectangle2D.Double(5, 10, 7.5, 15);

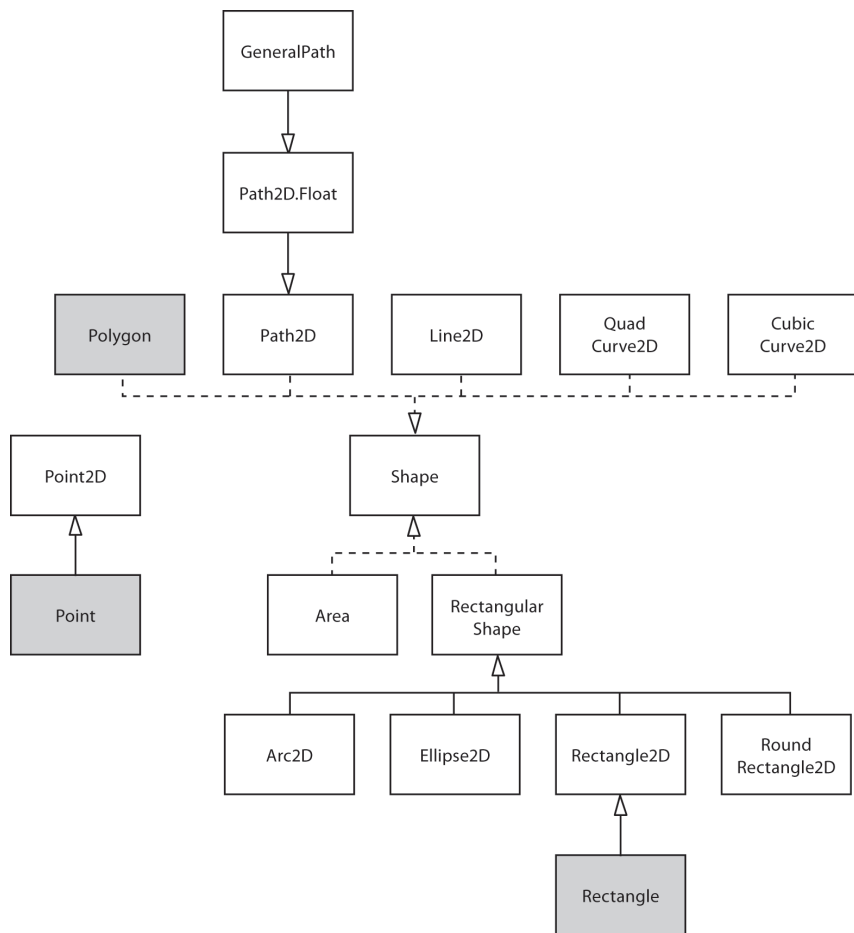
```

Klasy `xxx2D.Float` i `xxx2D.Double` są klasami pochodnymi klas `xxx2D`. Po utworzeniu obiektu nie ma już znaczenia, do której z klas pochodnych on należy. Do przechowywania referencji obiektu możemy wtedy używać zmiennej o typie klasy bazowej, tak jak pokazaliśmy to w powyższym przykładzie.

Jak łatwo domyślić się na podstawie nazw klas `xxx2D.Float` i `xxx2D.Double`, są one klasami wewnętrznymi klas `xxx2D`. Rozwiązanie takie ma na celu uniknięcie nadmiaru nazw klas zewnętrznych.

Rysunek 7.3 przedstawia zależności pomiędzy klasami reprezentującymi figury. Pominięto na nim podklasy `xxx2D.Float` i `xxx2D.Double`. Tradycyjne klasy dostępne zanim pojawiła się Java 2D zaznaczono na nim za pomocą prostokątów wypełnionych kolorem szarym.

**Rysunek 7.3.**  
Zależności  
pomiędzy klasami  
reprezentującymi  
figury





## 7.2.1. Wykorzystanie klas obiektów graficznych

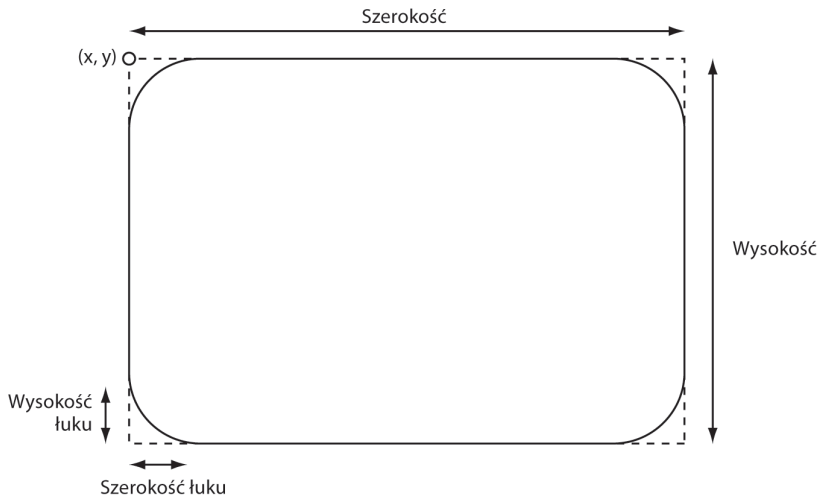
W rozdziale 7. książki *Java. Podstawy* pokazaliśmy już sposób wykorzystania klas `Rectangle2D`, `Ellipse2D` i `Line2D`. Teraz zajmiemy się więc pozostałymi klasami.

W przypadku klasy `RoundRectangle2D` musimy określić współrzędne lewego górnego wierzchołka figury, jej szerokość i wysokość, a także szerokość i wysokość obszaru narożnika, który powinien zostać zaokrąglony (patrz rysunek 7.4). Na przykład wywołanie

```
RoundRectangle2D r = new RoundRectangle2D.Double(150, 200, 100, 50, 20, 20)
```

utworzy obiekt reprezentujący prostokąt, którego narożniki będą zaokrąglone za pomocą łuków okręgu o promieniu 20 pikseli.

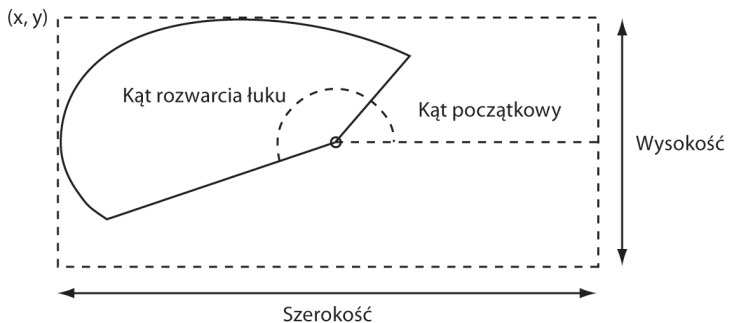
**Rysunek 7.4.**  
Parametry  
prostokątów klasy  
`RoundRectangle2D`



Tworząc łuk, musimy określić ograniczający go prostokąt, kąt początkowy, kąt rozpinający łuku (patrz rysunek 7.5) oraz sposób jego zamknięcia jako jedną z wartości `Arc2D.OPEN`, `Arc2D.PIE`, `Arc2D.CHORD`.

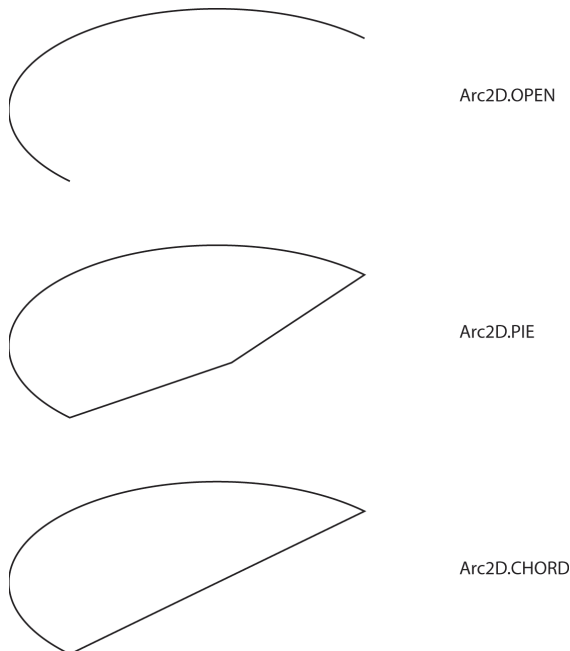
```
Arc2D a = new Arc2D(x, y, width, height, startAngle, arcAngle, closureType);
```

**Rysunek 7.5.**  
Parametry łuku  
klasy `Arc2D`



Rysunek 7.6 pokazuje różne rodzaje zamknięcia łuku.

**Rysunek 7.6.**  
Rodzaje łuków



Jeśli łuk jest eliptyczny, to wyznaczenie kątów łuku nie jest trywialne. Dokumentacja mówi na ten temat: „Kąty podaje się względem prostokąta ograniczającego w taki sposób, że kąt 45 stopni zawsze wypada na linii biegnącej ze środka elipsy do prawego górnego wierzchołka prostokąta ograniczającego. W rezultacie, jeśli prostokąt ograniczający jest wyraźnie dłuższy wzdłuż jednej osi, kąty do początku i końca segmentu łuku będą zniekształcone wzdłuż dłuższej osi ograniczenia”. Niestety dokumentacja milczy na temat sposobu wyznaczenia tego zniekształcenia. A oto szczegóły:

Założmy, że środkiem łuku jest początek układu współrzędnych, a punkt  $(x, y)$  leży na łuku. Wartość kąta zniekształconego wyznaczymy wtedy w sposób podany poniżej.

```
skewedAngle = Math.toDegrees(Math.atan2(-y * height, x * width));
```

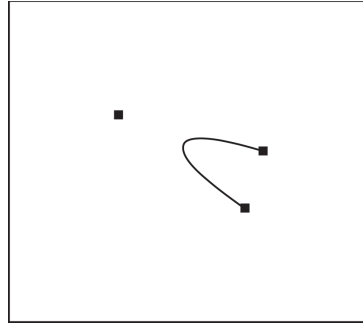
Otrzymana wartość będzie należeć do przedziału od  $-180$  do  $180$  stopni. W ten sposób musimy obliczyć wartość początkową i końcową zniekształconego kąta łuku, a następnie wyznaczyć ich różnicę. Jeśli początkowa wartość kąta bądź różnica wartości kąta jest ujemna, to należy dodać do niej 360. Uzyskaną wartość początkową kąta oraz różnicę wartości kąta przekazujemy konstruktorowi łuku.

Uruchamiając przykładowy program zamieszczony na końcu tego podrozdziału, możemy przekonać się, że opisana powyżej metoda daje właściwe wyniki (patrz rysunek 7.9).

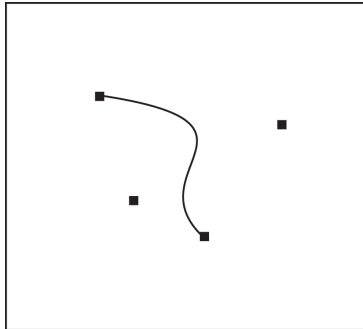
Java 2D udostępnia krzywe *drugiego* i *trzeciego* stopnia. W rozdziale tym nie będziemy zajmować się matematycznymi podstawami ich działania. Sugerujemy praktyczne zapoznanie się z ich możliwościami za pomocą programu z listingu 7.1. Rysunki 7.7 i 7.8 przedstawiają krzywe drugiego i trzeciego stopnia, które tworzone są przez określenie ich *końców* oraz jednego lub dwu *punktów kontrolnych*. Zmiana położenia punktów kontrolnych umożliwia zmianę kształtu krzywych.

**Rysunek 7.7.**

Krzywa drugiego stopnia

**Rysunek 7.8.**

Krzywa trzeciego stopnia



Parametrami konstruktorów krzywych drugiego i trzeciego stopnia są współrzędne ich końców i punktów kontrolnych:

```
QuadCurve2D q = new QuadCurve2D.Double(startX, startY,
    control1X, control1Y, endX, endY);
CubicCurve2D c = new CubicCurve2D.Double(startX, startY,
    control1X, control1Y, control2X, control2Y, endX, endY);
```

Krzywe drugiego stopnia nie są zbyt uniwersalne i dlatego w praktyce stosowane są rzadko. Natomiast krzywe trzeciego stopnia (takie jak krzywe Beziery reprezentowane przez klasę `CubicCurve2D`) wykorzystywane są powszechnie. Łącząc je w łańcuchy, w których koniec jednej krzywej jest początkiem następnej, możemy tworzyć skomplikowane kształty o dowolnym obrysie. Więcej informacji na ten temat zawiera książka *Computer Graphics: Principles and Practice, Second Edition in C* autorstwa Jamesa D. Foley, Andriasa van Dama, Stevena K. Feinera i in. (Addison-Wesley, 1995).

Klasa `GeneralPath` umożliwia tworzenie dowolnych sekwencji złożonych z odcinków prostych oraz krzywych drugiego i trzeciego stopnia. Współrzędne początku takiego obiektu określamy za pomocą metody `moveTo`, co pokazano poniżej.

```
GeneralPath path = new GeneralPath();
path.moveTo(10, 20);
```

Następnie rozbudowujemy sekwencję o odcinek linii prostej, krzywej drugiego lub trzeciego stopnia, korzystając odpowiednio z metod `lineTo`, `quadTo` bądź `curveTo`. W przypadku metody `lineTo` musimy określić jedynie koniec odcinka. W pozostałych wypadkach najpierw przekazujemy metodom współrzędne punktów kontrolnych, a następnie współrzędne końca segmentu, na przykład:

```
path.lineTo(20, 30);
path.curveTo(control1X, control1Y, control2X, control2Y, endX, endY);
```

Tworzoną sekwencję możemy zamknąć, korzystając z metody `closePath`. Rysuje ona odcinek do punktu, którego współrzędne zawierało ostatnie wywołanie metody `moveTo`.

Aby, korzystając z obiektu klasy `GeneralPath`, utworzyć wielokąt, należy wywołać metodę `moveTo`, przekazując jej współrzędne pierwszego wierzchołka, a następnie wywoływać metodę `lineTo`, podając współrzędne kolejnych wierzchołków. Wywołanie metody `closePath` spowoduje połączenie ostatniego z wierzchołków z pierwszym. Tekst źródłowy programu z listingu 7.1 pokazuje szczegóły tych operacji.

### Listing 7.1. *shape/ShapeTest.java*

```
package shape;

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

/**
 * Program demonstrujący tworzenie figur za pomocą Java 2D.
 * @version 1.02 2007-08-16
 * @author Cay Horstmann
 */
public class ShapeTest
{
    public static void main(String[] args)
    {
        EventQueue.invokeLater(new Runnable()
        {
            public void run()
            {
                JFrame frame = new ShapeTestFrame();
                frame.setTitle("ShapeTest");
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.setVisible(true);
            }
        });
    }
}

/**
 * Ramka zawierająca listę rozwijalną wyboru figury
 * oraz panel, na którym rysowana jest jej reprezentacja.
 */
class ShapeTestFrame extends JFrame
{
    public ShapeTestFrame()
    {
        final ShapeComponent comp = new ShapeComponent();
        add(comp, BorderLayout.CENTER);
        final JComboBox<ShapeMaker> comboBox = new JComboBox<>();
        comboBox.addItem(new LineMaker());
        comboBox.addItem(new RectangleMaker());
    }
}
```

```

comboBox.addItem(new RoundedRectangleMaker());
comboBox.addItem(new EllipseMaker());
comboBox.addItem(new ArcMaker());
comboBox.addItem(new PolygonMaker());
comboBox.addItem(new QuadCurveMaker());
comboBox.addItem(new CubicCurveMaker());
comboBox.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent event)
    {
        ShapeMaker shapeMaker = comboBox.getItemAt(comboBox.getSelectedIndex());
        comp.setShapeMaker(shapeMaker);
    }
});
add(comboBox, BorderLayout.NORTH);
comp.setShapeMaker((ShapeMaker) comboBox.getItemAt(0));
pack();
}
}

/**
 * Komponent rysujący figurę
 * i umożliwiający przesuwanie definiujących ją punktów.
 */
class ShapeComponent extends JComponent
{
    private static final int DEFAULT_WIDTH = 300;
    private static final int DEFAULT_HEIGHT = 200;

    private Point2D[] points;
    private static Random generator = new Random();
    private static int SIZE = 10;
    private int current;
    private ShapeMaker shapeMaker;

    public ShapeComponent()
    {
        addMouseListener(new MouseAdapter()
        {
            public void mousePressed(MouseEvent event)
            {
                Point p = event.getPoint();
                for (int i = 0; i < points.length; i++)
                {
                    double x = points[i].getX() - SIZE / 2;
                    double y = points[i].getY() - SIZE / 2;
                    Rectangle2D r = new Rectangle2D.Double(x, y, SIZE, SIZE);
                    if (r.contains(p))
                    {
                        current = i;
                        return;
                    }
                }
            }
        });

        public void mouseReleased(MouseEvent event)
        {

```

```

        current = -1;
    }
    });
    addMouseListener(new MouseMotionAdapter()
    {
        public void mouseDragged(MouseEvent event)
        {
            if (current == -1) return;
            points[current] = event.getPoint();
            repaint();
        }
    });
    current = -1;
}

/**
 * Inicjuje obiekt ShapeMaker losowo wybranym zbiorem punktów kontrolnych.
 * @param aShapeMaker obiekt klasy ShapeMaker definiujący figurę
 * za pomocą zbioru punktów kontrolnych.
 */
public void setShapeMaker(ShapeMaker aShapeMaker)
{
    shapeMaker = aShapeMaker;
    int n = shapeMaker.getPointCount();
    points = new Point2D[n];
    for (int i = 0; i < n; i++)
    {
        double x = generator.nextDouble() * getWidth();
        double y = generator.nextDouble() * getHeight();
        points[i] = new Point2D.Double(x, y);
    }
    repaint();
}

public void paintComponent(Graphics g)
{
    if (points == null) return;
    Graphics2D g2 = (Graphics2D) g;
    for (int i = 0; i < points.length; i++)
    {
        double x = points[i].getX() - SIZE / 2;
        double y = points[i].getY() - SIZE / 2;
        g2.fill(new Rectangle2D.Double(x, y, SIZE, SIZE));
    }

    g2.draw(shapeMaker.makeShape(points));
}

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

/**
 * Klasa abstrakcyjna reprezentująca figurę
 * za pomocą zbioru punktów kontrolnych.
 * Jej konkretne klasy pochodne muszą implementować
 * metodę makeShape zwracającą dany rodzaj figury.
 */

```

```

abstract class ShapeMaker
{
    public abstract Shape makeShape(Point2D[] p);
    private int pointCount;

    /**
     * Tworzy obiekt klasy ShapeMaker.
     * @param aPointCount liczba punktów kontrolnych definiujących figurę.
     */
    public ShapeMaker(int aPointCount)
    {
        pointCount = aPointCount;
    }

    /**
     * Zwraca liczbę punktów kontrolnych definiujących figurę.
     * @return liczba punktów kontrolnych
     */
    public int getPointCount()
    {
        return pointCount;
    }

    /**
     * Tworzy figurę na podstawie zbioru punktów.
     * @param p punkty definiujące figurę
     * @return zdefiniowana figura
     */

    public String toString()
    {
        return getClass().getName();
    }
}

/**
 * Tworzy odcinek linii prostej łączący dwa punkty.
 */
class LineMaker extends ShapeMaker
{
    public LineMaker()
    {
        super(2);
    }

    public Shape makeShape(Point2D[] p)
    {
        return new Line2D.Double(p[0], p[1]);
    }
}

/**
 * Tworzy prostokąt rozpięty za pomocą dwu narożników.
 */
class RectangleMaker extends ShapeMaker
{
    public RectangleMaker()
    {

```

```

        super(2);
    }

    public Shape makeShape(Point2D[] p)
    {
        Rectangle2D s = new Rectangle2D.Double();
        s.setFrameFromDiagonal(p[0], p[1]);
        return s;
    }
}

/**
 * Tworzy zaokrąglony prostokąt rozpięty za pomocą dwu narożników.
 */
class RoundRectangleMaker extends ShapeMaker
{
    public RoundRectangleMaker()
    {
        super(2);
    }

    public Shape makeShape(Point2D[] p)
    {
        RoundRectangle2D s = new RoundRectangle2D.Double(0, 0, 0, 0, 20, 20);
        s.setFrameFromDiagonal(p[0], p[1]);
        return s;
    }
}

/**
 * Tworzy elipsę zawartą w prostokącie ograniczającym
 * rozpiętym za pomocą dwu narożników.
 */
class EllipseMaker extends ShapeMaker
{
    public EllipseMaker()
    {
        super(2);
    }

    public Shape makeShape(Point2D[] p)
    {
        Ellipse2D s = new Ellipse2D.Double();
        s.setFrameFromDiagonal(p[0], p[1]);
        return s;
    }
}

/**
 * Tworzy łuk zawarty w prostokącie ograniczającym
 * rozpiętym za pomocą dwu narożników. Dodatkowe dwa punkty
 * kontrolne umożliwiają określenie wartości początkowej i końcowej
 * kąta łuku. Aby zilustrować poprawność obliczeń kątów łuku, metoda makeShape
 * zwraca figurę złożoną z łuku, prostokąta ograniczającego i linii
 * łączących środek łuku z punktami kontrolnymi kąta.
 */
class ArcMaker extends ShapeMaker
{

```



```

public ArcMaker()
{
    super(4);
}

public Shape makeShape(Point2D[] p)
{
    double centerX = (p[0].getX() + p[1].getX()) / 2;
    double centerY = (p[0].getY() + p[1].getY()) / 2;
    double width = Math.abs(p[1].getX() - p[0].getX());
    double height = Math.abs(p[1].getY() - p[0].getY());

    double skewedStartAngle = Math.toDegrees(Math.atan2(-(p[2].getY() - centerY) *
↳width, (p[2].getX() - centerX) * height));
    double skewedEndAngle = Math.toDegrees(Math.atan2(-(p[3].getY() - centerY) *
↳width, (p[3].getX() - centerX) * height));
    double skewedAngleDifference = skewedEndAngle - skewedStartAngle;
    if (skewedStartAngle < 0) skewedStartAngle += 360;
    if (skewedAngleDifference < 0) skewedAngleDifference += 360;

    Arc2D s = new Arc2D.Double(0, 0, 0, 0, skewedStartAngle,
↳skewedAngleDifference, Arc2D.OPEN);
    s.setFrameFromDiagonal(p[0], p[1]);

    GeneralPath g = new GeneralPath();
    g.append(s, false);
    Rectangle2D r = new Rectangle2D.Double();
    r.setFrameFromDiagonal(p[0], p[1]);
    g.append(r, false);
    Point2D center = new Point2D.Double(centerX, centerY);
    g.append(new Line2D.Double(center, p[2]), false);
    g.append(new Line2D.Double(center, p[3]), false);
    return g;
}
}

/**
 * Tworzy wielokąt o sześciu wierzchołkach.
 */
class PolygonMaker extends ShapeMaker
{
    public PolygonMaker()
    {
        super(6);
    }

    public Shape makeShape(Point2D[] p)
    {
        GeneralPath s = new GeneralPath();
        s.moveTo((float) p[0].getX(), (float) p[0].getY());
        for (int i = 1; i < p.length; i++)
            s.lineTo((float) p[i].getX(), (float) p[i].getY());
        s.closePath();
        return s;
    }
}

```

```

/**
 * Tworzy krzywą drugiego stopnia zdefiniowaną przez jej końce
 * i pojedynczy punkt kontrolny.
 */
class QuadCurveMaker extends ShapeMaker
{
    public QuadCurveMaker()
    {
        super(3);
    }

    public Shape makeShape(Point2D[] p)
    {
        return new QuadCurve2D.Double(p[0].getX(), p[0].getY(), p[1].getX(),
        ↪p[1].getY(), p[2].getX(), p[2].getY());
    }
}

/**
 * Tworzy krzywą trzeciego stopnia zdefiniowaną przez jej końce
 * i dwa punkty kontrolne.
 */
class CubicCurveMaker extends ShapeMaker
{
    public CubicCurveMaker()
    {
        super(4);
    }

    public Shape makeShape(Point2D[] p)
    {
        return new CubicCurve2D.Double(p[0].getX(), p[0].getY(), p[1].getX(),
        ↪p[1].getY(), p[2].getX(), p[2].getY(), p[3].getX(), p[3].getY());
    }
}

```

W ogólności obiekt klasy `GeneralPath` nie musi reprezentować sekwencji połączonych ze sobą odcinków. Metodę `moveTo` możemy wywoływać dla niego dowolną ilość razy, rozpoczynając za każdym razem nowy fragment sekwencji.

Do sekwencji możemy także dodać obiekt dowolnej klasy implementującej interfejs `Shape`. Jego obrys zostanie dołączony na końcu sekwencji za pomocą metody `append`. Jej drugi parametr zadecyduje o tym, czy obrys powinien zostać połączony z ostatnim punktem sekwencji. Na przykład wywołanie

```

Rectangle2D r = . . . ;
path.append(r, false);

```

spowoduje dodanie obrysu prostokąta do sekwencji, ale bez połączenia. Natomiast wywołanie

```

path.append(r, true);

```

połączy dodatkowo za pomocą odcinka początkowy wierzchołek prostokąta z ostatnim punktem sekwencji.

Program, którego tekst źródłowy zawiera listing 7.1, pozwala tworzyć proste sekwencje na przykład, takie jak pokazano na rysunkach 7.7. i 7.8. Z listy rozwijalnej można wybrać następujące rodzaje tworzonych figur:

- odcinki linii prostych,
- prostokąty, zwykłe i zaokrąglone, elipsy,
- łuki (tworzone przez zmianę rozmiarów prostokąta ograniczającego oraz kąta początkowego i końcowego łuku),
- wielokąty (przy użyciu klasy `GeneralPath`),
- krzywe drugiego i trzeciego stopnia.

Punkty kontrolne wszystkich figur można zmieniać za pomocą myszy. Podczas ich przesuwania reprezentacja graficzna figury odrysowywana jest na bieżąco.

Kod programu jest dość skomplikowany, głównie za sprawą obsługi wielu rodzajów figur i przesuwania punktów kontrolnych.

Bazowa klasa abstrakcyjna `ShapeMaker` hermetyzuje wspólne właściwości klas umożliwiających tworzenie różnych rodzajów figur. Każda z tych klas definiuje określoną liczbę punktów kontrolnych, którymi może manipulować użytkownik. Metoda `getPointCount` zwraca liczbę tych punktów. Metoda abstrakcyjna

```
Shape makeShape(Point2D[] points)
```

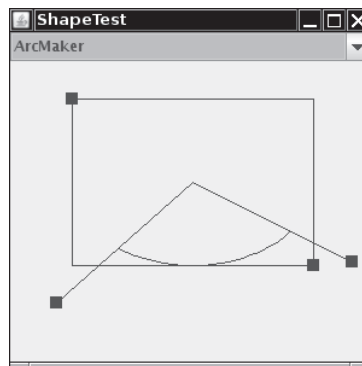
tworzy reprezentację figury, korzystając z aktualnego położenia punktów kontrolnych. Metoda `toString` zwraca po prostu nazwę klasy, umożliwiając w ten sposób wypełnienie listy rozwijalnej klasy `JComboBox` nazwami specjalizowanych obiektów `ShapeMaker`.

Aby umożliwić przesuwanie punktów kontrolnych, klasa `ShapePanel` musi obsługiwać zdarzenia związane zarówno z wyborem za pomocą myszy, jak i jej ruchem. Jeśli klawisz myszy przyciśnięty zostanie, gdy jej kursor znajduje się ponad prostokątem reprezentującym punkt kontrolny, to przesunięcie kursora spowoduje także przesunięcie punktu kontrolnego.

Implementacja większości klas umożliwiających tworzenie figur jest stosunkowo prosta. Ich metody `makeShape` tworzą figury na podstawie bieżącego położenia punktów kontrolnych. Jedynie klasa `ArcMaker` musi dodatkowo wyznaczać zniekształcone wartości początkowe i końcowe kąta łuku. Dodatkowo aby pokazać, że obliczenia te są prawidłowe, metoda `makeShape` klasy `ArcMaker` zwraca obiekt klasy `GeneralPath` złożony z łuku, ale i ograniczającego go prostokąta oraz linii łączących środek łuku z punktami kontrolnymi kątów (patrz rysunek 7.9).

### Rysunek 7.9.

Program `ShapeTest`  
w działaniu



**API** java.awt.geom.RoundRectangle2D.Double 1.2

- RoundRectangle2D.Double(double x, double y, double w, double h, double arcWidth, double arcHeight)

tworzy prostokąt o zaokrąglonych narożnikach o podanej pozycji i wymiarach. Objasnienie parametrów arcWidth i arcHeight znajdziesz na rysunku 7.4.

**API** java.awt.geom.Arc2D.Double 1.2

- Arc2D.Double(double x, double y, double w, double h, double startAngle, double arcAngle, int type)

tworzy łuk ograniczony prostokątem o określonych wartościach początkowej i kącie rozwarcia, a także typie zamknięcia. Parametry startAngle i arcAngle zostały omówione na rysunku 7.5. Parametr type może przyjmować jedną z wartości Arc2D.OPEN, Arc2D.PIE i Arc2D.CHORD.

**API** java.awt.geom.QuadCurve2D.Double 1.2

- QuadCurve2D.Double(double x1, double y1, double ctrlx, double ctrly, double x2, double y2)

tworzy krzywą drugiego stopnia na podstawie współrzędnych początku krzywej, punktu kontrolnego i końca krzywej.

**API** java.awt.geom.CubicCurve2D 1.2

- CubicCurve2D.Double(double x1, double y1, double ctrlx1, double ctrly1, double ctrlx2, double ctrly2, double x2, double y2)

tworzy krzywą trzeciego stopnia na podstawie współrzędnych początku krzywej, dwu punktów kontrolnych i końca krzywej.

**API** java.awt.geom.GeneralPath 1.2


- GeneralPath()

tworzy pustą sekwencję.

**API** java.awt.geom.Path2D.Float 6

- void moveTo(float x, float y)  
przyjmuje punkt (x, y) za *punkt bieżący* nowego segmentu.
- void lineTo(float x, float y)
- void quadTo(float ctrlx, float ctrly, float x, float y)
- void curveTo(float ctrlx1, float ctrly1, float ctrlx2, float ctrly2, float x, float y)

Rysują odcinek linii prostej, krzywej drugiego bądź trzeciego stopnia łączący bieżący punkt z punktem (x, y), który staje się nowym punktem bieżącym segmentu.

**API** java.awt.geom.Path2D 

- void append(Shape s, boolean connect)  
dodaje obrys figury do sekwencji. Jeśli connect posiada wartość true, to bieżący punkt sekwencji łączony jest za pomocą odcinka linii prostej z początkowym wierzchołkiem figury.
- void closePath()  
zamyka sekwencję wierzchołków, łącząc bieżący punkt z pierwszym z wierzchołków sekwencji.

## 7.3. Pola

W poprzednim podrozdziale pokazaliśmy, w jaki sposób można tworzyć skomplikowane figury, konstruując je z odcinków prostych i krzywych. Stosując odpowiednio dużą liczbę takich odcinków, możemy narysować dowolnie skomplikowaną figurę. Nawet czcionki, które widzimy na ekranie bądź wydruku, tworzone są jako kombinacje odcinków prostych i krzywych trzeciego stopnia.

Czasami prościej jednak opisać figurę jako kombinację *pól* ograniczonych prostokątami, wielokątami czy elipsami. Java 2D udostępnia cztery operacje *geometrii pól*, z których każda tworzy nowe pole jako kombinację dwu innych pól:

- add — utworzone pole zawiera wszystkie punkty, które należały do jednego z pól wyjściowych,
- subtract — utworzone pole zawiera wszystkie punkty, które należały do pierwszego pola, ale nie do drugiego,
- intersect — utworzone pole zawiera wszystkie punkty, które należały jednocześnie do obu pól,
- exclusiveOr — utworzone pole zawiera wszystkie punkty, które należały do jednego z pól wyjściowych, ale nie do obu naraz.

Rysunek 7.10 przedstawia wymienione operacje.

Tworzenie skomplikowanego pola rozpoczynamy od utworzenia domyślnego obiektu pola.

```
Area a = new Area();
```

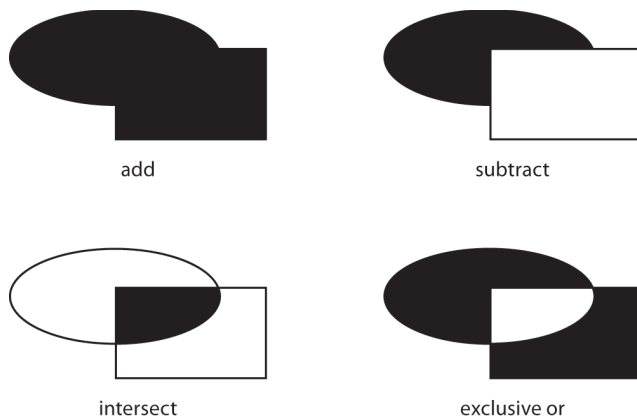
Następnie tworzymy jego kombinację z dowolnym innym polem:

```
a.add(new Rectangle2D.Double(. . .));
a.subtract(path);
. . .
```

Klasa Area implementuje interfejs Shape. Możemy więc narysować obrys pola, korzystając z metody draw klasy Graphics2D i wypełnić go za pomocą metody fill.

Rysunek 7.10.

Operacje na polach



**API** java.awt.geom.Area

- void add(Area other)
- void subtract(Area other)
- void intersect(Area other)
- void exclusiveOr(Area other)

Wykonują operacje geometrii pól na danym obiekcie oraz obiekcie będącym parametrem metody. Obiekt, na rzecz którego wywołano metody, staje się automatycznie wynikiem operacji.

## 7.4. Ślad pędzla

Metoda `draw` klasy `Graphics2D` rysuje obrys figury, korzystając z aktualnego *ślada pędzla*. Domyślnie ślad pędzla posiada postać ciągłej linii o szerokości jednego piksela. Możemy go zmienić, używając metody `setStroke`, której parametrem jest obiekt klasy implementującej interfejs `Stroke`. Java 2D definiuje tylko jedną taką klasę o nazwie `BasicStroke`. W tym podrozdziale przedstawimy jej możliwości.

Stosując klasę `BasicStroke`, możemy utworzyć ślad pędzla dowolnej szerokości. Poniższy przykład tworzy ślad pędzla o szerokości 10 pikseli.

```
g2.setStroke(new BasicStroke(10.0F));
g2.draw(new Line2D.Double(. . .));
```

Gdy ślad pędzla posiada szerokość większą niż jeden piksel, to dostępne są różne jego *zakończenia*. Rysunek 7.11 przedstawia następujące rodzaje zakończeń:

- *proste* — urywa ślad pędzla w punkcie końcowym,
- *zaokrąglone* — dodaje półkole za punktem końcowym,
- *kwadratowe* — dodaje półkwadrat za punktem końcowym.

**Rysunek 7.11.**

Rodzaje zakończeń śladu pędzla



Zakończenie proste

Zakończenie zaokrąglone

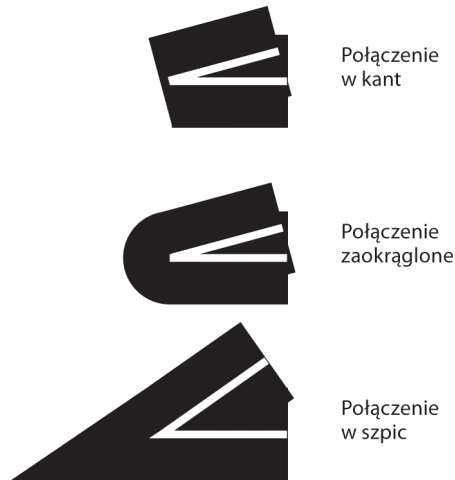
Zakończenie kwadratowe

Podobnie możemy określić też trzy rodzaje *połączeń* dwu śladów pędzla (patrz rysunek 7.12):

- *kant* — kończy połączenie śladów pędzla prostym odcinkiem prostopadłym do dwusiecznej kąta pomiędzy śladami,
- *okrągłe* — dodaje zaokrąglone połączenie śladów,
- *szpic* — przedłuża ślady, tak by tworzyły szpiczaste zakończenie.

**Rysunek 7.12.**

Rodzaje połączeń



Połączenie w kant

Połączenie zaokrąglone

Połączenie w szpic

Połączenie w szpic nie może być zastosowane w przypadku śladów łączących się pod niewielkim kątem. Zapobiega to tworzeniu wyjątkowo długich zakończeń połączeń. Jeśli kąt jest mniejszy od pewnej *wartości granicznej* kąta ustalonej dla połączenia w szpic, to automatycznie zastępowane jest ono kantem, co zapobiega powstawaniu zbyt długich „szpiców”. Domyślna wartość graniczna kąta wynosi 10 stopni.

Rodzaj zakończenia oraz połączenia możemy określić, tworząc obiekt klasy `BasicStroke`, na przykład jak poniżej:

```
g2.setStroke(new BasicStroke(10.0f, BasicStroke.CAP_ROUND,
    BasicStroke.JOIN_ROUND));
g2.setStroke(new BasicStroke(10.0f, BasicStroke.CAP_BUTT,
    BasicStroke.JOIN_MITER, 15.0f /* kąt graniczny połączenia w szpic */));
```

Możemy określić także dowolny *przerwany wzór* śladu pędzla. Program z listingu 7.2 umożliwia wybranie śladu pędzla odpowiadającego sygnałowi SOS w alfabecie Morse'a. Przerwany wzór śladu pędzla opisujemy za pomocą tablicy `float[]`, której kolejne pozycje określają na przemian długości odcinków ciągłych i przerw wzoru (patrz rysunek 7.13).

**Rysunek 7.13.**  
Przerwany wzór  
śladu pędzla



**Listing 7.2.** *stroke/StrokeTest.java*

```
package stroke;

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

/**
 * Program demonstrujący różne ślady pędzla.
 * @version 1.03 2007-08-16
 * @author Cay Horstmann
 */
public class StrokeTest
{
    public static void main(String[] args)
    {
        EventQueue.invokeLater(new Runnable()
        {
            public void run()
            {
                JFrame frame = new StrokeTestFrame();
                frame.setTitle("StrokeTest");
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.setVisible(true);
            }
        });
    }
}

/**
 * Ramka zawierająca przyciski wyboru parametrów śladu pędzla
 * i pokazująca efekt zastosowania śladu pędzla.
 */
class StrokeTestFrame extends JFrame
{
    private StrokeComponent canvas;
    private JPanel buttonPanel;

    public StrokeTestFrame()
    {
        canvas = new StrokeComponent();
        add(canvas, BorderLayout.CENTER);

        buttonPanel = new JPanel();
        buttonPanel.setLayout(new GridLayout(3, 3));
    }
}
```



```

add(buttonPanel, BorderLayout.NORTH);

ButtonGroup group1 = new ButtonGroup();
makeCapButton("Butt Cap", BasicStroke.CAP_BUTT, group1);
makeCapButton("Round Cap", BasicStroke.CAP_ROUND, group1);
makeCapButton("Square Cap", BasicStroke.CAP_SQUARE, group1);

ButtonGroup group2 = new ButtonGroup();
makeJoinButton("Miter Join", BasicStroke.JOIN_MITER, group2);
makeJoinButton("Bevel Join", BasicStroke.JOIN_BEVEL, group2);
makeJoinButton("Round Join", BasicStroke.JOIN_ROUND, group2);

ButtonGroup group3 = new ButtonGroup();
makeDashButton("Solid Line", false, group3);
makeDashButton("Dashed Line", true, group3);
}

/**
 * Tworzy przycisk wyboru zakończenia śladu pędzla.
 * @param label etykieta przycisku
 * @param style rodzaj zakończenia
 * @param group grupa przycisków wyboru
 */
private void makeCapButton(String label, final int style, ButtonGroup group)
{
    // wybiera pierwszy przycisk grupy
    boolean selected = group.getButtonCount() == 0;
    JRadioButton button = new JRadioButton(label, selected);
    buttonPanel.add(button);
    group.add(button);
    button.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent event)
        {
            canvas.setCap(style);
        }
    });
    pack();
}

/**
 * Tworzy przycisk wyboru połączenia śladów pędzla.
 * @param label etykieta przycisku
 * @param style rodzaj połączenia
 * @param group grupa przycisków wyboru
 */
private void makeJoinButton(String label, final int style, ButtonGroup group)
{
    // wybiera pierwszy przycisk grupy
    boolean selected = group.getButtonCount() == 0;
    JRadioButton button = new JRadioButton(label, selected);
    buttonPanel.add(button);
    group.add(button);
    button.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent event)
        {
            canvas.setJoin(style);
        }
    });
    pack();
}

```

```

    }
    });
}

/**
 * Tworzy przycisk wyboru wzoru śladu pędzla.
 * @param label etykieta przycisku
 * @param style false dla linii ciągłej, true dla przerywanej
 * @param group grupa przycisków wyboru
 */
private void makeDashButton(String label, final boolean style, ButtonGroup group)
{
    // wybiera pierwszy przycisk grupy
    boolean selected = group.getButtonCount() == 0;
    JRadioButton button = new JRadioButton(label, selected);
    buttonPanel.add(button);
    group.add(button);
    button.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent event)
        {
            canvas.setDash(style);
        }
    });
}

/**
 * Komponent rysujący dwa połączone odcinki
 * za pomocą różnych śladów pędzla i umożliwiający
 * użytkownikowi manipulację końcami odcinków.
 */
class StrokeComponent extends JComponent
{
    private static final int DEFAULT_WIDTH = 400;
    private static final int DEFAULT_HEIGHT = 400;
    private static int SIZE = 10;

    private Point2D[] points;
    private int current;
    private float width;
    private int cap;
    private int join;
    private boolean dash;

    public StrokeComponent()
    {
        addMouseListener(new MouseAdapter()
        {
            public void mousePressed(MouseEvent event)
            {
                Point p = event.getPoint();
                for (int i = 0; i < points.length; i++)
                {
                    double x = points[i].getX() - SIZE / 2;
                    double y = points[i].getY() - SIZE / 2;
                    Rectangle2D r = new Rectangle2D.Double(x, y, SIZE, SIZE);
                    if (r.contains(p))

```

```

        {
            current = i;
            return;
        }
    }
}

public void mouseReleased(MouseEvent event)
{
    current = -1;
}
}):

addMouseMotionListener(new MouseMotionAdapter()
{
    public void mouseDragged(MouseEvent event)
    {
        if (current == -1) return;
        points[current] = event.getPoint();
        repaint();
    }
}):

points = new Point2D[3];
points[0] = new Point2D.Double(200, 100);
points[1] = new Point2D.Double(100, 200);
points[2] = new Point2D.Double(200, 200);
current = -1;
width = 8.0F;
}

public void paintComponent(Graphics g)
{
    Graphics2D g2 = (Graphics2D) g;
    GeneralPath path = new GeneralPath();
    path.moveTo((float) points[0].getX(), (float) points[0].getY());
    for (int i = 1; i < points.length; i++)
        path.lineTo((float) points[i].getX(), (float) points[i].getY());
    BasicStroke stroke;
    if (dash)
    {
        float miterLimit = 10.0F;
        float[] dashPattern = { 10F, 10F, 10F, 10F, 10F, 10F, 30F, 10F, 30F, 10F,
            ↪30F, 10F, 10F, 10F, 10F, 10F, 30F };
        float dashPhase = 0;
        stroke = new BasicStroke(width, cap, join, miterLimit, dashPattern, dashPhase);
    }
    else stroke = new BasicStroke(width, cap, join);
    g2.setStroke(stroke);
    g2.draw(path);
}

/**
 * Określa rodzaj połączenia.
 * @param j rodzaj połączenia
 */
public void setJoin(int j)
{

```

```

        join = j;
        repaint();
    }

    /**
     * Określa rodzaj zakończenia.
     * @param c rodzaj zakończenia
     */
    public void setCap(int c)
    {
        cap = c;
        repaint();
    }

    /**
     * Określa rodzaj linii.
     * @param d false dla ciągłej, true dla przerywanej
     */
    public void setDash(boolean d)
    {
        dash = d;
        repaint();
    }

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

```

Tworząc obiekt klasy `BasicStroke`, możemy określić nie tylko wzór śladu pędzla, ale także jego *fazę*. Określa ona miejsce wzoru pędzla, od którego powinno rozpocząć się rysowanie linii. Najczęściej wartość ta wynosi 0 i oznacza rozpoczęcie rysowania od początku wzoru.

```

float[] dashpattern
    = { 10, 10, 10, 10, 10, 10, 30, 10, 30, ...};
g2.setStroke(new BasicStroke(10.0F, BasicStroke.CAP_BUTT,
    BasicStroke.JOIN_MITER, 15.0F /* kąt graniczny połączenia w szpic */,
    dashPattern, 0 /* faza */));

```

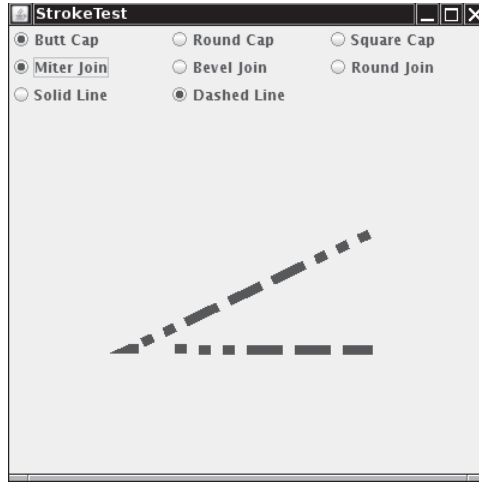


Wybrany rodzaj zakończenia używany jest także dla *poszczególnych odcinków* przerywanego wzoru pędzla.

Program z listingu 7.2 pozwala określać rodzaj zakończeń i połączeń oraz wybierać ciągły lub przerywany wzór pędzla (patrz rysunek 7.14). Dodatkowo pozwala on na zmianę położenia niepołączonych końców odcinków. Dzięki temu możliwe jest przetestowanie granicznej wartości kąta dla połączenia w szpic. Jeśli wybierzemy połączenie w szpic, a następnie będziemy zmniejszali kąt pomiędzy odcinkami, to po przekroczeniu wartości granicznej kąta połączenie w szpic zostanie automatycznie zastąpione kantem.

Program ten działa podobnie jak program z listingu 7.1. Obiekt nasłuchujący zdarzenia wyboru myszą zapamiętuje sytuacje wyboru końca odcinka, a obiekt nasłuchujący ruchu myszy monitoruje przesuwanie tego punktu. Zestaw przycisków wyboru służy określeniu rodzaju śladu pędzla. Metoda `paintComponent` klasy `StrokePanel` tworzy obiekt klasy `GeneralPath` złożony z trzech punktów połączonych dwoma odcinkami, którymi może manipulować użytkownik. Następnie kreuje obiekt klasy `BasicStroke` zgodnie z wyborem jego parametrów dokonany przez użytkownika i rysuje obiekt `GeneralPath`.

**Rysunek 7.14.**  
Program *StrokeTest*  
w działaniu



**API** java.awt.Graphics2D 1.2

- void setStroke(Stroke s)  
sprawia, że obiekt implementujący interfejs *Stroke* tworzyć będzie ślad pędzla dla danego kontekstu graficznego.

**API** java.awt.BasicStroke 1.2

- BasicStroke(float width)
- BasicStroke(float width, int cap, int join)
- BasicStroke(float width, int cap, int join, float miterLimit)
- BasicStroke(float width, int cap, int join, float miterLimit, float[] dash, float dashPhase)

Tworzą ślad pędzla o określonych atrybutach.

<i>Parametry:</i>	width	szerokość pędzla,
	cap	zakończenie śladu pędzla, jedna z wartości CAP_BUTT, CAP_ROUND i CAP_SQUARE,
	join	połączenie śladów pędzla, jedna z wartości JOIN_BEVEL, JOIN_MITER i JOIN_ROUND,
	miterLimit	wartość graniczna kąta wyrażona w stopniach, poniżej której połączenie w szpic jest automatycznie zastępowane kantem,
	dash	tablica długości występujących na przemian odcinków ciągłych i odstępów przerywanego śladu pędzla,
	dashPhase	„faza” wzoru przerywanego pędzla. Określa długość segmentu wzoru pędzla, która powinna zostać pominięta za każdym razem, gdy rozpoczynane jest rysowanie.

## 7.5. Wypełnienia

Wywołanie metody `fill` powoduje *wypełnienie* obszaru figury. Metoda `setPaint` umożliwia określenie rodzaju wypełnienia za pomocą obiektu implementującego interfejs `Paint`. Java 2D udostępnia trzy klasy implementujące ten interfejs:

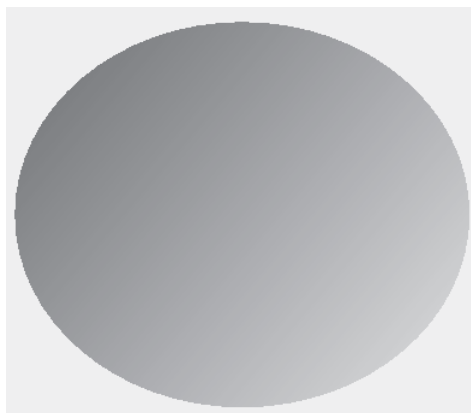
- `Color` — umożliwia wypełnienie figury pojedynczym kolorem, na przykład:

```
g2.setPaint(Color.RED);
```

- `GradientPaint` — zmienia stopniowo kolor wypełnienia pomiędzy dwiema określonymi wartościami (patrz rysunek 7.15),

### Rysunek 7.15.

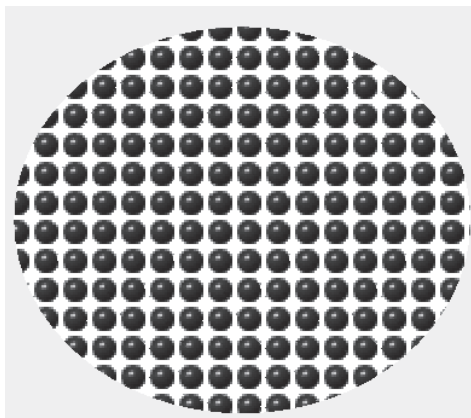
Przykład zastosowania klasy `GradientPaint`



- `TexturePaint` — wypełnia obszar figury, powtarzając zadany wzorec (patrz rysunek 7.16).

### Rysunek 7.16.

Przykład zastosowania klasy `TexturePaint`



Obiekt klasy `GradientPaint` tworzymy, określając kolor w dwu wybranych punktach obszaru figury.

```
g2.setPaint(new GradientPaint(p1, Color.RED, p2, Color.YELLOW));
```

Kolory są zmieniane wzdłuż linii łączącej te punkty, natomiast pozostają stałe wzdłuż prostopadłych. Punktom, które znajdują się poza punktami końcowymi odcinka, przyporządkowany zostaje taki sam kolor jak dla punktów końcowych.

Alternatywnie, możemy także wykorzystać inną wersję konstruktora, w której dodatkowy parametr `cyclic` określa, czy kolor zmieniać ma się także dla punktów leżących na zewnątrz odcinka.

```
g2.setPaint(new GradientPaint(p1, Color.RED, p2, Color.YELLOW, true));
```

Aby utworzyć obiekt klasy `TexturePaint`, musimy przekazać mu obiekt klasy `BufferedImage` zawierający obrazek wzorca wypełnienia i określić prostokąt *zakotwiczenia*.

```
g2.setPaint(new TexturePaint(bufferedImage, anchorRectangle));
```

Klasę `BufferedImage` omówimy dokładniej podczas przedstawiania operacji na obrazach. Najprościej możemy utworzyć obiekt klasy `BufferedImage` wczytując plik zawierający obrazek:

```
bufferedImage = ImageIO.read(new File("blue-ball.gif "));
```

Obrazek wzorca jest skalowany, tak by mieścił się w prostokącie zakotwiczenia, a prostokąt ten powielany jest następnie w kierunkach osi *x* i *y* tak, by pokryć obszar całej figury.

#### API | java.awt.Graphics2D 1.2

##### ■ void setPaint(Paint s)

sprawia, że obiekt implementujący interfejs `Paint` tworzyć będzie wypełnienie dla danego kontekstu graficznego.

#### API | java.awt.GradientPaint 1.2

##### ■ GradientPaint(float x1, float y1, Color color1, float x2, float y2, Color color2)

##### ■ GradientPaint(float x1, float y1, Color color1, float x2, float y2, Color color2, boolean cyclic)

##### ■ GradientPaint(Point2D p1, Color color1, Point2D p2, Color color2)

##### ■ GradientPaint(Point2D p1, Color color1, Point2D p2, Color color2, boolean cyclic)

Tworzą obiekt, który wypełnia obszar figury w taki sposób, że punkt początkowy posiada kolor `color1`, a punkt końcowy kolor `color2`. Punkty leżące na odcinku pomiędzy punktem początkowym i końcowym posiadają zmieniający się stopniowo kolor. Punkty leżące na prostopadłych do odcinka posiadają ten sam kolor. Domyślnie wzór zmiany koloru nie jest cykliczny. Oznacza to, że punkty leżące poza odcinkiem posiadają taki sam kolor jak punkt początkowy lub końcowy. Jeśli wzór zmiany koloru jest cykliczny, oznacza to, że powtarzany będzie także dla punktów leżących poza odcinkiem.

API java.awt.TexturePaint 1.2

- TexturePaint(BufferedImage texture, Rectangle2D anchor)

tworzy wypełnienie za pomocą wzorca. Prostokąt kotwiczący definiuje sposób wypełnienia obszaru figury wzorcem. Prostokąt ten jest powtarzany w kierunkach osi  $x$  i  $y$ , a wymiary wzorca są dopasowywane do jego wymiarów.

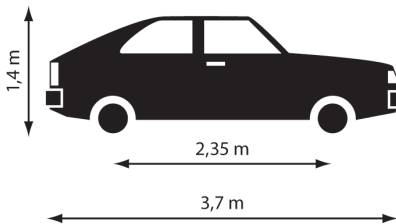
## 7.6. Przekształcenia układu współrzędnych

Załóżmy, że naszym zadaniem jest przedstawienie na rysunku pewnego modelu samochodu. Z jego specyfikacji technicznej znamy wymiary, takie jak wysokość, rozstaw osi i długość całkowita. Możemy oczywiście sami wyznaczyć rozmiary rysunku, przyjmując pewną liczbę pikseli na metr. Jest jednak łatwiejszy sposób. Wykonanie odpowiednich przekształceń możemy zlecić kontekstowi graficznemu.

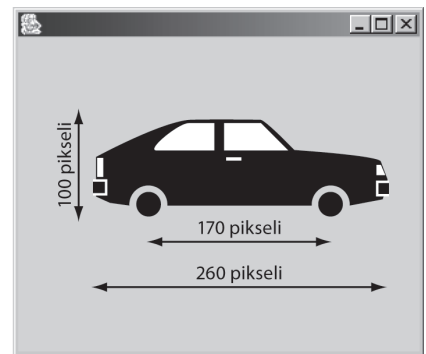
```
g2.scale(pixelsPerMeter, pixelsPerMeter);
g2.draw(new Line2D.Double(współrzędne wyrażone w metrach));
// przekształcenie metrów na piksele i narysowanie linii w odpowiedniej skali
```

Metoda `scale` klasy `Graphics2D` określa w tym przypadku *przekształcenie układu współrzędnych* polegające na jego przeskalowaniu. Przekształcenie to zamienia *współrzędne użytkownika* (w określonych przez niego jednostkach) na *współrzędne ekranowe* (piksele). Rysunek 7.17 przedstawia zasadę takiego przekształcenia.

**Rysunek 7.17.**  
Współrzędne użytkownika i współrzędne ekranowe



Współrzędne użytkownika



Współrzędne ekranowe

Przekształcenia układu współrzędnych okazują się bardzo przydatne w praktyce. Pozwalają pisać programy z wykorzystaniem najwygodniejszego w danym momencie układu współrzędnych. Przekształceniem do układu współrzędnych ekranowych zajmuje się kontekst graficzny.

Wyróżniamy cztery podstawowe rodzaje przekształceń układu współrzędnych:

- *skalowanie* polegające na wydłużeniu lub skróceniu wszystkich odległości mierzonych od pewnego punktu,
- *obrót* wszystkich punktów rysunku wokół środka obrotu,

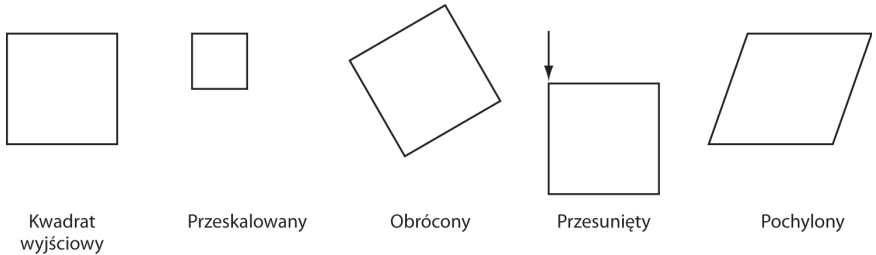


- *przesunięcie* wszystkich punktów o pewną odległość,
- *pochylenie* polegające na przesunięciu wszystkich linii rysunku równoległych do wybranej linii o wartość proporcjonalną do ich odległości od tej linii.

Rysunek 7.18 pokazuje efekt zastosowania tych przekształceń na przykładzie kwadratu.

**Rysunek 7.18.**

Podstawowe przekształcenia



Metody `scale`, `rotate`, `translate` i `shear` klasy `Graphics2D` pozwalają określić jeden z wymienionych rodzajów przekształcenia dla danego kontekstu graficznego.

Możliwe jest *składanie* przekształceń. Możemy na przykład obrócić daną figurę i następnie powiększyć ją dwukrotnie. W tym celu musimy określić oba przekształcenia dla kontekstu graficznego.

```
g2.rotate(angle);
g2.scale(2, 2);
g2.draw( . . . );
```

W powyższym przypadku kolejność określenia przekształceń nie ma znaczenia. Jednak w większości przypadków złożenia przekształceń kolejność jest istotna. Na przykład kiedy składamy obrót i pochylenie, uzyskany efekt zależy od tego, które z przekształceń zostanie wykonane jako pierwsze. Kontekst graficzny wykonuje przekształcenia w porządku odwrotnym do tego, w którym je określiliśmy. Tak więc ostatnie przekształcenie określone dla danego kontekstu wykonywane jest jako pierwsze.

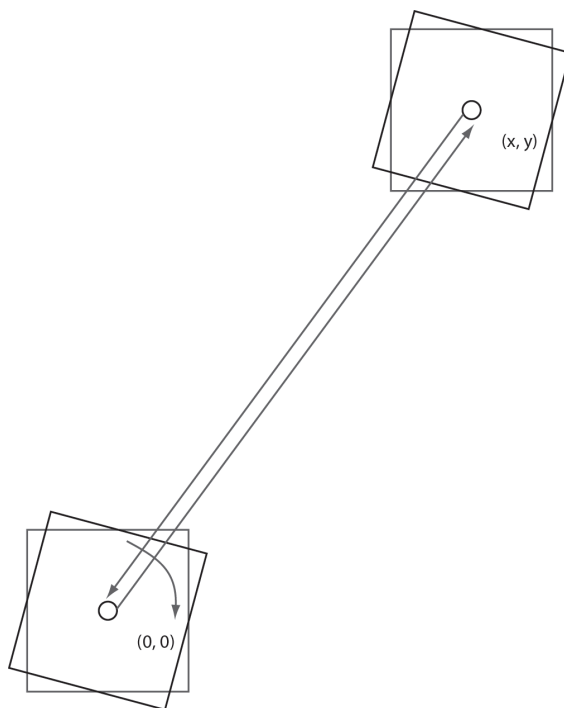
Dla danego kontekstu graficznego możemy określić dowolnie wiele przekształceń. Rozważmy na przykład poniższą sekwencję:

```
g2.translate(x, y);
g2.rotate(a);
g2.translate(-x, -y);
```

Ostatnie z przekształceń (wykonywane jako pierwsze) przesuwa punkt  $(x, y)$  do początku układu współrzędnych. Kolejne wykonuje obrót o kąt  $a$  dookoła początku układu współrzędnych. Ostatnie z przekształceń umieszcza początek układu współrzędnych na powrót w punkcie  $(x, y)$ . Efektem złożenia tych trzech przekształceń jest obrót dookoła punktu  $(x, y)$ , co możemy zobaczyć na rysunku 7.19. Ponieważ obroty względem punktu innego niż początek układu współrzędnych są często spotykanym rodzajem przekształcenia, to Java 2D udostępnia dla nich osobną metodę:

```
g2.rotate(a, x, y);
```

**Rysunek 7.19.**  
Składanie  
przekształceń



Wszystkie rodzaje przekształceń oraz ich złożenia opisane mogą być za pomocą działania na macierzach poniższej postaci:

$$\begin{bmatrix} x_{\text{nowe}} \\ y_{\text{nowe}} \\ 1 \end{bmatrix} = \begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Takie przekształcenie macierzy nazywane jest *przekształceniem afinicznym*. W Java 2D jest ono reprezentowane przez klasę `AffineTransform`. Jeśli znamy elementy macierzy przekształceń, to możemy utworzyć obiekt tej klasy w poniższy sposób:

```
AffineTransform t = new AffineTransform(a, b, c, d, e, f);
```

Dostępne są metody fabryki `getRotateInstance`, `getScaleInstance`, `getTranslateInstance` oraz `getShearInstance`, które tworzą macierze dla odpowiednich rodzajów przekształceń. Na przykład wywołanie:

```
t = AffineTransform.getScaleInstance(2.0F, 0.5F);
```

zwraca przekształcenie reprezentowane przez następującą macierz:

$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 0,5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Metody `setToRotation`, `setToScale`, `setToTranslation` i `setToShear` umożliwiają zmianę przekształcenia reprezentowanego przez obiekt, na przykład:

```
t.setToRotation(angle); //przekształcenie t będzie obrotem
```

Za pomocą obiektu klasy `AffineTransform` możemy także określić przekształcenie układu współrzędnych dla danego kontekstu graficznego.

```
g2.setTransform(t); //zastępuje dotychczasowe przekształcenie układu współrzędnych
```

W praktyce jednak metoda ta nie jest zalecana, ponieważ zastępuje także obszar przycięcia danego kontekstu graficznego. W praktyce może zdarzyć się, że na przykład kontekst graficzny dla drukowania na arkuszu o orientacji poziomej zawiera już odpowiedni obrót o 90 stopni. Jeśli wywołamy metodę `setTransform`, to zastąpimy ten obrót własnym przekształceniem. W takim przypadku lepiej więc skorzystać z metody `transform`, która złoży istniejące przekształcenie z reprezentowanym przez obiekt klasy `AffineTransform`.

```
g2.transform(t); //składa bieżące przekształcenie z reprezentowanym przez t
```

Jeśli chcemy zastosować pewne przekształcenie tylko jednorazowo, to powinniśmy najpierw zachować istniejące przekształcenie, następnie dokonać złożenia przekształceń, wykonać operacje graficzne i przywrócić wyjściowe przekształcenie.

```
AffineTransform oldTransform = g2.getTransform();
//przechowuje bieżące przekształcenie
g2.transform(t); //dodaje nowe przekształcenie
operacje graficzne z wykorzystaniem kontekstu g2
g2.setTransform(oldTransform); //przywraca wyjściowe przekształcenie
```

#### API java.awt.geom.AffineTransform 1.2

- `AffineTransform(double a, double b, double c, double d, double e, double f)`
- `AffineTransform(float a, float b, float c, float d, float e, float f)`

Tworzą macierz przekształcenia afinicznego postaci.

$$\begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix}$$

- `AffineTransform(double[] m)`
- `AffineTransform(float[] m)`

Tworzą macierz przekształcenia afinicznego postaci.

$$\begin{bmatrix} m[0] & m[2] & m[4] \\ m[1] & m[3] & m[5] \\ 0 & 0 & 1 \end{bmatrix}$$

- `static AffineTransform getRotateInstance(double a)`

tworzy obrót dookoła początku układu współrzędnych o kąt  $a$  (wyrażony w radianach). Macierz przekształcenia ma postać:

$$\begin{bmatrix} \cos(a) & -\sin(a) & 0 \\ \sin(a) & \cos(a) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Jeśli  $a$  jest z przedziału od 0 do  $\pi/2$ , to dodatnia część osi  $x$  zostaje obrócona w kierunku dodatniej części osi  $y$ .

- `static AffineTransform getRotateInstance(double a, double x, double y)`

tworzy obrót dookoła początku punktu  $(x, y)$  o kąt  $a$  (wyrażony w radianach).

- `static AffineTransform getScaleInstance(double sx, double sy)`

tworzy przekształcenie skalowania o współczynnik  $sx$  dla osi  $x$  i współczynnik  $sy$  dla osi  $y$ . Macierz przekształcenia ma postać:

$$\begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- `static AffineTransform getShearInstance(double shx, double shy)`

tworzy pochylenie o współczynnik  $shx$  dla osi  $x$  i współczynnik  $shy$  dla osi  $y$ . Macierz przekształcenia ma postać:

$$\begin{bmatrix} 1 & shx & 0 \\ shy & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- `static AffineTransform getTranslateInstance(double tx, double ty)`

tworzy przesunięcie o współczynnik na odległość  $tx$  w osi  $x$  i odległość  $ty$  w osi  $y$ . Macierz przekształcenia ma postać:

$$\begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix}$$

- `void setToRotation(double a)`
- `void setToRotation(double a, double x, double y)`
- `void setToScale(double sx, double sy)`
- `void setToShear(double shx, double shy)`
- `void setToTranslation(double tx, double ty)`

Określają rodzaj danego przekształcenia afinicznego za pomocą jednego z czterech podstawowych przekształceń. Opis przekształceń i ich parametrów zawiera opis metod `getXXXInstance`.

## API java.awt.Graphics2D 1.2

- void setTransform(AffineTransform t)  
zastępuje istniejące przekształcenie układu współrzędnych danego kontekstu graficznego za pomocą przekształcenia t.
- void transform(AffineTransform t)  
składa istniejące przekształcenie układu współrzędnych danego kontekstu graficznego z przekształceniem t.
- void rotate(double a)
- void rotate(double a, double x, double y)
- void scale (double sx, double sy)
- void shear (double shx, double shy)
- void translate(double tx, double ty)

Składają istniejące przekształcenie układu współrzędnych danego kontekstu graficznego z podstawowym przekształceniem o podanych parametrach. Opis przekształceń i ich parametrów zawiera opis metody `AffineTransform.getInstance()`.

## 7.7. Przycinanie

Przez określenie obszaru przycięcia ograniczamy wszystkie operacje graficzne do jego wnętrza.

```
g2.setClip(clipShape); // efekt dla metody poniżej
g2.draw(shape); // rysuje jedynie tę część figury, która mieści się w obszarze przycięcia
```

Zwykle jednak nie będziemy korzystać z metody `setClip`, ponieważ zastępuje ona istniejący już obszar przycięcia kontekstu graficznego, na przykład, jak pokażemy w dalszej części tego rozdziału, kontekst graficzny drukowania zawiera już obszar przycięcia uniemożliwiający drukowanie na marginesach. Dlatego też lepiej w takim przypadku skorzystać z metody `clip`.

```
g2.clip(clipShape); // lepiej
```

Metoda `clip` powoduje utworzenie nowego obszaru przycięcia, który jest częścią wspólną istniejącego dotąd obszaru i obszaru przekazanego jako jej parametr.

Jeśli chcemy wykorzystać dany obszar przycięcia jednorazowo, to powinniśmy zachować najpierw istniejący obszar przycięcia, następnie dodać nowy obszar, wykonać operacje graficzne kontekstu i odtworzyć wyjściowy obszar przycięcia:

```
Shape oldClip = g2.getClip(); // przechowuje obszar przycięcia
g2.clip(clipShape); // dodaje nowy obszar przycięcia
operacje graficzne kontekstu g2
g2.setClip(oldClip); // przywraca wyjściowy obszar przycięcia
```

Rysunek 7.20 demonstruje sposób przycięcia rysunku złożonego z wielu linii za pomocą złożonego obszaru wyznaczonego przez obrys znaków tekstu.

**Rysunek 7.20.**

Przycięcie za pomocą  
obrysu liter



Aby otrzymać obrys czcionek, musimy uzyskać najpierw *kontekst tworzenia czcionki*. Użyjemy w tym celu metody `getFontRenderContext` klasy `Graphics2D`.

```
FontRenderContext context = g2.getFontRenderContext();
```

Następnie, używając łańcucha znaków, czcionki i kontekstu tworzenia czcionki, tworzymy obiekt klasy `TextLayout`:

```
TextLayout layout = new TextLayout("Hello", font, context);
```

Obiekt ten opisuje wygląd ciągu znaków utworzonych za pomocą określonego kontekstu tworzenia czcionki. Wygląd ten zależy od zastosowanego kontekstu tworzenia czcionki i będzie różny w przypadku ekranu i drukarki.

Jego metoda `getOutline` zwraca obiekt implementujący interfejs `Shape` opisujący obrys czcionek tekstu. Obrys ten umieszczony jest w początku układu współrzędnych, co nie jest odpowiednie dla większości przypadków. Dlatego też operacji `getOutline` musimy dostarczyć odpowiednio zdefiniowane przekształcenie afiniczne, które spowoduje umieszczenie obrysu tekstu w żądanym miejscu. W naszym przypadku przekształceniem tym będzie przesunięcie do punktu `(0, 100)`.

```
AffineTransform transform
    = AffineTransform.getTranslateInstance(0, 100);
Shape outline = layout.getOutline(transform);
```

Uzyskany w ten sposób obrys tekstu dołączamy do obszaru przycięcia.

```
GeneralPath clipShape = new GeneralPath();
clipShape.append(outline, false);
```

Następnie określamy obszar przycięcia dla kontekstu graficznego i rysujemy zbiór linii, które widoczne są jedynie wewnątrz obszaru znaków tekstu.

```
g2.setClip(clipShape);
Point2D p = new Point2D.Double(0, 0);
for(int i = 0; i < NLINES; i++)
{
    double x = . . . ;
    double y = . . . ;
    Point2D q = new Point2D.Double(x, y);
    g2.draw(new Line2D.Double(p, q)); //linie są przycinane
}
```

**API** java.awt.Graphics 1.0

- void setClip(Shape s) 1.2  
sprawia, że obszar przycięcia określony jest przez obiekt s.
- Shape getClip() 1.2  
zwraca bieżący obszar przycięcia

**API** java.awt.Graphics2D 1.2

- void clip(Shape s)  
tworzy część wspólną bieżącego obszaru przycięcia z obszarem definiowanym przez s.
- FontRendererContext getFontRendererContext()  
zwraca kontekst tworzenia czcionki niezbędny do utworzenia obiektów klasy TextLayout.

**API** java.awt.font.TextLayout 1.2

- TextLayout(String s, Font f, FontRendererContext context)  
zwraca obiekt opisujący obrys łańcucha znaków s zapisanego czcionką f w kontekście graficznym context.
- float getAdvance()  
zwraca szerokość obrysu tekstu.
- float getAscent()
- float getDescent()  
Zwracają wysokość obrysu tekstu odpowiednio nad i pod linią bazową tekstu.
- float getLeading()  
zwraca odległość między kolejnymi liniami czcionki używanej do utworzenia obrysu tekstu.

## 7.8. Przezroczystość i składanie obrazów

W modelu kolorów RGB każdy kolor opisany jest przez wartość jego czerwonej, zielonej i niebieskiej składowej. Czasami przydatna okazuje się jednak możliwość określenia także stopnia *przejrzystości* obrazu. Gdy nakładamy obraz na już istniejący, to przejrzyste piksele nie zastępują zupełnie istniejących wcześniej pikseli, lecz tworzona jest ich kombinacja. Rysunek 7.21 pokazuje efekt nałożenia częściowo przezroczystego prostokąta na obrazek. Jak widać, nadal możemy rozróżnić szczegóły obrazka przesłonięte częściowo przez prostokąt.

**Rysunek 7.21.**

Nalóżenie częściowo  
przezroczystego  
prostokąta  
na obrazek



Java 2D opisuje przezroczystość za pomocą *wartości alfa*. Każdy piksel oprócz wartości składowej czerwonej, zielonej i niebieskiej jest scharakteryzowany także przez wartość alfa z przedziału od 0 (całkowita przezroczystość) do 1 (całkowite przesłanianie). Prostokąt na pokazany rysunku 7.21 został wypełniony kolorem żółtym o przejrzystości 50%:

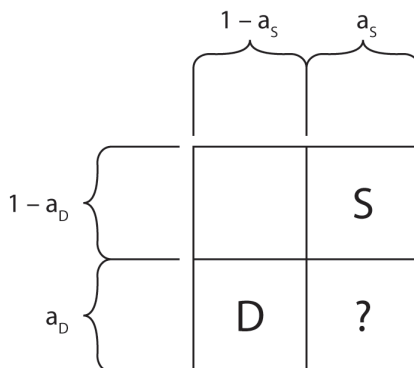
```
new Color(0.7F, 0.7F, 0.0F, 0.5F);
```

Podczas składania obrazów musimy utworzyć nie tylko kombinację składowych kolorów, ale także i wartości alfa. Dwóch badaczy z dziedziny grafiki komputerowej, Porter i Duff, podało dwanaście możliwych *reguł składania*. Java 2D implementuje wszystkie te reguły. Zanim przejdziemy do ich omówienia, należy zaznaczyć, że jedynie dwie z tych reguł posiadają znaczenie praktyczne. Jeśli reguły składania obrazów wydadzą się mało czytelne, to polecamy korzystanie jedynie z reguły SRC\_OVER. Jest ona domyślną regułą składania obrazów dla obiektów klasy Graphics2D i daje rezultaty zgodne z naszą intuicją.

Przejdźmy zatem do teorii reguł składania obrazów. Założymy w tym celu, że *piksel źródłowy* posiada wartość alfa równą  $a_S$ . Natomiast istniejący już obraz posiada *piksel docelowy* o wartości alfa równej  $a_D$ . Będziemy chcieli utworzyć kombinację obu pikseli. Sposób tworzenia reguły ich złożenia pokazuje diagram na rysunku 7.22.

**Rysunek 7.22.**

Projektowanie reguły  
składania obrazów



Porter i Duff przez wartość alfa rozumieją prawdopodobieństwo, że zostanie użyty kolor piksela. Z perspektywy piksela źródłowego prawdopodobieństwo to wynosi  $a_S$ , natomiast prawdopodobieństwo, że jego kolor nie zostanie użyty, wynosi  $1 - a_S$ . Analogicznie wartości te wyglądają dla piksela docelowego. Tworząc kombinację pikseli, musimy założyć, że są to prawdopodobieństwa zdarzeń niezależnych. W rezultacie otrzymujemy cztery przypadki pokazane na rysunku 7.22. Jeśli kolor piksela źródłowego ma być użyty, a docelowego nie, to oczywiście zastosujemy kolor piksela źródłowego. Dlatego też prawe górne pole diagramu oznaczyliśmy literą  $S$ . Prawdopodobieństwo tego zdarzenia wynosi  $a_S \cdot (1 - a_D)$ . Analogicznie lewe dolne pole oznaczyliśmy literą  $D$ . A co w przypadku, gdy każdy z pikseli będzie chciał



zastosować własny kolor? Właśnie tu znajdują zastosowanie reguły Portera-Duffa. Jeśli zdecydujemy, że piksel źródłowy jest „ważniejszy”, to także lewe dolne pole oznaczymy literą  $S$ . Na tym polega właśnie domyślna reguła SRC\_OVER. Tworzy ona kombinację koloru piksela źródłowego o wadze  $a_S$  z kolorem piksela docelowego o wadze  $(1-a_S)*a_D$ .

Efektym graficznym zastosowania tej reguły jest złożenie kolorów piksela źródłowego i docelowego z przewagą koloru źródłowego. W szczególności jeśli  $a_S$  wynosi 1, to kolor piksela źródłowego w ogóle nie jest brany pod uwagę. Natomiast w przypadku wartości 0 piksel źródłowy jest zupełnie przezroczysty i zachowany zostaje niezmienny kolor piksela docelowego.

Możliwe są inne reguły składania obrazów różniące się sposobem umieszczenia liter na diagramie. Tabela 7.1 oraz rysunek 7.23 prezentują reguły Portera-Duffa. Rysunek 7.23 pokazuje wyniki reguł złożenia źródłowego obszaru prostokąta o wartości alfa równej 0,75 z docelowym obszarem wycinka elipsy o wartości alfa równej 1.

**Tabela 7.1.** Zasady składania obrazów Portera-Duffa

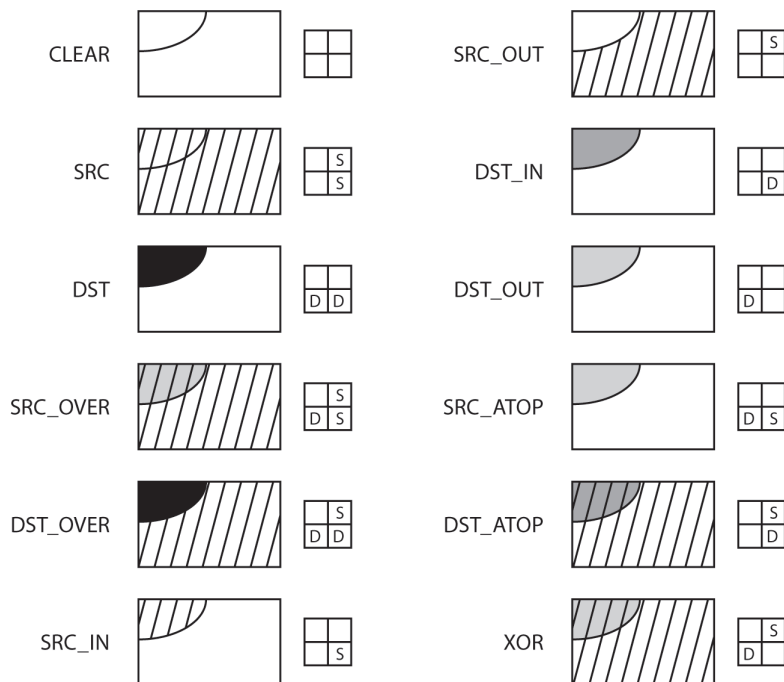
Zasada	Opis
CLEAR	Piksele obrazu źródłowego usuwają piksele obrazu docelowego.
SRC	Piksele obrazu źródłowego zastępują piksele obrazu docelowego oraz puste piksele.
DST	Piksele obrazu źródłowego nie mają wpływu na piksele obrazu docelowego.
SRC_OVER	Piksele obrazu źródłowego tworzą kombinację z pikselami obrazu docelowego i zastępują puste piksele.
DST_OVER	Piksele obrazu źródłowego nie mają wpływu na piksele obrazu docelowego, ale zastępują puste piksele.
SRC_IN	Piksele obrazu źródłowego zastępują piksele obrazu docelowego.
SRC_OUT	Piksele obrazu źródłowego usuwają piksele obrazu docelowego i zastępują puste piksele.
DST_IN	Wartość alfa pikseli obrazu źródłowego modyfikuje piksele obrazu docelowego.
DST_OUT	Uzupełnienie wartości alfa pikseli obrazu źródłowego modyfikuje piksele obrazu docelowego.
SRC_ATOP	Piksele obrazu źródłowego tworzą kombinację z pikselami obrazu docelowego.
DST_ATOP	Wartość alfa pikseli obrazu źródłowego modyfikuje piksele obrazu docelowego. Piksele obrazu źródłowego zastępują puste piksele.
XOR	Uzupełnienie wartości alfa pikseli obrazu źródłowego modyfikuje piksele obrazu docelowego. Piksele obrazu źródłowego zastępują puste piksele.

Jak łatwo zauważyć większość reguł jest mało przydatna. Skrajnym przypadkiem jest reguła DST\_IN. Nie bierze ona w ogóle pod uwagę koloru źródła, ale używa jego wartości alfa do modyfikacji obrazu docelowego. Natomiast reguła SRC jest przynajmniej potencjalnie przydatna — wymusza ona użycie koloru źródła, nie tworząc jego kombinacji z kolorem obrazu docelowego.

Więcej informacji o regułach Portera-Duffa odnajdziemy na przykład we wspomnianej już książce autorstwa Foley, van Dama, Feinera et al.

**Rysunek 7.23.**

Reguły składania obrazów  
Portera-Duffa



Metody `setComposite` klasy `Graphics2D` używamy w celu instalacji obiektu implementującego interfejs `Composite`. Java 2D dostarcza jedną klasę o nazwie `AlphaComposite`, która implementuje reguły Portera-Duffa przedstawione na rysunku 7.23.

Metoda fabryki `getInstance` klasy `AlphaComposite` udostępnia obiekt klasy `AlphaComposite` na podstawie podanej reguły i wartości alfa.

```
int rule = AlphaComposite.SRC_OVER;
float alpha = 0.5;
g2.setComposite(AlphaComposite.getInstance(rule, alpha));
g2.setPaint(Color.blue);
g2.fill(rectangle);
```

Powyższy fragment kodu rysuje prostokąt wypełniony kolorem niebieskim. Ponieważ wartość alfa wynosi dla niego 0,5, a regułą złożenia jest `SRC_OVER`, to nakrywa on półprzezroczyste już istniejący rysunek.

Program z listingu 7.3 umożliwia zapoznanie się w praktyce z działaniem reguł składania obrazów. Lista rozwijalna ułatwia wybór jednej z reguł złożenia, a suwak wartości alfa — nakładanego obrazu.

Dodatkowo program wyświetla krótkie objaśnienie sposobu działania wybranej reguły. Zwróćmy uwagę, że opisy te tworzone są na podstawie zawartości diagramu składania obrazów.

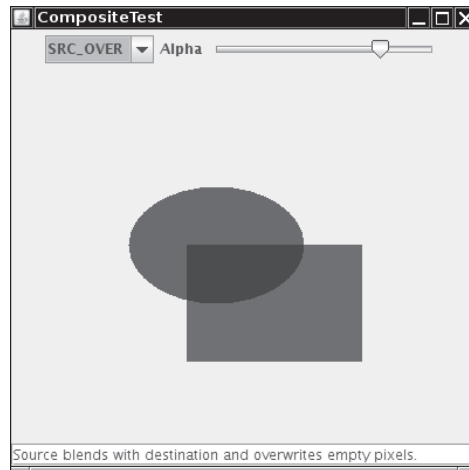
Z działaniem programu składającego obrazy wiąże się pewien problem. Nie ma żadnej gwarancji, że kontekst graficzny związany z ekranem wyposażony jest w kanał wartości alfa. (W rzeczywistości najczęściej nie jest). Gdy piksele umieszczane są bez wykorzystania kanału alfa, to wartości ich kolorów są przemnażane przez wartość alfa, która następnie jest pomi-

jana. Ponieważ kilka reguł Portera-Duffa korzysta jednak z wartości alfa obrazu docelowego, to istotne jest, by posiadał on kanał alfa. Z tego też powodu do składania kolorów wykorzystujemy obiekt klasy `BufferedImage` wyposażony w model kolorów ARGB. Dopiero po złożeniu obrazów rysujemy je na ekranie.

```
BufferedImage image = new BufferedImage(getWidth(),
    getHeight(), BufferedImage.TYPE_INT_ARGB);
Graphics2D gImage = image.createGraphics();
// teraz możemy narysować obraz
g2.drawImage(image, null, 0, 0);
```

Na listingach 7.3 i 7.4 przedstawiamy kod klasy ramki i klasy komponentu. Klasa `Rule` pokazana na listingu 7.5 dostarcza krótkiego objaśnienia dla każdej z reguł złożenia, co ilustruje rysunek 7.24. Po wybraniu reguły złożenia obrazów warto przesunąć suwak, aby zaobserwować wpływ wartości alfa. W szczególności warto też zwrócić uwagę, że jedyną różnicą między regułami `DST_IN` i `DST_OUT` jest sposób zmiany koloru obrazu docelowego (!) w odpowiedzi na zmianę wartości alfa obrazu źródłowego.

**Rysunek 7.24.**  
Program  
`CompositeTest`  
w działaniu



**Listing 7.3.** `composite/CompositeTestFrame.java`

```
package composite;

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

/**
 * Ramka zawierająca listę rozwijalną wyboru reguły,
 * suwak zmiany wartości źródłowej alfa
 * oraz komponent prezentujący efekt złożenia.
 */
class CompositeTestFrame extends JFrame
{
    private static final int DEFAULT_WIDTH = 400;
    private static final int DEFAULT_HEIGHT = 400;

    private CompositeComponent canvas;
```

```
private JComboBox<Rule> ruleCombo;
private JSlider alphaSlider;
private JTextField explanation;

public CompositeTestFrame()
{
    setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);

    canvas = new CompositeComponent();
    add(canvas, BorderLayout.CENTER);

    ruleCombo = new JComboBox<>(new Rule[] { new Rule("CLEAR", " ", " "),
        new Rule("SRC", " S", " S"), new Rule("DST", " ", "DD"),
        new Rule("SRC_OVER", " S", "DS"), new Rule("DST_OVER", " S", "DD"),
        new Rule("SRC_IN", " ", " S"), new Rule("SRC_OUT", " S", " "),
        new Rule("DST_IN", " ", " D"), new Rule("DST_OUT", " ", "D "),
        new Rule("SRC_ATOP", " ", "DS"), new Rule("DST_ATOP", " S", " D"),
        new Rule("XOR", " S", "D "), });
    ruleCombo.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent event)
        {
            Rule r = (Rule) ruleCombo.getSelectedItem();
            canvas.setRule(r.getValue());
            explanation.setText(r.getExplanation());
        }
    });

    alphaSlider = new JSlider(0, 100, 75);
    alphaSlider.addChangeListener(new ChangeListener()
    {
        public void stateChanged(ChangeEvent event)
        {
            canvas.setAlpha(alphaSlider.getValue());
        }
    });
    JPanel panel = new JPanel();
    panel.add(ruleCombo);
    panel.add(new JLabel("Alpha"));
    panel.add(alphaSlider);
    add(panel, BorderLayout.NORTH);

    explanation = new JTextField();
    add(explanation, BorderLayout.SOUTH);

    canvas.setAlpha(alphaSlider.getValue());
    Rule r = ruleCombo.getItemAt(ruleCombo.getSelectedIndex());
    canvas.setRule(r.getValue());
    explanation.setText(r.getExplanation());
}
}
```

---

**Listing 7.4.** *composite/CompositeComponent.java*

---

```
package composite;

import java.awt.*;
```

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

/**
 * Komponent prezentujący złożenie dwu figur.
 */
class CompositeComponent extends JComponent
{
    private int rule;
    private Shape shape1;
    private Shape shape2;
    private float alpha;

    public CompositeComponent()
    {
        shape1 = new Ellipse2D.Double(100, 100, 150, 100);
        shape2 = new Rectangle2D.Double(150, 150, 150, 100);
    }

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

        BufferedImage image = new BufferedImage(getWidth(), getHeight(),
        ↳BufferedImage.TYPE_INT_ARGB);
        Graphics2D gImage = image.createGraphics();
        gImage.setPaint(Color.red);
        gImage.fill(shape1);
        AlphaComposite composite = AlphaComposite.getInstance(rule, alpha);
        gImage.setComposite(composite);
        gImage.setPaint(Color.blue);
        gImage.fill(shape2);
        g2.drawImage(image, null, 0, 0);
    }

    /**
     * Określa regułę złożenia.
     * @param r reguła (jako stała AlphaComposite)
     */
    public void setRule(int r)
    {
        rule = r;
        repaint();
    }

    /**
     * Określa wartość alfa dla źródła
     * @param a wartość alfa z przedziału od 0 do 100
     */
    public void setAlpha(int a)
    {
        alpha = (float) a / 100.0F;
        repaint();
    }
}
```

Listing 7.5. *composite/Rule.java*

```

package composite;

import java.awt.*;

/**
 * Klasa reprezentująca regułę Portera-Duffa.
 */
class Rule
{
    private String name;
    private String porterDuff1;
    private String porterDuff2;

    /**
     * Tworzy obiekt reprezentujący regułę Portera-Duffa
     * @param n nazwa reguły
     * @param pd1 pierwszy wiersz diagramu Portera-Duffa
     * @param pd2 drugi wiersz diagramu Portera-Duffa
     */
    public Rule(String n, String pd1, String pd2)
    {
        name = n;
        porterDuff1 = pd1;
        porterDuff2 = pd2;
    }

    /**
     * Zwraca objaśnienie sposobu działania reguły.
     * @return objaśnienie
     */
    public String getExplanation()
    {
        StringBuilder r = new StringBuilder("Source ");
        if (porterDuff2.equals(" ")) r.append("clears");
        if (porterDuff2.equals(" S")) r.append("overwrites");
        if (porterDuff2.equals("DS")) r.append("blends with");
        if (porterDuff2.equals(" D")) r.append("alpha modifies");
        if (porterDuff2.equals("D ")) r.append("alpha complement modifies");
        if (porterDuff2.equals("DD")) r.append("does not affect");
        r.append(" destination");
        if (porterDuff1.equals(" S")) r.append(" and overwrites empty pixels");
        r.append(".");
        return r.toString();
    }

    public String toString()
    {
        return name;
    }

    /**
     * Zwraca wartość wyznaczoną przez regułę
     * jako obiekt klasy AlphaComposite
     * @return stała AlphaComposite lub -1, jeśli nie istnieje odpowiednia stała.
     */
    public int getValue()
    {

```

```

    try
    {
        return (Integer) AlphaComposite.class.getField(name).get(null);
    }
    catch (Exception e)
    {
        return -1;
    }
}
}

```

**API** java.awt.Graphics2D 1.2

- void setComposite(Composite s)

sprawia, że sposób składania obrazów dla danego kontekstu graficznego określony jest przez obiekt s implementujący interfejs Composite.

**API** java.awt.AlphaComposite 1.2

- static AlphaComposite getInstance(int rule)
- static AlphaComposite getInstance(int rule, float alpha)

Tworzą obiekt klasy AlphaComposite. Parametr rule przyjmuje jedną z wartości CLEAR, SRC, SRC\_OVER, DST\_OVER, SRC\_IN, SRC\_OUT, DST\_IN, DST\_OUT, DST, DST\_ATOP, SRC\_ATOP lub XOR.

## 7.9. Wskazówki operacji graficznych

Na podstawie lektury poprzednich podrozdziałów można przekonać się, że proces tworzenia obrazów może być bardzo złożony. Choć Java 2D okazuje się w większości przypadków zaskakująco efektywna, to jednak zdarzają się sytuacje, w których przydatna jest możliwość określenia kompromisu między szybkością operacji graficznych a jakością otrzymanych obrazów. Java 2D udostępnia w tym celu *wskazówki operacji graficznych*. Metoda setRenderingHint klasy Graphics2D umożliwia określenie pojedynczej wskazówki. Klucze i wartości dostępnych wskazówek zadeklarowane są w klasie RenderingHints. Tabela 7.2 prezentuje wszystkie możliwości. Wartości, których nazwa kończy się przyrostkiem \_DEFAULT, oznaczają wartości domyślne wybrane przez określoną implementację jako najlepszy kompromis pomiędzy szybkością działania i jakością obrazu.

Jedną z bardziej przydatnych wskazówek związana jest z techniką *antialiasingu*. Technika ta służy wygładzeniu pochyłych linii prostych oraz krzywych. Jak pokazano na rysunku 7.25, krzywa rysowana jest za pomocą pikseli układających się w „schodki”. Zwłaszcza w przypadku prezentacji za pomocą urządzeń o niskiej rozdzielczości daje to niezbyt elegancki efekt. Okazuje się, że reprezentację krzywej można wygładzić, rysując dodatkowo piksele krzywej o „częściowym” wypełnieniu. Efekt ten uzyskujemy, nadając pikselom wartość koloru proporcjonalną do stopnia ich pokrycia przez krzywą. Jednak ceną, jaką musimy zapłacić za wygładzenie krzywej, są oczywiście obliczenia związane z wyznaczeniem kolorów dodatkowych pikseli.

Tabela 7.2. Wskazówki operacji graficznych

Klucz	Wartości	Objaśnienie
KEY_ANTIALIASING	VALUE_ANTIALIAS_ON VALUE_ANTIALIAS_OFF VALUE_ANTIALIAS_DEFAULT	Włącza (wyłącza) antialiasing dla rysowania figur.
KEY_TEXT_ANTIALIASING	VALUE_TEXT_ANTIALIAS_ON VALUE_TEXT_ANTIALIAS_OFF VALUE_TEXT_ANTIALIAS_DEFAULT VALUE_TEXT_ANTIALIAS_GASP 6 VALUE_TEXT_ANTIALIAS_LCD_HRGB 6 VALUE_TEXT_ANTIALIAS_LCD_HRGR 6 VALUE_TEXT_ANTIALIAS_LCD_VRGB 6 VALUE_TEXT_ANTIALIAS_LCD_VBGR 6	Włącza (wyłącza) antialiasing dla czcionek. W przypadku wartości VALUE_TEXT_ANTIALIAS_GASP decyzja o zastosowaniu antialiasingu dla określonego rozmiaru czcionki jest podejmowana na podstawie tabeli opisującej rasteryzację czcionki.
KEY_FRACTIONAL_METRICS	VALUE_FRACTIONALMETRICS_ON VALUE_FRACTIONALMETRICS_OFF VALUE_FRACTIONALMETRICS_DEFAULT	Zwiększa lub zmniejsza dokładność obliczeń rozmiarów czcionek.
KEY_RENDERING	VALUE_RENDER_QUALITY VALUE_RENDER_SPEED VALUE_RENDER_DEFAULT	Wybiera algorytmy rysowania o większej efektywności lub dokładności (jeśli są dostępne).
KEY_STROKE_CONTROL 1.3	VALUE_STROKE_NORMALIZE VALUE_STROKE_PURE VALUE_STROKE_DEFAULT	Wybiera tworzenia śladów pędzla kontrolowane przez akcelerator graficzny (który może przesuwać ślad pędzla o nie więcej niż pół piksela) lub przez „czystą” regułę dopuszczającą jedynie przebieg śladu przez „środek” pikseli.
KEY_DITHERING	VALUE_DITHER_ENABLE VALUE_DITHER_DISABLE VALUE_DITHER_DEFAULT	Włącza (wyłącza) symulowanie wartości kolorów za pomocą grup pikseli o różnych kolorach. (Sposób działania może zależeć od tego, czy stosowany jest również antialiasing).
KEY_ALPHA_INTERPOLATION	VALUE_ALPHA_INTERPOLATION_QUALITY VALUE_ALPHA_INTERPOLATION_SPEED VALUE_ALPHA_INTERPOLATION_DEFAULT	Włącza (wyłącza) precyzyjne obliczenia wartości alfa.
KEY_COLOR_RENDERING	VALUE_COLOR_RENDER_QUALITY VALUE_COLOR_RENDER_SPEED VALUE_COLOR_RENDER_DEFAULT	Zwiększa dokładność lub efektywność wyznaczania kolorów.
KEY_INTERPOLATION	VALUE_INTERPOLATION_NEAREST_NEIGHBOR VALUE_INTERPOLATION_BILINEAR VALUE_INTERPOLATION_BICUBIC	Wybiera regułę interpolacji pikseli podczas skalowania obrazów.

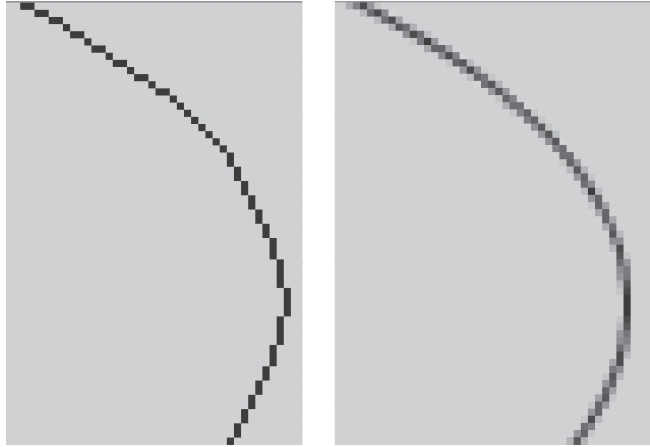
Na przykład wykorzystania antialiasingu podczas rysowania obrysów figur zażądamy w następujący sposób:

```
g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
    RenderingHints.VALUE_ANTIALIAS_ON);
```

Technika antialiasingu jest także przydatna w przypadku rysowania czcionek:



**Rysunek 7.25.**  
Antialiasing



```
g2.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING,
    RenderingHints.VALUE_TEXT_ANTIALIAS_ON);
```

Pozostałe wskazówki operacji graficznych używane są rzadziej.

Zbiór par klucz-wartość wskazówek możemy także umieścić na mapie i wyegzekwować je wszystkie za pomocą pojedynczego wywołania metody `setRenderHints`. Wykorzystać możemy w tym celu dowolną klasę kolekcji implementującą interfejs mapy, ale także i samą klasę `RenderingHints`. Implementuje ona także interfejs `Map` i dostarcza domyślnej implementacji mapy, jeśli jej konstruktorowi prześlemy wartość `null`, na przykład:

```
RenderingHints hints = new RenderingHints(null);
hints.put(RenderingHints.KEY_ANTIALIASING,
    RenderingHints.VALUE_ANTIALIAS_ON);
hints.put(RenderingHints.KEY_TEXT_ANTIALIASING,
    RenderingHints.VALUE_TEXT_ANTIALIAS_ON);
g2.setRenderHints(hints);
```

Technikę tę wykorzystujemy w programie z listingu 7.6. Tworzy on obrazek, korzystając na początku z wartości wskazówek, które uznaliśmy za potencjalnie przydatne w tym przypadku. Zauważmy, że:

- Antialiasing wygładza elipsę.
- Antialiasing wygładza tekst.
- Na niektórych platformach zastosowanie dokładniejszych obliczeń dla czcionek (`VALUE_FRACTIONALMETRICS_ON`) powoduje ich nieco ciaśniejsze wyświetlanie.
- Zastosowanie wskazówki `VALUE_RENDER_QUALITY` wygładza przeskalowany obrazek (ten sam efekt uzyskamy, stosując wartość `VALUE_INTERPOLATION_BICUBIC` dla wskazówki `KEY_INTERPOLATION`).
- Jeśli wyłączymy antialiasing, to wybór wartości `VALUE_STROKE_NORMALIZE` spowoduje zmianę wyglądu elipsy oraz zmianę położenia przekątnej kwadratu.

Listing 7.6. *renderQuality/RenderQualityTestFrame.java*

```

package renderQuality;

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

/**
 * Ramka zawierająca przyciski wyboru wskazówek
 * oraz obrazek tworzony przy ich zastosowaniu.
 */
public class RenderQualityTestFrame extends JFrame
{
    private RenderQualityComponent canvas;
    private JPanel buttonBox;
    private RenderingHints hints;
    private int r;

    public RenderQualityTestFrame()
    {
        buttonBox = new JPanel();
        buttonBox.setLayout(new GridBagLayout());
        hints = new RenderingHints(null);

        makeButtons("KEY_ANTIALIASING", "VALUE_ANTIALIAS_OFF", "VALUE_ANTIALIAS_ON");
        makeButtons("KEY_TEXT_ANTIALIASING", "VALUE_TEXT_ANTIALIAS_OFF",
        ↪ "VALUE_TEXT_ANTIALIAS_ON");
        makeButtons("KEY_FRACTIONALMETRICS", "VALUE_FRACTIONALMETRICS_OFF",
        "VALUE_FRACTIONALMETRICS_ON");
        makeButtons("KEY_RENDERING", "VALUE_RENDER_SPEED", "VALUE_RENDER_QUALITY");
        makeButtons("KEY_STROKE_CONTROL", "VALUE_STROKE_PURE", "VALUE_STROKE_NORMALIZE");
        canvas = new RenderQualityComponent();
        canvas.setRenderingHints(hints);

        add(canvas, BorderLayout.CENTER);
        add(buttonBox, BorderLayout.NORTH);
        pack();
    }

    /**
     * Tworzy zestaw przycisków wyboru wskazówek
     * @param key nazwa klucza
     * @param value1 nazwa pierwszej wartości dla klucza
     * @param value2 nazwa drugiej wartości dla klucza
     */
    void makeButtons(String key, String value1, String value2)
    {
        try
        {
            final RenderingHints.Key k = (RenderingHints.Key)
            ↪ RenderingHints.class.getField(key).get(
                null);
            final Object v1 = RenderingHints.class.getField(value1).get(null);
            final Object v2 = RenderingHints.class.getField(value2).get(null);
            JLabel label = new JLabel(key);

```

```

        buttonBox.add(label, new GBC(0, r).setAnchor(GBC.WEST));
        ButtonGroup group = new ButtonGroup();
        JRadioButton b1 = new JRadioButton(value1, true);

        buttonBox.add(b1, new GBC(1, r).setAnchor(GBC.WEST));
        group.add(b1);
        b1.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                hints.put(k, v1);
                canvas.setRenderingHints(hints);
            }
        });
        JRadioButton b2 = new JRadioButton(value2, false);

        buttonBox.add(b2, new GBC(2, r).setAnchor(GBC.WEST));
        group.add(b2);
        b2.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                hints.put(k, v2);
                canvas.setRenderingHints(hints);
            }
        });
        hints.put(k, v1);
        r++;
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}

/**
 * Komponent tworzący rysunek z zastosowaniem wskazówek.
 */
class RenderQualityComponent extends JComponent
{
    private static final int DEFAULT_WIDTH = 750;
    private static final int DEFAULT_HEIGHT = 150;

    private RenderingHints hints = new RenderingHints(null);
    private Image image;

    public RenderQualityComponent()
    {
        image = new ImageIcon(getClass().getResource("face.gif")).getImage();
    }

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

        g2.draw(new Ellipse2D.Double(10, 10, 60, 50));
    }
}

```

```

g2.setFont(new Font("Serif", Font.ITALIC, 40));
g2.drawString("Hello", 75, 50);

g2.draw(new Rectangle2D.Double(200, 10, 40, 40));
g2.draw(new Line2D.Double(201, 11, 239, 49));

g2.drawImage(image, 250, 10, 100, 100, null);
}

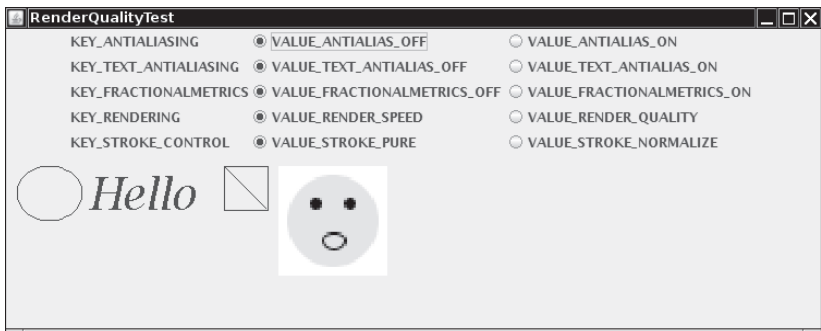
/**
 * Konfiguruje wskazówki i powoduje odrysowanie obrazka.
 * @param h wskazówki
 */
public void setRenderingHints(RenderingHints h)
{
    hints = h;
    repaint();
}

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

```

Na rysunku 7.26 przedstawiony został efekt działania programu.

**Rysunek 7.26.**  
Program  
RenderingHints  
w działaniu



**API** java.awt.Graphics2D **1.2**

- void setRenderingHint(RenderingHints.Key key, Object value)  
stosuje wskazówkę do danego kontekstu graficznego.
- void setRenderingHints(Map m)  
stosuje do danego kontekstu graficznego wszystkie wskazówki, których pary klucz-wartość przechowuje mapa.

**API** java.awt.RenderingHints **1.2**

- RenderingHints(Map<RenderingHints.Key, ?> m)  
tworzy mapę wskazówek operacji graficznych. Jeśli m posiada wartość null, to wykorzystywana jest domyślna implementacja mapy.

## 7.10. Czytanie i zapisywanie plików graficznych

W wersjach wcześniejszych niż 1.4, Java SE posiadała bardzo ograniczone możliwości odczytu i zapisu plików graficznych. Możliwy był odczyt plików zapisanych w formatach GIF oraz JPEG, ale zapis obrazów nie był możliwy w żadnym z formatów.

Sytuacja ta uległa poprawie z wprowadzeniem Java SE 1.4. Pakiet `javax.imageio` dostarcza gotowej implementacji operacji odczytu i zapisu plików graficznych w kilku powszechnie stosowanych formatach, a także szkielet umożliwiający tworzenie własnych implementacji dla innych formatów plików graficznych. W Java SE 6 możliwy jest odczyt i zapis plików zapisanych w formatach GIF, JPEG, PNG, BMP (Windows bitmap) i WBMP (wireless bitmap). Brak możliwości zapisu plików w formacie GIF we wcześniejszych wersjach był spowodowany ograniczeniami patentowymi.

Sposób korzystania z podstawowych operacji na plikach graficznych jest bardzo prosty. Aby załadować zawartość pliku graficznego, korzystamy z metody statycznej `read` klasy `ImageIO`:

```
File f = . . . ;
BufferedImage image = ImageIO.read(f);
```

Klasa `ImageIO` sama dobiera odpowiedni obiekt do odczytu danego formatu na podstawie rozszerzenia pliku oraz identyfikatora formatu zapisanego na początku pliku. Jeśli obiekt odczytu dla danego formatu jest niedostępny, to metoda `read` zwraca wartość `null`.

Równie łatwo można zapisać obraz w pliku:

```
File f = . . . ;
String format = . . . ;
ImageIO.write(image, format, f);
```

Parametr `format` metody `write` identyfikuje docelowy format pliku za pomocą łańcucha znaków postaci "JPEG" lub "PNG". Na tej podstawie klasa `ImageIO` dobiera odpowiedni obiekt zapisu pliku.

### 7.10.1. Wykorzystanie obiektów zapisu i odczytu plików graficznych

W przypadku bardziej zaawansowanych operacji na plikach graficznych metody statyczne `read` i `write` klasy `ImageIO` mogą okazać się niewystarczające. W takich sytuacjach musimy najpierw uzyskać odpowiedni dla danego formatu obiekt odczytu `ImageReader` lub obiekt zapisu `ImageWriter`. Klasa `ImageIO` tworzy wyliczenie dostępnych obiektów na podstawie jednej z poniższych informacji:

- formatu pliku (na przykład "JPEG"),
- rozszerzenia nazwy pliku (na przykład ".jpg"),
- typu określonego w standardzie MIME (na przykład "image/jpeg").



Standard MIME (*Multipurpose Internet Mail Extensions*) definiuje formaty wymiany danych, na przykład "image/jpeg" czy "application/pdf". Wersję dokumentu RFC w formacie HTML definiującego standard MIME można odnaleźć pod adresem <http://www.oac.uci.edu/indiv/ehood/MIME>.

Obiekt odczytu pliku w formacie JPEG możemy uzyskać w poniższy sposób:

```
ImageReader reader = null;
Iterator<ImageReader> iter =
    ImageIO.getImageReadersByFormatName("JPEG");
if (iter.hasNext()) reader = iter.next();
```

Można też wykorzystać w tym celu metody `getImageReadersBySuffix` lub `getImageReadersByMIMEType`, które tworzą wyliczenie na podstawie odpowiednio rozszerzenia pliku lub typu MIME.

Może zdarzyć się, że klasa `ImageIO` zlokalizuje więcej niż jeden obiekt umożliwiający odczyt danego formatu pliku. Należy wtedy wybrać jeden z nich. Aby uzyskać więcej informacji o dostępnych obiektach odczytu, musimy utworzyć dla nich *interfejs dostawcy*:

```
ImageReaderSpi spi = reader.getOriginatingProvider();
```

Korzystając z niego, możemy pobrać informację o nazwie producenta (dostawcy) i numerze wersji:

```
String vendor = spi.getVendor();
String version = spi.getVersion();
```

Informacja ta może okazać się pomocna w wyborze odpowiedniego obiektu lub zostać wykorzystana do utworzenia listy, która pozwoli wybrać obiekt użytkownikowi programu. Dla dalszych rozważań przyjmijmy, że korzystając będziemy z pierwszego ze znalezionych obiektów.

Program zamieszczony na listingu 7.7 znajduje wszystkie rozszerzenia plików dla wszystkich dostępnych obiektów odczytu plików graficznych. Rozszerzenia te wykorzystuje następnie do utworzenia filtra nazw plików. W tym celu używamy metody statycznej `ImageIO.getReaderFileSuffixes`:

```
String[] extensions = ImageIO.getWriterFileSuffixes();
chooser.setFileFilter(new FileNameExtensionFilter("Image files", extensions));
```

---

**Listing 7.7.** *imageIO/ImageIOFrame.java*

---

```
package imageIO;

import java.awt.event.*;
import java.awt.image.*;
import java.io.*;
import java.util.*;
import javax.imageio.*;
import javax.imageio.stream.*;
import javax.swing.*;
import javax.swing.filechooser.*;

/**
 * Ramka wyświetlająca zawartość pliku graficznego.
 * Jej menu umożliwia odczyt i zapis plików.
 */
public class ImageIOFrame extends JFrame
{
    private static final int DEFAULT_WIDTH = 400;
    private static final int DEFAULT_HEIGHT = 400;
```

```

private static Set<String> writerFormats = getWriterFormats();

private BufferedImage[] images;

public ImageIOFrame()
{
    setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);

    JMenu fileMenu = new JMenu("File");
    JMenuItem openItem = new JMenuItem("Open");
    openItem.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent event)
        {
            openFile();
        }
    });
    fileMenu.add(openItem);

    JMenu saveMenu = new JMenu("Save");
    fileMenu.add(saveMenu);
    Iterator<String> iter = writerFormats.iterator();
    while (iter.hasNext())
    {
        final String formatName = iter.next();
        JMenuItem formatItem = new JMenuItem(formatName);
        saveMenu.add(formatItem);
        formatItem.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                saveFile(formatName);
            }
        });
    }

    JMenuItem exitItem = new JMenuItem("Exit");
    exitItem.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent event)
        {
            System.exit(0);
        }
    });
    fileMenu.add(exitItem);

    JMenuBar menuBar = new JMenuBar();
    menuBar.add(fileMenu);
    setJMenuBar(menuBar);
}

/**
 * Otwiera plik i ładuje obrazek.
 */
public void openFile()
{

```

```

JFileChooser chooser = new JFileChooser();
chooser.setCurrentDirectory(new File("."));
String[] extensions = ImageIO.getReaderFileSuffixes();
chooser.setFileFilter(new FileNameExtensionFilter("Image files", extensions));
int r = chooser.showOpenDialog(this);
if (r != JFileChooser.APPROVE_OPTION) return;
File f = chooser.getSelectedFile();
Box box = Box.createVerticalBox();
try
{
    String name = f.getName();
    String suffix = name.substring(name.lastIndexOf('.') + 1);
    Iterator<ImageReader> iter = ImageIO.getImageReadersBySuffix(suffix);
    ImageReader reader = iter.next();
    ImageInputStream imageIn = ImageIO.createImageInputStream(f);
    reader.setInput(imageIn);
    int count = reader.getNumImages(true);
    images = new BufferedImage[count];
    for (int i = 0; i < count; i++)
    {
        images[i] = reader.read(i);
        box.add(new JLabel(new ImageIcon(images[i])));
    }
}
catch (IOException e)
{
    JOptionPane.showMessageDialog(this, e);
}
setContentPane(new JScrollPane(box));
validate();
}

/**
 * Zapisuje bieżący obrazek w pliku.
 * @param formatName format pliku
 */
public void saveFile(final String formatName)
{
    if (images == null) return;
    Iterator<ImageWriter> iter = ImageIO.getImageWritersByFormatName(formatName);
    ImageWriter writer = iter.next();
    JFileChooser chooser = new JFileChooser();
    chooser.setCurrentDirectory(new File("."));
    String[] extensions = writer.getOriginatingProvider().getFileSuffixes();
    chooser.setFileFilter(new FileNameExtensionFilter("Image files", extensions));

    int r = chooser.showSaveDialog(this);
    if (r != JFileChooser.APPROVE_OPTION) return;
    File f = chooser.getSelectedFile();
    try
    {
        ImageOutputStream imageOut = ImageIO.createImageOutputStream(f);
        writer.setOutput(imageOut);

        writer.write(new IIOMImage(images[0], null, null));
        for (int i = 1; i < images.length; i++)
        {

```



```

        IIImage iioImage = new IIImage(images[i], null, null);
        if (writer.canInsertImage(i)) writer.writeInsert(i, iioImage, null);
    }
}
catch (IOException e)
{
    JOptionPane.showMessageDialog(this, e);
}
}

/**
 * Tworzy zbiór "preferowanych" nazw formatów plików graficznych,
 * dla których istnieją obiekty odczytu. Preferowaną nazwą formatu
 * zostaje pierwsza nazwa dla danego obiektu odczytu.
 * @return zbiór nazw formatów
 */
public static Set<String> getWriterFormats()
{
    Set<String> writerFormats = new TreeSet<>();
    Set<String> formatNames = new TreeSet<>(Arrays.asList(ImageIO
        .getWriterFormatNames()));
    while (formatNames.size() > 0)
    {
        String name = formatNames.iterator().next();
        Iterator<ImageWriter> iter = ImageIO.getImageWritersByFormatName(name);
        ImageWriter writer = iter.next();
        String[] names = writer.getOriginatingProvider().getFormatNames();
        String format = names[0];
        if (format.equals(format.toLowerCase())) format = format.toUpperCase();
        writerFormats.add(format);
        formatNames.removeAll(Arrays.asList(names));
    }
    return writerFormats;
}
}
}

```

Zapis plików wymaga trochę więcej pracy. Chcemy zaprezentować użytkownikowi menu formatów graficznych do wyboru, ale, niestety, metoda `getWriterFormatNames` klasy `ImageIO` tworzy mało przydatną listę powtarzających się nazw:

```
jpg, BMP, bmp, JPG, jpeg, wbmp, png, JPEG, PNG, WBMP, GIF, gif
```

Na pewno menu wyboru nie powinno wyglądać w ten sposób. Musimy raczej stworzyć listę „preferowanych” nazw formatów. W tym celu tworzymy metodę pomocniczą `getWriterFormatNames` (patrz listing 7.7). Przeglądając listę, możemy pobrać pierwszą nazwę formatu i wyszukać z nią obiekt zapisu. Następnie pobrać dla niego nazwę formatu w nadziei, że będzie to najbardziej popularna wersja nazwy. Dla obiektu zapisu formatu JPEG rozwiązanie takie sprawdza się: otrzymujemy nazwę "JPEG". (Jednak już dla obiektu zapisu formatu PNG jako pierwszą uzyskamy nazwę "png". Mamy nadzieję, że sytuacja ta zostanie uporządkowana w kolejnej wersji biblioteki. Tymczasem rozwiązujemy problem, zamieniając wszystkie nazwy na duże litery). Po wybraniu preferowanej nazwy usuwamy wszystkie alternatywne nazwy z listy wyjściowej. W ten sposób postępujemy dla wszystkich formatów.

## 7.10.2. Odczyt i zapis plików zawierających sekwencje obrazów

Niektóre pliki, w szczególności pliki animacji w formacie GIF, mogą zawierać wiele obrazów. Metoda `read` klasy `ImageIO` umożliwia odczyt pojedynczego obrazu. Aby załadować wiele obrazów, musimy przekształcić źródło wejścia (na przykład strumień wejściowy lub plik) w obiekt klasy `ImageInputStream`.

```
InputStream in = . . . ;
ImageInputStream imageIn = ImageIO.createImageInputStream(in);
```

A następnie dołączyć uzyskany obiekt do obiektu odczytu pliku:

```
reader.setInput(imageIn, true);
```

Drugi z parametrów metody `setInput` wskazuje, że źródło umożliwiać będzie tylko odczyt w trybie „do przodu”. W przeciwnym razie możliwy będzie swobodny dostęp do danych źródła albo przez buforowanie ich w czasie odczytu, albo za pomocą pliku o dostępie swobodnym. Swobodny dostęp do pliku graficznego bywa przydatny w pewnych sytuacjach. Na przykład jeśli chcemy uzyskać informacje o liczbie obrazów w pliku GIF, musimy przeczytać cały plik. Jeśli zechcemy następnie pobrać jeden z obrazów, to plik nie będzie musiał być odczytany ponownie.

Powyższa uwaga jest istotna jedynie w przypadku odczytu za pośrednictwem strumienia, pliku, który zawiera wiele obrazów i jego format nie udostępnia w nagłówku pliku informacji o liczbie zapisanych obrazów. Natomiast kiedy czytamy obrazy bezpośrednio z pliku, wystarczy nam poniższy fragment kodu:

```
File f = . . . ;
ImageInputStream imageIn = ImageIO.createImageInputStream(f);
reader.setInput(imageIn);
```

Dysponując obiektem odczytu pliku, możemy wczytać określony obraz za pomocą poniższego wywołania:

```
BufferedImage image = reader.read(index);
```

gdzie `index` jest numerem obrazu (numeracja rozpoczyna się od zera).

Jeśli źródło obrazów jest w trybie odczytu „do przodu”, to metodę `read` wywołujemy tak długo, aż nie wyrzuci ona wyjątku `IndexOutOfBoundsException`. W przeciwnym razie możemy skończyć z metody `getNumImages`:

```
int n = reader.getNumImages(true);
```

Jej parametr zezwala na przeszukanie zawartości pliku w celu ustalenia liczby obrazów. Jeśli ich źródło jest jednak w trybie odczytu „do przodu”, to metoda ta wyrzuci wyjątek `IllegalStateException`. Jeśli podając metodzie parametr o wartości `false`, zabronimy przeszukiwania pliku, to zwróci ona wartość `-1`, w przypadku gdy ustalenie liczby obrazów bez przeszukania pliku nie jest możliwe. W takiej sytuacji musimy odczytywać kolejne obrazy aż do wyrzucenia wyjątku `IndexOutOfBoundsException`.

Nagłówki niektórych plików graficznych mogą zawierać miniatury umożliwiające podgląd ich zawartości. Liczbę miniatur obrazu o danym indeksie możemy określić, wywołując poniższą metodę:

```
int count = reader.getNumThumbnails(index);
```

Określoną miniaturę danego obrazu wczytujemy, tak jak poniżej:

```
BufferedImage thumbnail = reader.readThumbnail(index,
    thumbnailIndex);
```

Często przydatna jest możliwość uzyskania informacji o rozmiarze obrazu — na przykład zanim zostanie załadowany za pośrednictwem sieci o niskiej przepustowości. Następujące wywołania

```
int width = reader.getWidth(index);
int height = reader.getHeight(index);
```

umożliwiają uzyskanie informacji o rozmiarach obrazu o danym indeksie.

Aby zapisać w pliku sekwencję obrazów, musimy najpierw uzyskać odpowiedni obiekt klasy `ImageWriter`. Klasa `ImageIO` umożliwia wyliczenie obiektów zapisu dla określonego formatu graficznego:

```
String format = . . . ;
ImageWriter writer = null;
Iterator<ImageWriter> iter =
    ImageIO.getImageWritersByFormatName(format);
if (iter.hasNext()) writer = iter.next();
```

Następnie musimy przekształcić strumień wyjścia lub plik w obiekt klasy `ImageOutputStream`, który udostępniemy obiektowi zapisu. Możemy to zrobić w sposób przedstawiony poniżej:

```
File f = . . . ;
ImageOutputStream imageOut = ImageIO.createImageOutputStream(f);
writer.setOutput(imageOut);
```

Każdy z zapisywanych obrazów musimy obudować jeszcze za pomocą obiektu klasy `IIIOImage`. Opcjonalnie możemy dostarczyć także listę miniatur oraz metadane obrazów (opisujące na przykład algorytm kompresji czy sposób kodowania kolorów). W naszym przypadku użyjemy wartości `null` dla obu opcjonalnych parametrów. Więcej informacji na temat ich wykorzystania znajdziemy w dokumentacji Java 2D.

```
IIIOImage iioImage = new IIIOImage(images[i], null, null);
```

Pierwszy z obrazów zapiszemy, korzystając z metody `write`:

```
writer.write(new IIIOImage(images[0], null, null));
```

Natomiast kolejne w następujący sposób:

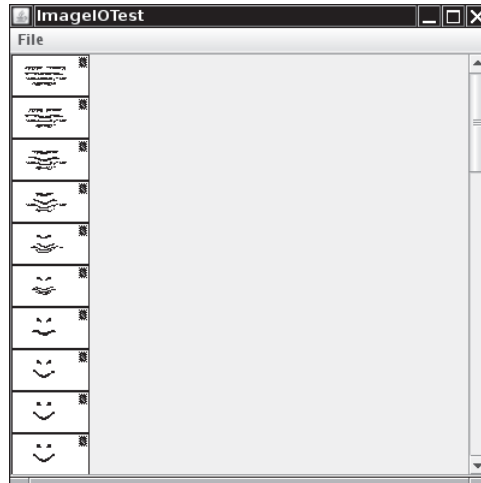
```
if (writer.canInsertImage(i))
    writer.writeInsert(i, iioImage, null);
```

Trzeci z parametrów metody `writeInsert` może być obiektem klasy `ImageWriteParam` opisującym szczegóły zapisu obrazu, takie jak na przykład sposób kompresji. Jeśli chcemy skorzystać z domyślnych sposobów zapisu obrazów, to podajemy po prostu wartość `null`.

Nie wszystkie formaty graficzne umożliwiają zapis wielu obrazów w jednym pliku. W takim przypadku metoda `canInsertImage` zwróci dla wartości indeksu większych od zera wartość `false`.

Program, którego tekst źródłowy zawiera listing 7.7, umożliwia odczyt i zapis plików graficznych w formatach, dla których biblioteka języka Java dostarcza implementacji obiektów odczytu i zapisu. Program wyświetla wszystkie obrazy zawarte w pliku (patrz rysunek 7.27), ale nie wyświetla miniatur.

**Rysunek 7.27.**  
Plik animacji  
w formacie GIF



**API** javax.imageio.ImageIO 1.4

- static BufferedImage read(File input)
- static BufferedImage read(InputStream input)
- static BufferedImage read(URL input)

Czytają obraz ze źródła input.

- static boolean write(RenderedImage image, String formatName, File output)
- static boolean write(RenderedImage image, String formatName, OutputStream output)

Zapisują obraz w określonym formacie do wyjścia output. Zwraca wartość false, jeśli nie został znaleziony obiekt zapisu odpowiedni dla danego formatu.

- static Iterator<ImageReader> getImageReadersByFormatName(String formatName)
- static Iterator<ImageReader> getImageReadersBySuffix(String fileSuffix)
- static Iterator<ImageReader> getImageReadersByMIMEType(String mimeType)
- static Iterator<ImageWriter> getImageWritersByFormatName(String formatName)
- static Iterator<ImageWriter> getImageWritersBySuffix(String fileSuffix)
- static Iterator<ImageWriter> getImageWritersByMIMEType(String mimeType)

Tworzą wyliczenie wszystkich obiektów odczytu lub zapisu odpowiednich dla danego formatu (określonego za pomocą nazwy — na przykład "JPEG", rozszerzenia nazwy pliku — na przykład ".jpg" bądź typu zgodnego ze standardem MIME — na przykład "image/jpeg").

- `static String[] getReaderFormatNames()`
- `static String[] getReaderMIMETypes()`
- `static String[] getWriterFormatNames()`
- `static String[] getWriterMIMETypes()`
- `static String[] getReaderFileSuffixes()` 6
- `static String[] getWriterFileSuffixes()` 6

Zwracają tablicę nazw formatów, nazw typów MIME lub rozszerzeń nazw, dla których istnieją obiekty odczytu lub zapisu.

- `InputStream createInputStream(Object input)`
- `OutputStream createOutputStream(Object output)`

Przekształca podany obiekt na obiekt klasy `InputStream` lub `OutputStream`. Parametrem może być plik, strumień, obiekt klasy `RandomAccessFile` lub inny obiekt, dla którego istnieje dostawca usługi. Zwraca wartość `null`, jeśli żaden z zarejestrowanych dostawców usług nie potrafi dokonać konwersji.

#### API | `javax.imageio.ImageReader` 1.4

- `void setInput(Object input)`
- `void setInput(Object input, boolean seekForwardOnly)`

Określają źródło danych dla obiektu odczytu.

*Parametry:* `input`                      obiekt klasy `InputStream` lub inny obiekt akceptowany przez obiekt odczytu,  
                   `seekForwardOnly`      wartość `true` oznacza, że obiekt może czytać jedynie dane „do przodu”. Domyślnie obiekt odczytu wykorzystuje swobodny dostęp do danych, które buforuje, jeśli zachodzi taka potrzeba.

- `BufferedImage read(int index)`  
 wczytuje obraz o danym indeksie (wartości indeksu rozpoczynają się od 0). Wyrzuca wyjątek `IndexOutOfBoundsException`, jeśli obraz taki nie jest dostępny.
- `int getNumImages(boolean allowSearch)`  
 pobiera liczbę obrazów dostępnych za pomocą danego obiektu odczytu. Jeśli parametr `allowSearch` ma wartość `false`, a liczba obrazów nie może być ustalona bez wcześniejszego odczytania danych, to metoda zwraca wartość `-1`. Jeśli parametr `allowSearch` ma wartość `true`, a obiekt odczytu znajduje się w trybie odczytu „do przodu”, to wyrzucany jest wyjątek `IllegalStateException`.
- `int getNumThumbnails(int index)`  
 zwraca liczbę miniatur obrazu o podanym indeksie.
- `BufferedImage readThumbnail(int index, int thumbnailIndex)`  
 zwraca miniaturę o indeksie `thumbnailIndex` dla obrazu o indeksie `index`.

- `int getWidth(int index)`
- `int getHeight(int index)`

Zwracają szerokość i wysokość obrazu. Wyrzucają wyjątek `IndexOutOfBoundsException`, jeśli obraz taki nie jest dostępny.

- `ImageReaderSpi getOriginatingProvider()`  
zwraca dostawcę usługi dla danego obiektu odczytu.

#### API `javax.imageio.spi.IIOServiceProvider` 1.4

- `String getVendorName()`
- `String getVersion()`

Zwracają nazwę producenta i wersję danego dostawcy usługi.

#### API `javax.imageio.spi.ImageReaderWriterSpi` 1.4

- `String[] getFormatNames()`
- `String[] getFileSuffixes()`
- `String[] getMIMETypes()`

Zwracają nazwy formatów, rozszerzenia nazw plików lub typy MIME obsługiwane przez obiekty odczytu i zapisu danego dostawcy usługi.

#### API `javax.imageio.ImageWriter` 1.4

- `void setOutput(Object output)`  
określa cel danych dla obiektu zapisu.  
*Parametry:* `output`      obiekt klasy `ImageOutputStream` lub inny obiekt akceptowany przez obiekt zapisu.
- `void write(IIImage image)`
- `void write(RenderedImage image)`  
Zapisują pojedynczy obraz.
- `void writeInsert(int index, IIImage image, ImageWriteParam param)`  
zapisuje obraz do pliku zawierającego sekwencję obrazów.  
*Parametry:* `index`      indeks obrazu,  
`image`      zapisywany obraz,  
`param`      parametry zapisu lub wartość `null`.
- `boolean canInsertImage(int index)`  
zwraca wartość `true`, jeśli obraz może zostać zapisany na pozycji określonej wartością indeksu.

- `ImageWriterSpi` `getOriginatingProvider()`  
zwraca dostawcę usługi dla danego obiektu zapisu.

**API** `javax.imageio.IIOImage` 1.4

- `IIOImage(RenderedImage image, List thumbnails, IIOMetadata metadata)`  
obudowuje obraz, jego miniatury i metadane za pomocą obiektu klasy `IIOImage`.  
*Parametry:* `image` obraz,  
`thumbnails` lista obiektów klasy `BufferedImage` lub wartość `null`,  
`metadata` metadane opisujące obraz lub wartość `null`.

## 7.11. Operacje na obrazach

W praktyce często pojawia się potrzeba dostosowania wczytanego z pliku obrazu do własnych potrzeb przez modyfikację części jego pikseli, a nawet utworzenia własnego obrazu od podstaw — na przykład w celu prezentacji wyników pomiarów lub obliczeń. Klasa `BufferedImage` umożliwia pełną kontrolę nad pikselami obrazu, a klasy implementujące interfejs `BufferedImageOp` pozwalają wykonywać operacje na obrazach.



JDK 1.0 dysponowało zupełnie innym, znacznie bardziej skomplikowanym szkieletem przetwarzania obrazów, zoptymalizowanym pod kątem *przyrostowego tworzenia obrazów* ładowanych z serwerów internetowych linia po linii i prezentowanych użytkownikowi stopniowo w miarę napływania kolejnych danych. Wykorzystanie tego rozwiązania było w praktyce dość trudne. W tej książce nie będziemy zajmować się wspomnianym szkieletem.

### 7.11.1. Dostęp do danych obrazu

W większości przypadków obrazy, na których operują nasze programy, wczytywane są z pliku — zapisanego przez cyfrowy aparat fotograficzny, skaner bądź program graficzny. W bieżącym podrozdziale zajmiemy się tworzeniem obrazów od podstaw — piksel po pikselu.

Aby utworzyć nowy obraz, musimy najpierw skonstruować obiekt klasy `BufferedImage`.

```
image = new BufferedImage(width, height, BufferedImage.TYPE_INT_ARGB);
```

Korzystając z metody `getRaster`, uzyskamy obiekt typu `WritableRaster`, który umożliwi modyfikacje pikseli obrazu.

```
WritableRaster raster = image.getRaster();
```

Metoda `setPixel` tego obiektu pozwala modyfikować pojedynczy piksel. Nie wystarczy jednak w tym celu przypisać mu po prostu odpowiedniej wartości typu `Color`. Musimy wiedzieć, w jaki sposób obraz przechowuje informacje o kolorach, co zależy od jego *typu*. W przypadku typu `TYPE_INT_ARGB` każdy z pikseli opisany jest przez cztery wartości z przedziału od 0 do 255

reprezentujące wartość koloru czerwonego, zielonego, niebieskiego oraz wartość alfa. Przekazemy je metodzie `setPixel` za pomocą tablicy.

```
int[] black = { 0, 0, 0, 255 };
raster.setPixel(i, j, black);
```

W żargonie Java 2D wartości te określa się mianem *wartości próbki* piksela.



Istnieją także wersje metody `setPixel`, których parametrami są tablice typów `float[]` bądź `double[]`. Wartości, które należy umieścić w tych tablicach, *nie* są jednak znormalizowanymi wartościami kolorów z przedziału od 0 do 1.

```
float[] red = { 1.0F, 0.0F, 0.0F, 1.0F };
raster.setPixel(i, j, red); //ŹLE
```

Niezależnie od typu tablicy zawsze musimy dostarczyć w niej wartości z przedziału od 0 do 255.

Za pomocą metody `setPixels` możemy zmodyfikować od razu prostokątny obszar pikseli. Metodzie musimy przekazać pozycję pierwszego z pikseli, szerokość i wysokość obszaru prostokąta oraz tablicę zawierającą wartości próbek dla wszystkich pikseli obszaru. Jeśli obraz jest typu `TYPE_INT_ARGB`, to w tablicy tej umieścimy wartości koloru czerwonego, zielonego, niebieskiego i wartość alfa dla pierwszego piksela, następnie wartości dla drugiego piksela itd.

```
int[] pixels = new int [4 * width * height];
pixels[0] = . . . //wartość koloru czerwonego dla pierwszego piksela
pixels[1] = . . . //wartość koloru zielonego dla pierwszego piksela
pixels[2] = . . . //wartość koloru niebieskiego dla pierwszego piksela
pixels[3] = . . . //wartość alfa dla pierwszego piksela
. . .
raster.setPixels(x, y, width, height, pixels);
```

Aby odczytać wartości piksela, korzystamy z metody `getPixel`, której musimy dostarczyć tablicę dla wartości próbki.

```
int[] sample = new int[4];
raster.getPixel(x, y, sample);
Color c = new Color(sample[0], sample[1], sample[2], sample[3]);
```

Za pomocą metody `getPixels` możemy odczytać wartości próbek dla wielu pikseli.

```
raster.getPixels(x, y, width, height, samples);
```

Metod `setPixel` i `getPixel` możemy oczywiście używać także dla innych typów obrazów niż `TYPE_INT_ARGB` pod warunkiem, że znamy sposób reprezentacji wartości pikseli dla danego typu obrazu.

Jeśli natomiast musimy operować na obrazie dowolnego typu, to zadanie jest nieco trudniejsze. Każdy typ obrazu posiada określony *model kolorów*, który umożliwia przekształcenie wartości próbek na model kolorów RGB.

Metoda `getColorModel` zwraca model kolorów dla danego obrazu.

```
ColorModel model = image.getColorModel();
```





Kolor zapisany przy użyciu modelu RGB może wyglądać różnie w zależności od charakterystyki konkretnego urządzenia. International Color Consortium ([www.color.org](http://www.color.org)) zaleca więc, aby danym o kolorach towarzyszył zawsze *profil ICC* określający sposób odwzorowania kolorów w stosunku do standardowej specyfikacji kolorów 1931 CIE XYZ. Specyfikacja ta została zaproponowana przez międzynarodową organizację CIE (*Commission Internationale de l'Eclairage* — [www.cie.co.at/cie](http://www.cie.co.at/cie)) zajmującą się tworzeniem technicznych standardów związanych z kolorami. Specyfikacja ta określa kolory rozróżniane przez ludzkie oko za pomocą trzech współrzędnych X, Y, Z. (Więcej informacji na ten temat znajdziemy w rozdziale 13. książki Foley, van Damma, Feinera, et al.)

Profile ICC są dość skomplikowane i dlatego zaproponowano prostszy standard sRGB (patrz <http://www.w3.org/Graphics/Color/sRGB.html>) określający odwzorowanie pomiędzy wartościami modelu RGB a wartościami specyfikacji 1931 CIE XYZ, które sprawdza się w przypadku typowych monitorów kolorowych. Java 2D korzysta z tego odwzorowania podczas konwersji pomiędzy RGB i innymi modelami kolorów.

Aby uzyskać informacje o kolorze piksela, wywołujemy metodę `getDataElements` klasy `Raster`. Zwraca ona obiekt zawierający opis wartości koloru piksela w danym modelu kolorów.

```
Object data = raster.getDataElements(x, y, null);
```



Obiekt zwracany przez metodę `getDataElements` jest w rzeczywistości tablicą wartości próbki. Oczywiście przetwarzając go, nie musimy tego wiedzieć, ale wyjaśnia to nazwę wybraną przez projektantów dla tej metody.

Model kolorów umożliwia zamianę wartości próbki zawartych w obiekcie na standardowe wartości ARGB. Metoda `getRGB` zwraca wartość typu `int`, która zawiera wartości koloru czerwonego, zielonego, niebieskiego i wartość alfa upakowane w czterech blokach po osiem bitów każdy. Na podstawie tej wartości możemy utworzyć obiekt klasy `Color`, korzystając z konstruktora `Color(int argb, boolean hasAlpha)`.

```
int argb = model.getRGB(data);
Color color = new Color(argb, true);
```

Jeśli chcemy nadać pikselowi pewien kolor, to musimy wykonać opisane kroki w odwrotnej kolejności. Metoda `getRGB` zwróci wartość typu `int` zawierającą upakowane wartości koloru czerwonego, zielonego, niebieskiego i wartość alfa. Wartość tę musimy przekazać metodzie `getDataElements` klasy `ColorModel`. Zwróci ona obiekt zawierający wartości próbki w danym modelu. Obiekt ten prześlemy następnie metodzie `setDataElements` klasy `WritableRaster`.

```
int argb = color.getRGB();
Object data = model.getDataElements(argb, null);
raster.setDataElements(x, y, data);
```

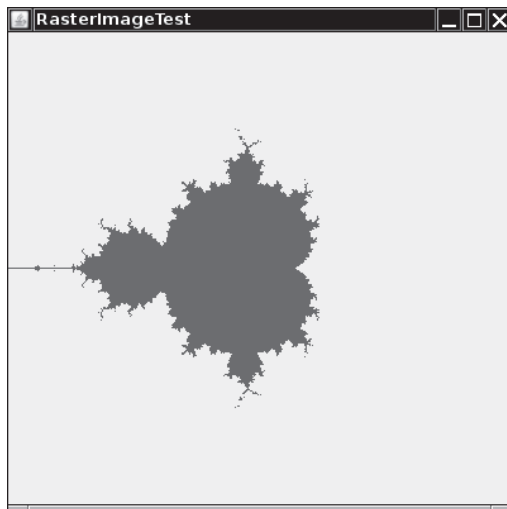
Ilustracją omówionych metod tworzenia obrazu z pojedynczych pikseli zbiór Mandelbrota pokazany na rysunku 7.28.

Idea zbioru Mandelbrota opiera się na związaniu z każdym punktem płaszczyzny ciągu liczb. Jeśli ciąg ten posiada skończoną granicę, to zaznaczamy odpowiadający mu punkt. Jeśli dąży do nieskończoności, to związany z ciągiem punkt płaszczyzny pozostawiamy bez zmian.

A oto sposób konstrukcji najprostszego zbioru Mandelbrota. Dla każdego punktu  $(a, b)$  tworzenie ciągu rozpoczynamy od  $(x, y) = (0, 0)$  i następnie wykonujemy iteracje, korzystając z poniższych wzorów:

**Rysunek 7.28.**

Zbiór Mandelbrota



$$x_{\text{new}} = x^2 - y^2 + a$$

$$y_{\text{new}} = 2 * x * y + b$$

Następnie należy sprawdzić, czy uzyskany ciąg posiada granicę, czy dąży do nieskończoności. W praktyce okazuje się, że jeśli  $x$  i  $y$  będą większe od 2, to ciąg będzie dążył do nieskończoności. Zaznaczamy jedynie te piksele, dla których ciąg posiada skończoną granicę. (Formuła tworzenia ciągu liczb wywodzi się z matematyki liczb zespolonych. Nie będziemy jej wyprowadzać, lecz skorzystamy z gotowego wzoru. Więcej informacji na temat fraktali znajdziesz na stronie <http://classes.yale.edu/fractals>).

Listing 7.8 zawiera kompletny tekst źródłowy programu. Program pokazuje między innymi, w jaki sposób należy wykorzystać klasę `ColorModel` do przekształcenia wartości typu `Color` na dane opisujące piksel. Sposób ten jest niezależny od typu obrazu. Możemy sprawdzić jego działanie, zmieniając typ obrazu na przykład na `TYPE_BYTE_GRAY`. Nie wymaga to żadnych innych modyfikacji kodu, ponieważ przekształceniem wartości kolorów na wartości próbek zajmie się automatycznie model kolorów.

**Listing 7.8.** *rasterImage/RasterImageFrame.java*

```
package rasterImage;

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

/**
 * Ramka wyświetlająca zbiór Mandelbrota.
 */
public class RasterImageFrame extends JFrame
{
    private static final double XMIN = -2;
    private static final double XMAX = 2;
    private static final double YMIN = -2;
```

```

private static final double YMAX = 2;
private static final int MAX_ITERATIONS = 16;
private static final int IMAGE_WIDTH = 400;
private static final int IMAGE_HEIGHT = 400;

public RasterImageFrame()
{
    BufferedImage image = makeMandelbrot(IMAGE_WIDTH, IMAGE_HEIGHT);
    add(new JLabel(new ImageIcon(image)));
    pack();
}

/**
 * Tworzy obraz zbioru Mandelbrota.
 * @param width szerokość
 * @param height wysokość
 * @return obraz
 */
public BufferedImage makeMandelbrot(int width, int height)
{
    BufferedImage image = new BufferedImage(width, height, BufferedImage.TYPE_INT_ARGB);
    WritableRaster raster = image.getRaster();
    ColorModel model = image.getColorModel();

    Color fractalColor = Color.red;
    int argb = fractalColor.getRGB();
    Object colorData = model.getDataElements(argb, null);

    for (int i = 0; i < width; i++)
        for (int j = 0; j < height; j++)
        {
            double a = XMIN + i * (XMAX - XMIN) / width;
            double b = YMIN + j * (YMAX - YMIN) / height;
            if (!escapesToInfinity(a, b)) raster.setDataElements(i, j, colorData);
        }
    return image;
}

private boolean escapesToInfinity(double a, double b)
{
    double x = 0.0;
    double y = 0.0;
    int iterations = 0;
    while (x <= 2 && y <= 2 && iterations < MAX_ITERATIONS)
    {
        double xnew = x * x - y * y + a;
        double ynew = 2 * x * y + b;
        x = xnew;
        y = ynew;
        iterations++;
    }
    return x > 2 || y > 2;
}
}

```

**API** java.awt.image.BufferedImage 1.2

- BufferedImage(int width, int height, int imageType)  
tworzy obiekt klasy BufferedImage.  
*Parametry:* width, height szerokość i wysokość obrazu  
imageType jedna z wartości TYPE\_INT\_RGB, TYPE\_INT\_ARGB,  
TYPE\_BYTE\_GRAY, TYPE\_BYTE\_INDEXED,  
TYPE\_USHORT\_555\_RGB itd.
- ColorModel getColorModel()  
zwraca model kolorów dla danego obrazu.
- WritableRaster getRaster()  
pobiera raster umożliwiający operacje na pikselach obrazu.

**API** java.awt.image.Raster 1.2

- Object getDataElements(int x, int y, Object data)  
zwraca wartości próbki piksela w postaci tablicy, której typ i liczba elementów zależy od modelu kolorów. Jeśli parametr data różny jest od null, to metoda przyjmuje, że reprezentuje on tablicę odpowiednią do umieszczenia wartości próbek w danym modelu. W przeciwnym razie tworzona i wypełniana jest nowa tablica. Typ i liczba elementów tablicy zależą od modelu kolorów.
  - int[] getPixel(int x, int y, int[] sampleValues)
  - float[] getPixel(int x, int y, float[] sampleValues)
  - double[] getPixel(int x, int y, double[] sampleValues)
  - int[] getPixels(int x, int y, int w, int h, int[] sampleValues)
  - float[] getPixels(int x, int y, int w, int h, float[] sampleValues)
  - double[] getPixels(int x, int y, int w, int h, double[] sampleValues)
- Zwracają wartości próbki dla pojedynczego piksela lub wielu pikseli leżących w obszarze prostokąta. Wartości te zwracane są w postaci tablicy, której długość zależy od modelu kolorów. Jeśli parametr sampleValues jest różny od null, to przyjmuje się, że reprezentuje on tablicę o odpowiedniej długości do umieszczenia w niej wartości próbek i wypełnia ją. W przeciwnym razie tworzona jest nowa tablica. Metody te są przydatne pod warunkiem, że znamy sposób interpretacji wartości próbek w danym modelu kolorów.

**API** java.awt.image.WritableRaster 1.2

- void setDataElements(int x, int y, Object data)  
nadaje pikselowi wartość próbki. Argument data jest tablicą wypełnioną wartościami próbki w danym modelu kolorów. Typ i liczba elementów tablicy zależą od modelu kolorów.

- `void setPixel(int x, int y, int[] sampleValues)`
- `void setPixel(int x, int y, float[] sampleValues)`
- `void setPixel(int x, int y, double[] sampleValues)`
- `void setPixels(int x, int y, int width, int height, int[] sampleValues)`
- `void setPixels(int x, int y, int width, int height, float[] sampleValues)`
- `void setPixels(int x, int y, int width, int height, double[] sampleValues)`

nadają wartości próbki pojedynczemu pikselowi lub prostokątnemu obszarowi pikseli. Metody te są przydatne pod warunkiem, że znamy sposób interpretacji wartości próbek w danym modelu kolorów.

#### API | `java.awt.image.ColorModel` 1.2

- `int getRGB(Object data)`

zwraca wartość ARGB odpowiadającą wartości próbki przekazanej za pomocą tablicy `data`. Typ i liczba elementów tablicy zależą od modelu kolorów.

- `Object getDataElements(int argb, Object data)`

zwraca wartości próbki dla danej wartości koloru. Jeśli parametr `data` różny jest od `null`, to metoda przyjmuje, że reprezentuje on tablicę odpowiednią do umieszczenia wartości próbek w danym modelu i wypełnia ją. W przeciwnym razie tworzona i wypełniana jest nowa tablica. Typ i liczba elementów tablicy zależą od modelu kolorów.

#### API | `java.awt.Color` 1.0

- `Color(int argb, boolean hasAlpha)` 1.2

tworzy kolor o podanej wartości ARGB, gdy parametr `hasAlpha` ma wartość `true` lub o podanej wartości RGB, gdy parametr `hasAlpha` ma wartość `false`.

- `int getRGB()`

zwraca wartość koloru ARGB odpowiadającą danemu kolorowi.

## 7.11.2. Filtrowanie obrazów

W poprzednim podrozdziale pokazaliśmy, w jaki sposób utworzyć obraz od podstaw. Często jednak dysponujemy już gotowym obrazem, który następnie należy poddać obróbce.

Oczywiście możemy w tym celu użyć przedstawionych już metod `getPixel/getDataElements`. Jednak Java 2D dostarcza także gotowych *filtrów* obrazów, które implementują najczęściej spotykane operacje.

Klasy operacji na obrazach implementują interfejs `BufferedImageOp`. Po utworzeniu obiektu takiej klasy wywołujemy jej metodę `filter`, aby wykonać operację na obrazie.

```
BufferedImageOp op = . . . ;
BufferedImage filteredImage
    = new BufferedImage(image.getWidth(), image.getHeight(), image.getType());
op.filter(image, filteredImage);
```

Jedynie niektóre operacje mogą być wykonane bez konieczności utworzenia wynikowego obrazu (czyli przez wywołanie `op.filter(image, image)`).

Interfejs `BufferedImageOp` implementuje pięć następujących klas:

```
AffineTransformOp
RescaleOp
LookupOp
ColorConvertOp
ConvolveOp
```

Klasa `AffineTransformOp` umożliwi afiniczne przekształcenie pikseli. Poniżej przykład obrotu obrazu dookoła jego środka.

```
AffineTransform transform
    = AffineTransform.getRotateInstance(Math.toRadians(angle), image.getWidth() /
    ↪ 2, image.getHeight() / 2);
AffineTransformOp op
    = new AffineTransformOp(transform, interpolation);
op.filter(image, filteredImage);
```

Konstruktorowi klasy `AffineTransformOp` musimy przekazać przekształcenie afiniczne oraz strategię *interpolacji*. Interpolacja wykorzystywana jest do tworzenia pikseli obrazu wynikowego, w przypadku gdy nie istnieje jednoznaczne odwzorowanie pikseli obrazu źródłowego na piksele obrazu wynikowego. Sytuacja taka występuje na przykład podczas obrotu obrazu. Dostępne są dwie strategie interpolacji: `AffineTransformOp.TYPE_BILINEAR` oraz `AffineTransformOp.TYPE_NEAREST_NEIGHBOR`. Pierwsza z nich wymaga nieco więcej obliczeń, ale daje lepszy efekt.

Program z listingu 7.9 pozwala wykonywać obrót obrazu o kąt 5 stopni (patrz rysunek 7.29).

**Rysunek 7.29.**  
Obrót obrazu



Listing 7.9. *imageProcessing/ImageProcessingFrame.java*

```

package imageProcessing;

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import java.awt.image.*;
import java.io.*;
import javax.imageio.*;
import javax.swing.*;
import javax.swing.filechooser.*;

/**
 * Ramka posiadająca menu umożliwiające załadowanie obrazu z pliku
 * i wybór przekształcenia oraz komponent przedstawiający wynik przekształcenia.
 */
public class ImageProcessingFrame extends JFrame
{
    private static final int DEFAULT_WIDTH = 400;
    private static final int DEFAULT_HEIGHT = 400;

    private BufferedImage image;

    public ImageProcessingFrame()
    {
        setTitle("ImageProcessingTest");
        setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);

        add(new JComponent()
            {
                public void paintComponent(Graphics g)
                {
                    if (image != null) g.drawImage(image, 0, 0, null);
                }
            });

        JMenu fileMenu = new JMenu("File");
        JMenuItem openItem = new JMenuItem("Open");
        openItem.addActionListener(new ActionListener()
            {
                public void actionPerformed(ActionEvent event)
                {
                    openFile();
                }
            });
        fileMenu.add(openItem);

        JMenuItem exitItem = new JMenuItem("Exit");
        exitItem.addActionListener(new ActionListener()
            {
                public void actionPerformed(ActionEvent event)
                {
                    System.exit(0);
                }
            });
        fileMenu.add(exitItem);
    }
}

```

```
JMenu editMenu = new JMenu("Edit");
JMenuItem blurItem = new JMenuItem("Blur");
blurItem.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent event)
    {
        float weight = 1.0f / 9.0f;
        float[] elements = new float[9];
        for (int i = 0; i < 9; i++)
            elements[i] = weight;
        convolve(elements);
    }
});
editMenu.add(blurItem);

JMenuItem sharpenItem = new JMenuItem("Sharpen");
sharpenItem.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent event)
    {
        float[] elements = { 0.0f, -1.0f, 0.0f, -1.0f, 5.f, -1.0f, 0.0f,
        ↪ -1.0f, 0.0f };
        convolve(elements);
    }
});
editMenu.add(sharpenItem);

JMenuItem brightenItem = new JMenuItem("Brighten");
brightenItem.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent event)
    {
        float a = 1.1f;
        // float b = 20.0f;
        float b = 0;
        RescaleOp op = new RescaleOp(a, b, null);
        filter(op);
    }
});
editMenu.add(brightenItem);

JMenuItem edgeDetectItem = new JMenuItem("Edge detect");
edgeDetectItem.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent event)
    {
        float[] elements = { 0.0f, -1.0f, 0.0f, -1.0f, 4.f, -1.0f, 0.0f,
        ↪ -1.0f, 0.0f };
        convolve(elements);
    }
});
editMenu.add(edgeDetectItem);

JMenuItem negativeItem = new JMenuItem("Negative");
negativeItem.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent event)
```



```

        {
            short[] negative = new short[256 * 11];
            for (int i = 0; i < 256; i++)
                negative[i] = (short) (255 - i);
            ShortLookupTable table = new ShortLookupTable(0, negative);
            LookupOp op = new LookupOp(table, null);
            filter(op);
        }
    });
    editMenu.add(negativeItem);

    JMenuItem rotateItem = new JMenuItem("Rotate");
    rotateItem.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent event)
        {
            if (image == null) return;
            AffineTransform transform =
                ↪ AffineTransform.getRotateInstance(Math.toRadians(5),
                image.getWidth() / 2, image.getHeight() / 2);
            AffineTransformOp op = new AffineTransformOp(transform,
                AffineTransformOp.TYPE_BICUBIC);
            filter(op);
        }
    });
    editMenu.add(rotateItem);

    JMenuBar menuBar = new JMenuBar();
    menuBar.add(fileMenu);
    menuBar.add(editMenu);
    setJMenuBar(menuBar);
}

/**
 * Otwiera plik i ładuje obrazek.
 */
public void openFile()
{
    JFileChooser chooser = new JFileChooser(".");
    chooser.setCurrentDirectory(new File(getClass().getPackage().getName()));
    String[] extensions = ImageIO.getReaderFileSuffixes();
    chooser.setFileFilter(new FileNameExtensionFilter("Image files", extensions));
    int r = chooser.showOpenDialog(this);
    if (r != JFileChooser.APPROVE_OPTION) return;

    try
    {
        Image img = ImageIO.read(chooser.getSelectedFile());
        image = new BufferedImage(img.getWidth(null), img.getHeight(null),
            BufferedImage.TYPE_INT_RGB);
        image.getGraphics().drawImage(img, 0, 0, null);
    }
    catch (IOException e)
    {
        JOptionPane.showMessageDialog(this, e);
    }
    repaint();
}
}

```

```

/**
 * Stosuje filtr obrazu i odrysowuje obraz.
 * @param op rodzaj przekształcenia
 */
private void filter(BufferedImage op)
{
    if (image == null) return;
    image = op.filter(image, null);
    repaint();
}

/**
 * Wykonuje operację splotu i odrysowuje obraz.
 * @param elements jądro splotu (tablica zawierająca 9 elementów macierzy)
 */
private void convolve(float[] elements)
{
    Kernel kernel = new Kernel(3, 3, elements);
    ConvolveOp op = new ConvolveOp(kernel);
    filter(op);
}
}

```

Klasa `RescaleOp` umożliwia wykonanie na pikselach operacji określonej wzorem:

$$x_{\text{nowe}} = a \cdot x + b$$

gdzie  $x$  jest wartością próbki piksela. W wyniku operacji, dla której  $a > 1$ , otrzymujemy rozjaśniony obraz. Obiekt klasy `RescaleOp` tworzymy, określając parametry skalowania oraz opcjonalne wskazówki. W programie z listingu 7.9 używamy następujących parametrów:

```

float a = 1.1f;
float b = 20.0f;
RescaleOp op = new RescaleOp(a, b, null);

```

Można również dostarczyć osobny parametr skalowania dla każdej składowej koloru — patrz omówienie metod klasy `RescaleOp` na końcu tego podrozdziału.

Klasa `LookupOp` pozwala określić dowolne odwzorowanie wartości próbek. W tym celu musimy dostarczyć jej tablicę opisującej sposób odwzorowania. Nasz przykładowy program wykorzystuje klasę `LookupOp` do utworzenia *negatywu* obrazu zamieniając kolor  $c$  na  $255 - c$ .

Konstruktor klasy `LookupOp` musi otrzymać jako parametr obiekt klasy `LookupTable` oraz opcjonalnie mapę wskazówek. Klasa `LookupTable` jest klasą abstrakcyjną. Dostępne są dwie jej konkretne klasy pochodne: `ByteLookupTable` i `ShortLookupTable`. Ponieważ wartości RGB zapisywane są za pomocą bajtów, powinna nam wystarczyć klasa `ByteLookupTable`. Jednak ze względu na błąd opisany na stronie [http://bugs.sun.com/bugdatabase/view\\_bug.do?bug\\_id=6183251](http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6183251) użyjemy zamiast niej klasy `ShortLookupTable`. Oto sposób, w jaki tworzymy obiekt `LookupOp` w programie przykładowym:

```

short negative[] = new short[256];
for (int i = 0; i < 256; i++) negative[i] = (short) (255 - i);
ShortLookupTable table = new ShortLookupTable(0, negative);
LookupOp op = new LookupOp(table, null);

```

Odzworowywanie wykonywane jest dla wartości każdego z kolorów oddzielnie, lecz nie dla wartości alfa. Można również dostarczyć osobne tablice odzworowań dla każdej składowej koloru — patrz omówienie klas na końcu podrozdziału.



Klasy `LookupOp` nie możemy zastosować w przypadku obrazów posiadających kolor indeksowany. (W ich przypadku wartości próbki są indeksami palety kolorów).

Klasa `ColorConvertOp` przydatna jest do przekształceń przestrzeni kolorów. Nie będziemy jej tutaj omawiać.

Największych możliwości dostarcza klasa `ConvolveOp`, która umożliwia realizację operacji *splotu*. Nie będziemy wnikać głębiej w jej matematyczne podstawy. Podstawowa idea tej operacji jest prosta. Rozważmy na przykład *filtr rozmycia* (patrz rysunek 7.30).

### Rysunek 7.30.

Rozmycie obrazu



Efekt rozmycia uzyskujemy, zastępując każdy piksel wartością *średnią* danego piksela i jego ośmiu sąsiadów. Intuicyjnie rozumiemy sposób działania takiego przekształcenia. Natomiast matematyka pozwala nam je opisać jako splot o następującym *jądrze*:

$$\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$$

Jądro splotu jest macierzą przedstawiającą wagi, które zastosowane mają być do sąsiednich wartości. Efektem splotu o powyższym jądrze jest rozmycie obrazu. Natomiast jądro postaci

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

pozwala *wykryć krawędzie*, na których dokonuje się zmiana koloru (patrz rysunek 7.31). Wykrywanie krawędzi jest ważną techniką stosowaną w przetwarzaniu obrazów fotograficznych.

**Rysunek 7.31.**  
Wykrywanie  
krawędzi



Aby wykonać operację splotu, musimy najpierw utworzyć jej jądro jako obiekt klasy `Kernel`. Dla utworzonego na jego podstawie obiektu klasy `ConvolveOp` wykonujemy jak zwykle metodę `filter`.

```
float[] elements =
{
    0.0f, -1.0f, 0.0f,
    -1.0f, 4.0f, -1.0f,
    0.0f, -1.0f, 0.0f
};
Kernel kernel = new Kernel(3, 3, elements);
ConvolveOp op = new ConvolveOp(kernel);
op.filter(image, filteredImage);
```

Program z listingu 7.9 pozwala załadować obraz zapisany w pliku GIF lub JPEG i wykonać na nim omówione dotąd operacje. Dzięki wykorzystaniu implementacji tych operacji dostępnych w postaci klas Java 2D program jest bardzo prosty.

#### **API** `java.awt.image.BufferedImageOp` 1.2

- `BufferedImage filter(BufferedImage source, BufferedImage dest)`  
wykonuje operację na obrazie `source`, której wynikiem jest obraz `dest`. Jeśli `dest` posiada wartość `null`, to tworzony jest i zwracany nowy obraz.

#### **API** `java.awt.image.AffineTransformOp` 1.2

- `AffineTransformOp(AffineTransform t, int interpolationType)`  
tworzy obiekt przekształcenia afinicznego obrazu. Parametr `interpolationType` przyjmuje jedną z wartości `TYPE_BILINEAR` lub `TYPE_NEAREST_NEIGHBOR`.

**API** java.awt.image.RescaleOp 1.2

- RescaleOp(float a, float b, RenderingHints hints)
- RescaleOp(float[] as, float[] bs, RenderingHints hints)

tworzą obiekt klasy RescaleOp umożliwiający wykonanie operacji skalowania określonej wzorem  $x_{\text{nowe}} = a*x + b$ . W przypadku zastosowania pierwszego konstruktora podczas skalowania wszystkich składowych (ale nie wartości alfa) zastosowany zostaje ten sam współczynnik. W przypadku drugiego konstruktora dostarczamy osobnej wartości współczynników dla każdej składowej koloru (wtedy wartość alfa nie jest modyfikowana) lub dla każdej składowej koloru i wartości alfa.

**API** java.awt.image.LookupOp 1.2

- LookupOp(LookupTable table, RenderingHints hints)
- tworzy obiekt odwzorowania wartości próbek obrazu.

**API** java.awt.image.ByteLookupTable 1.2

- ByteLookupTable(int offset, byte[] data)
- ByteLookupTable(int offset, byte[][] data)

tworzą tablicę odwzorowania złożoną z bajtów. Wartość parametru offset zostaje odjęta od danych wejściowych przed zastosowaniem wartości data. W przypadku zastosowania pierwszego konstruktora podczas skalowania wszystkich składowych (ale nie wartości alfa) zastosowany zostaje ten sam współczynnik. W przypadku drugiego konstruktora dostarczamy osobnej wartości współczynników dla każdej składowej koloru (wtedy wartość alfa nie jest modyfikowana) lub dla każdej składowej koloru i wartości alfa.

**API** java.awt.image.ShortLookupTable 1.2

- ByteLookupTable(int offset, short[] data)
- ByteLookupTable(int offset, short[][] data)

tworzą tablicę odwzorowania złożoną z wartości typu short. Wartość parametru offset zostaje odjęta od danych wejściowych przed zastosowaniem wartości data. W przypadku zastosowania pierwszego konstruktora podczas skalowania wszystkich składowych (ale nie wartości alfa) zastosowany zostaje ten sam współczynnik. W przypadku drugiego konstruktora dostarczamy osobnej wartości współczynników dla każdej składowej koloru (wtedy wartość alfa nie jest modyfikowana) lub dla każdej składowej koloru i wartości alfa.

**API** java.awt.ConvolveOp 1.2

- ConvolveOp(Kernel kernel)

- `ConvolveOp(Kernel kernel, int edgeCondition, RenderingHints hints)`

Tworzy obiekt splotu. Warunki brzegowe zostają określone za pomocą stałej `EDGE_NO_OP` lub `EDGE_ZERO_FILL`. Wartości brzegowe wymagają specjalnego traktowania, ponieważ nie posiadają wystarczającej liczby wartości sąsiednich dla wyznaczenia splotu. Wartością domyślną jest `EDGE_ZERO_FILL`.

**API** `java.awt.image.Kernel` **1.2**

- `Kernel(int width, int height, float[] matrixElements)`  
tworzy jądro splotu dla podanej macierzy.

## 7.12. Drukowanie

Początkowo Java Development Kit nie zawierał obsługi operacji drukowania. Drukowanie z appletów nie było w ogóle możliwe, natomiast drukowanie z aplikacji wymagało wykorzystania bibliotek dostarczanych przez innych producentów. JDK 1.1 umożliwiał już tworzenie prostych wydruków o niskiej jakości. Model drukowania dostępny w tej wersji JDK służyć miał przede wszystkim producentom przeglądarek stron internetowych do wydruku appletów pojawiających się na stronach WWW. Nie zyskał sobie jednak sympatii ani producentów przeglądarek, ani pozostałych programistów.

Java SE 1.2 wprowadziła nowy model drukowania zintegrowany z możliwościami graficznymi platformy Java 2D. Java SE 1.4 przyniosła istotne rozszerzenia jego możliwości, takie jak wykrywanie charakterystyki drukarek czy wysyłanie prac do serwerów wydruków.

Omawiając możliwości drukowania na platformie Java, przedstawimy sposób tworzenia podstawowych wydruków, wydruków wielostronicowych oraz wykorzystania możliwości graficznych Java 2D do tworzenia podglądu wydruku.

### 7.12.1. Drukowanie grafiki

W podrozdziale tym przedstawimy sposób drukowania grafiki 2D. Oczywiście grafika taka może zawierać także tekst tworzony za pomocą różnych czcionek, a nawet składać się wyłącznie z samego tekstu.

Aby utworzyć wydruk, musimy:

- dostarczyć obiekt implementujący interfejs `Printable`,
- uruchomić zadanie drukowania.

Interfejs `Printable` posiada tylko jedną metodę:

```
int print(Graphics g, PageFormat format, int page)
```

Metoda ta jest wywoływana przez podsystem drukowania, gdy chce on uzyskać kolejną sformatowaną stronę do wydruku. Nasza implementacja tej metody musi więc utworzyć obraz

wydruku, korzystając z dostarczonego kontekstu graficznego. Drugi z parametrów metody opisuje rozmiary strony i marginesów, a trzeci parametr określa numer drukowanej strony.

Aby uruchomić zadanie drukowania, korzystamy z klasy `PrinterJob`. Najpierw musimy wywołać jej metodę statyczną `getPrinterJob`, która zwróci obiekt tej klasy. Następnie musimy przekazać otrzymanemu obiektowi klasy `PrinterJob` nasz obiekt implementujący interfejs `Printable`.

```
Printable canvas = . . . ;
PrinterJob job = PrinterJob.getPrinterJob();
job.setPrintable(canvas);
```

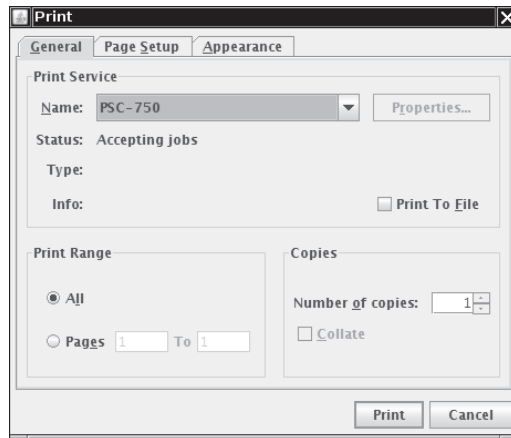


W JDK 1.1 dostępna jest klasa `PrintJob` wykorzystywana przez przestarzały model drukowania. Nie należy jej mylić z klasą `PrinterJob`.

Zanim uruchomimy zadanie drukowania, powinniśmy najpierw wywołać metodę `printDialog` wyświetlającą okno dialogowe drukowania (patrz rysunek 7.32). Umożliwia ono użytkownikowi wybór drukarki (jeśli dostępna jest więcej niż jedna), określenie zakresu drukowanych stron i innych opcji drukowania.

### Rysunek 7.32.

Okno dialogowe wydruku dla dowolnej platformy



Parametry wydruku zbieramy za pomocą obiektu implementującego interfejs `PrintRequestAttributeSet`. Dostępna jest gotowa klasa `HashPrintRequestAttributeSet` implementująca ten interfejs.

```
HashPrintRequestAttributeSet attributes = new HashPrintRequestAttributeSet();
```

Po dodaniu odpowiednich atrybutów obiekt `attributes` przekazujemy metodzie `printDialog`.

Metoda ta zwraca wartość `true`, jeśli użytkownik zaakceptował wydruk, wybierając przycisk *OK* lub wartość `false` — w przeciwnym razie. Jeśli użytkownik zaakceptował operację drukowania, to wywołujemy metodę `print` klasy `PrinterJob` uruchamiającą proces drukowania. Metoda `print` może wyrzucić wyjątek `PrinterException`. Poniżej przedstawiamy ogólną postać kodu wydruku.

```
if (job.printDialog(attributes))
{
    try
```

```

    {
        job.print(attributes);
    }
    catch (PrinterException exception)
    {
        . . .
    }
}

```



W wersjach wcześniejszych niż JDK 1.4 używano okna dialogowego wydruku dostarczanego przez konkretny system. Jeśli chcemy skorzystać z tego okna w nowszych wersjach JDK, należy wywołać metodę `printDialog` bez parametrów (nie można wtedy zebrać ustawień użytkownika w postaci zbioru atrybutów).

Podczas drukowania metoda `print` klasy `PrinterJob` wywołuje wielokrotnie metodę `print` obiektu implementującego interfejs `Printable`.

Ponieważ obiekt klasy `PrinterJob` nie posiada informacji o liczbie stron, które będą wydrukowane, to wywołuje po prostu cyklicznie metodę `print`. Tak długo, jak zwraca ona wartość `Printable.PAGE_EXISTS`, obiekt `PrinterJob` tworzy kolejne strony wydruku. Swoje działanie kończy, gdy metoda `print` zwróci wartość `Printable.NO_SUCH_PAGE`.



Numerzy stron, które obiekt klasy `PrinterJob` przekazuje metodzie `print`, rozpoczynają się od 0.

Obiekt klasy `PrinterJob` nie dysponuje więc informacją o liczbie drukowanych stron, dopóki wydruk nie zakończy się. Z tego powodu okno dialogowe drukowania nie potrafi wyświetlić odpowiedniego zakresu drukowanych stron, a jedynie zakres od 1 do 1. Pokażemy później, w jaki sposób można to poprawić, dostarczając obiektowi klasy `PrinterJob` obiektowi klasy `Book`.

Jak już wspomnieliśmy w procesie drukowania metoda `print` obiektu typu `Printable` wywoływana jest wielokrotnie. Obiekt klasy `PrinterJob` może wywołać ją wielokrotnie *dla tej samej strony wydruku*. Dlatego też metoda `print` nie powinna sama zliczać drukowanych stron, a jedynie bazować na przekazywanym jej parametrze określającym numer strony. Metoda `print` wywoływana jest wielokrotnie dla tej samej strony, zwłaszcza w przypadku wydruku za pomocą drukarek igłowych lub atramentowych, które stopniowo tworzą wydruk linia po linii. Także w przypadku drukarek laserowych, które posiadają techniczną możliwość wydrukowania od razu całości strony, metoda `print` może być wywoływana wielokrotnie, umożliwiając w ten sposób obiektowi klasy `PrinterJob` lepsze zarządzanie buforem wydruku.

Jeśli obiekt klasy `PrinterJob` chce wydrukować kolejny fragment strony, to określa odpowiednio granicę obszaru przycięcia kontekstu graficznego i wywołuje metodę `print`. Utworzony przez nią wydruk zostanie automatycznie przycięty do obszaru odpowiadającego bieżącemu fragmentowi strony. Metoda `print` nie musi być nawet tego świadoma z jednym wyjątkiem: *nie* powinna ona zmieniać obszaru przycięcia.

Parametr klasy `PageFormat` przekazuje metodzie `print` informacje o formacie drukowanej strony. Metody `getWidth` i `getHeight` zwracają rozmiar strony w *punktach*. Jeden punkt odpowiada  $\frac{1}{72}$  cala. (Jeden cal to 25,4 milimetra). Strona formatu *A4* posiada w przybliżeniu wymiary 595 na 842 punkty, a amerykańskiego formatu *letter* 612 na 792 punkty.





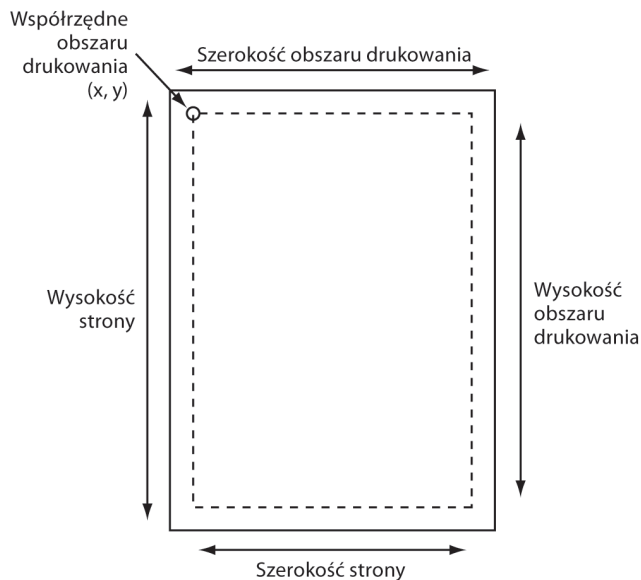
Kontekst graficzny otrzymywany przez metodę `print` posiada obszar przycięcia odpowiadający marginesom strony. Jeśli metoda `print` zmieni obszar przycięcia, to może uzyskać możliwość drukowania także w obszarze marginesów. Metoda `print` powinna jednak przestrzegać obszaru przycięcia przekazanego jej kontekstu graficznego. Operując na obszarze przycięcia, powinna korzystać z metody `clip` zamiast `setClip`. Jeśli z konkretnych powodów musi ona usunąć przekazany jej obszar przycięcia, to powinna go najpierw zachować, korzystając z metody `getClip` i odtworzyć po zakończeniu operacji z własnym obszarem przycięcia.

Punkty stosowane są nie tylko jako jednostka wymiarów strony i marginesów, ale także jako domyślna jednostka dla wszystkich kontekstów graficznych związanych z drukowaniem. Można sprawdzić to za pomocą przykładowego programu zamieszczonego na końcu podrozdziału. Drukuje on dwa wiersze tekstu w odległości 72 jednostek od siebie. Jeśli zmierzymy odległość między liniami bazowymi obu wierszy, to okaże się, że wynosi ona dokładnie 1 cal czyli 25,4 milimetra.

Metody `getWidth` i `getHeight` pozwalają określić wymiary całej strony. Jednak wydruk z reguły nie odbywa się na całym obszarze strony, ponieważ ograniczony jest marginesami. Nawet jeśli zrezygnujemy z marginesów, to i tak dla większości drukarek pozostaną pewne niewielkie obszary strony, na których nie możemy drukować, co związane jest z koniecznością uchwycenia arkusza papieru przez mechanizm przesuwu papieru drukarki.

Metody `getImageableWidth` i `getImageableHeight` umożliwiają uzyskanie rozmiarów obszaru, na którym możemy drukować. Ponieważ marginesy nie muszą być symetryczne, to istotne są też współrzędne lewego górnego wierzchołka tego obszaru (patrz rysunek 7.33), które możemy uzyskać, korzystając z metod `getImageableX` i `getImageableY`.

**Rysunek 7.33.**  
Rozmiary strony



Jeśli chcemy umożliwić użytkownikowi zmianę rozmiarów marginesów lub orientacji strony, to wywołujemy metodę `pageDialog` klasy `PrinterJob`:

```
PageFormat format = job.pageDialog(attributes);
```



Kontekst graficzny przekazywany metodzie `print` posiada obszar przycięcia wyznaczony przez marginesy strony. Jednak układ współrzędnych strony nadal ma swój początek w lewym górnym narożniku arkusza papieru. Wygodnie więc dokonać przekształcenia układu współrzędnych, tak by jego początek znajdował się w lewym górnym narożniku obszaru strony, na którym możemy drukować. W tym celu na początku metody `print` najlepiej wykonać poniższe wywołanie:

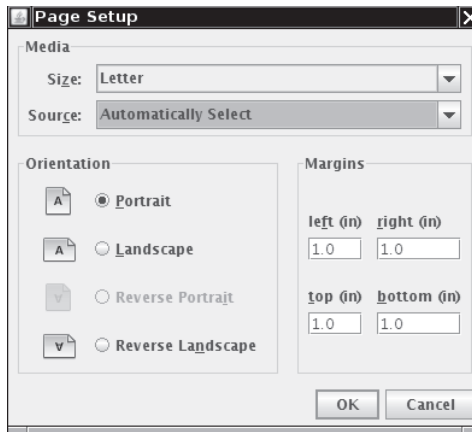
```
g.translate(pageFormat.getImageableX(), pageFormat.getImageableY());
```



Jedna z zakładek okna dialogowego wydruku także umożliwia określenie parametrów strony (rysunek 7.34). Metoda `pageDialog` zwraca obiekt `PageFormat` zawierający atrybuty strony określone przez użytkownika.

### Rysunek 7.34.

Okno dialogowe ustawień strony dla dowolnej platformy



Program przedstawiony na listingach 7.10 i 7.11 tworzy ten sam rysunek na ekranie i na wydruku. Klasa pochodna klasy `JPanel` implementuje interfejs `Printable`. Jej metody `paintComponent` oraz `Print` korzystają z jednej i tej samej metody tworzenia rysunku.

```
class PrintPanel extends JPanel implements Printable
{
    public void paintComponent(Graphics g)
    {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D)g;
        drawPage(g2);
    }

    public int print(Graphics g, PageFormat pf, int page)
        throws PrinterException
    {
        if (page >= 1) return Printable.NO_SUCH_PAGE;
        Graphics2D g2 = (Graphics2D)g;
        g2.translate(pf.getImageableX(), pf.getImageableY());
        drawPage(g2);
        return Printable.PAGE_EXISTS;
    }

    public void drawPage(Graphics2D g2)
```

```

    {
        // kod tworzenia rysunku tak na ekranie, jak i na wydruku
        ...
    }
    ...
}

```

**Listing 7.10.** *print/PrintTestFrame.java*

```

package print;

import java.awt.*;
import java.awt.event.*;
import java.awt.print.*;
import javax.print.attribute.*;
import javax.swing.*;

/**
 * Ramka zawierająca panel z grafiką 2D
 * i przyciski umożliwiające wydruk grafiki
 * oraz określenie formatu strony.
 */
public class PrintTestFrame extends JFrame
{
    private PrintComponent canvas;
    private PrintRequestAttributeSet attributes;

    public PrintTestFrame()
    {
        canvas = new PrintComponent();
        add(canvas, BorderLayout.CENTER);

        attributes = new HashPrintRequestAttributeSet();

        JPanel buttonPanel = new JPanel();
        JButton printButton = new JButton("Print");
        buttonPanel.add(printButton);
        printButton.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                try
                {
                    PrinterJob job = PrinterJob.getPrinterJob();
                    job.setPrintable(canvas);
                    if (job.printDialog(attributes)) job.print(attributes);
                }
                catch (PrinterException e)
                {
                    JOptionPane.showMessageDialog(PrintTestFrame.this, e);
                }
            }
        });

        JButton pageSetupButton = new JButton("Page setup");
        buttonPanel.add(pageSetupButton);
        pageSetupButton.addActionListener(new ActionListener()
        {

```

```
        public void actionPerformed(ActionEvent event)
        {
            PrinterJob job = PrinterJob.getPrinterJob();
            job.pageDialog(attributes);
        }
    });

    add(buttonPanel, BorderLayout.NORTH);
    pack();
}
}
```

---

**Listing 7.11.** *print/PrintComponent.java*

---

```
package print;

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

/**
 * Panel tworzący grafikę 2D na potrzeby prezentacji na ekranie
 * i wydruku.
 */
public class PrintComponent extends JComponent implements Printable
{
    private static final int DEFAULT_WIDTH = 300;
    private static final int DEFAULT_HEIGHT = 300;

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

    public int print(Graphics g, PageFormat pf, int page) throws PrinterException
    {
        if (page >= 1) return Printable.NO_SUCH_PAGE;
        Graphics2D g2 = (Graphics2D) g;
        g2.translate(pf.getImageableX(), pf.getImageableY());
        g2.draw(new Rectangle2D.Double(0, 0, pf.getImageableWidth(),
        ↪pf.getImageableHeight()));

        drawPage(g2);
        return Printable.PAGE_EXISTS;
    }

    /**
     * Metoda tworząca obraz strony, wykorzystująca kontekst
     * graficzny ekranu oraz drukarki.
     * @param g2 kontekst graficzny
     */
    public void drawPage(Graphics2D g2)
    {
        FontRenderContext context = g2.getFontRenderContext();
```

```

Font f = new Font("Serif", Font.PLAIN, 72);
GeneralPath clipShape = new GeneralPath();

TextLayout layout = new TextLayout("Hello", f, context);
AffineTransform transform = AffineTransform.getTranslateInstance(0, 72);
Shape outline = layout.getOutline(transform);
clipShape.append(outline, false);

layout = new TextLayout("World", f, context);
transform = AffineTransform.getTranslateInstance(0, 144);
outline = layout.getOutline(transform);
clipShape.append(outline, false);

g2.draw(clipShape);
g2.clip(clipShape);

final int NLINES = 50;
Point2D p = new Point2D.Double(0, 0);
for (int i = 0; i < NLINES; i++)
{
    double x = (2 * getWidth() * i) / NLINES;
    double y = (2 * getHeight() * (NLINES - 1 - i)) / NLINES;
    Point2D q = new Point2D.Double(x, y);
    g2.draw(new Line2D.Double(p, q));
}

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

```

Program wyświetla i drukuje obrazek pokazany na rysunku 7.20, czyli zbiór linii przycięty obrysem tekstu „Hello, World”.

Wybranie przycisku *Print* umożliwia wydrukowanie rysunku, a przycisku *Page Setup* — określenie parametrów strony. Listing 7.10 zawiera pełen tekst źródłowy programu.



Aby uzyskać okno dialogowe ustawień strony właściwe danemu systemowi, przekazujemy domyślny obiekt klasy `PageFormat` metodzie `pageDialog`. Metoda ta klonuje uzyskany obiekt, modyfikuje go zgodnie z wyborem dokonany przez użytkownika i zwraca klon obiektu.

```

PageFormat defaultFormat = printJob.defaultPage();
PageFormat selectedFormat = printJob.pageDialog(defaultFormat);

```

#### API `java.awt.print.Printable` 1.2

- `int print(Graphics g, PageFormat format, int pageNumber)`  
tworzy stronę i zwraca wartość `PAGE_EXISTS` lub `NO_SUCH_PAGE`.

*Parametry:*

<code>g</code>	kontekst graficzny,
<code>format</code>	format tworzonej strony,
<code>numer</code>	tworzonej strony.

**API** java.awt.print.PrinterJob 1.2

- static PrinterJob getPrinterJob()  
zwraca obiekt klasy PrinterJob.
- PageFormat defaultPage()  
zwraca domyślny format strony dla danej drukarki.
- boolean printDialog(PrintRequestAttributeSet attributes)
- boolean printDialog()

Wyświetlają okno dialogowe wydruku umożliwiające użytkownikowi wybór drukowanych stron i opcji drukowania. Pierwsza z metod wyświetla okno niezależne od platformy, na której pracuje aplikacja, natomiast druga — okno specyficzne dla danej platformy. Pierwsza z metod modyfikuje obiekt `attributes`, tak by odzwierciedlał wybór dokonany przez użytkownika. Obie metody zwracają wartość `true`, jeśli użytkownik zaakceptował wydruk.

- PageFormat pageDialog(PrintRequestAttributeSet attributes)
- PageFormat pageDialog()

Wyświetlają okno dialogowe ustawień strony. Pierwsza z nich wyświetla okno niezależne od platformy, natomiast druga — okno specyficzne dla danej platformy. Obie metody zwracają obiekt klasy `PageFormat` określający format wybrany przez użytkownika. Pierwsza z metod modyfikuje obiekt `attributes`, tak by odzwierciedlał wybór dokonany przez użytkownika. Druga z metod nie modyfikuje obiektu `defaults`.

- void setPrintable(Printable p)
- void setPrintable(Printable p, PageFormat format)

Określają obiekt typu `Printable` wykorzystywany przez dany obiekt klasy `PrinterJob` i opcjonalnie format strony.

- void print()
- void print(PrintRequestAttributeSet attributes)

wywołuje cyklicznie metodę `print` obiektu typu `Printable` i wysyła uzyskane obrazy stron na drukarkę tak długo, dopóki kolejna strona nie jest dostępna.

**API** java.awt.print.PageFormat 1.2

- double getWidth()
- double getHeight()  
Zwracają szerokość i wysokość strony.
- double getImageableWidth()
- double getImageableHeight()

Zwracają szerokość i wysokość obszaru strony, na którym możliwe jest drukowanie.

- `double getImageableX()`
- `double getImageableY()`

Zwracają współrzędne lewego górnego wierzchołka obszaru strony, na którym możliwe jest drukowanie.

- `int getOrientation()`

zwraca jedną z wartości `PORTRAIT`, `LANDSCAPE`, `REVERSE_LANDSCAPE`. Programista nie musi zajmować się opisaniem orientacji strony, ponieważ jest ona określana automatycznie dla formatu strony i kontekstu graficznego drukowania.

## 7.12.2. Drukowanie wielu stron

W praktyce obiektowi klasy `PrinterJob` nie przekazujemy zwykle obiektu implementującego jedynie interfejs `Printable`, a wykorzystujemy raczej obiekt implementujący interfejs `Pageable`. Java dostarcza klasę o nazwie `Book` implementującą interfejs `Pageable`. Obiekt klasy `Book` składa się z rozdziałów, z których każdy implementuje interfejs `Printable`. Obiekt klasy `Book` tworzymy, dodając do niego kolejne rozdziały i liczniki ich stron.

```
Book book = new Book();
Printable coverPage = . . . ;
Printable bodyPages = . . . ;
book.append(coverPage, pageFormat); // dodaje pierwszą stronę
book.append(bodyPages, pageFormat, pageCount);
```

Aby przekazać obiektowi klasy `PrinterJob` obiekt klasy `Book`, korzystamy z metody `setPageable`.

```
printJob.setPageable(book);
```

W ten sposób obiekt klasy `PrinterJob` zna zawsze liczbę stron, które będą wydrukowane. Dzięki temu okno dialogowe wydruku może zaprezentować użytkownikowi właściwy zakres drukowanych stron.



Gdy obiekt klasy `PrinterJob` wywołuje metodę `print` rozdziału reprezentowanego przez obiekt typu `Printable`, to przekazuje jej numer strony w obrębie *całego wydruku*, a nie danego *rozdziału*. Jest to poważny problem, ponieważ jeśli metoda `print` chce wykorzystać przekazany jej numer strony, to musi znać także wartości liczników stron poprzednich rozdziałów.

Z punktu widzenia programisty największy problem związany z wykorzystaniem obiektów klasy `Book` polega na określeniu liczby stron dla każdego z rozdziałów. Klasa implementująca interfejs `Printable` dysponować musi w tym celu *algorytmem rozmieszczenia* tworzonego materiału na kolejnych stronach. Algorytm ten musi zostać wykonany przed rozpoczęciem drukowania w celu ustalenia początku kolejnych stron i ich liczby. Informację tę warto zachować i wykorzystać w procesie tworzenia wydruku.

Pamiętać przy tym należy, że jeśli użytkownik zmieni format strony, to algorytm musi zostać wykonany od nowa, nawet jeśli drukowana informacja nie uległa zmianie.

Program z listingu 7.13 pokazuje sposób tworzenia wydruku wielostronicowego. Drukuje on tekst komunikatu bardzo dużą czcionką, tak że zajmuje on kilka stron (patrz rysunek 7.35). Możemy obciąć marginesy wydrukowanych stron i skleić je razem w transparent.

**Rysunek 7.35.**  
Transparent



Metoda `layoutPages` klasy `Banner` wyznacza rozmieszczenie tekstu na stronach, korzystając z czcionki o rozmiarze 72 punktów. Obliczamy wysokość tekstu i porównujemy ją z wysokością obszaru strony, na którym można drukować. Z porównania tego uzyskujemy współczynnik przeskalowania tekstu, który wykorzystujemy do powiększenia tekstu podczas wydruku.

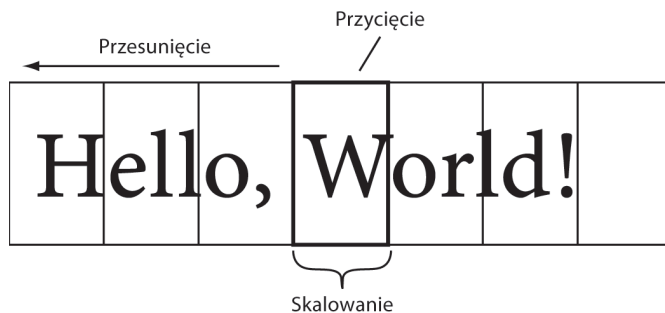


Aby precyzyjnie wyznaczyć rozmieszczenie materiału na stronach, z reguły potrzebne są informacje, które możemy uzyskać z kontekstu graficznego drukowania. Jednak dostęp do niego nie jest możliwy przed rozpoczęciem drukowania. Nasz przykładowy program wyznacza sposób rozmieszczenia tekstu, korzystając z ekranowego kontekstu graficznego w nadziei, że rozmiary czcionek ekranowych i na wydruku nie będą się różnić.

Metoda `getPageCount` klasy `Banner` wywołuje najpierw metodę wyznaczającą rozmieszczenie wydruku. Następnie przeskalowuje szerokość łańcucha znaków i dzieli ją przez szerokość zadrukowywanego obszaru strony. Uzyskany wynik po zaokrągleniu w górę jest liczbą stron wydruku.

Drukowanie transparentu wydawać się może kłopotliwe ze względu na to, że poszczególne znaki mogą być dzielone pomiędzy kolejne strony. Jednak gdy korzystamy z Java 2D, problem ten nie występuje. Kiedy obiekt klasy `PrinterJob` żąda utworzenia wydruku kolejnej strony, posługujemy się po prostu metodą `translate` klasy `Graphics2D`, aby przesunąć lewy górny wierzchołek łańcucha znaków w lewo. Następnie określamy obszar przycięcia jako obszar bieżącej strony (patrz rysunek 7.36) i skalujemy kontekst graficzny za pomocą współczynnika uzyskanego przez metodę wyznaczającą rozmieszczenie tekstu.

**Rysunek 7.36.**  
Drukowanie strony transparentu



Przykład ten stanowi doskonałą ilustrację praktycznej przydatności przekształceń. Dzięki ich wykorzystaniu do umieszczania rysunku w żądanym położeniu kod programu pozostaje prosty i przejrzysty. Operacja przycięcia umożliwia łatwe usunięcie fragmentów rysunku, które nie mieszczą się na danej stronie. W następnym podrozdziale przedstawimy kolejny przykład wykorzystania przekształceń do tworzenia podglądu wydruku.

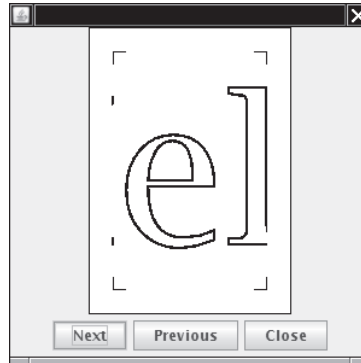


### 7.12.3. Podgląd wydruku

Większość profesjonalnych aplikacji udostępnia mechanizm podglądu wydruku, który pozwala obejrzeć jego strony przed ich wydrukowaniem i uniknąć marnowania papieru, jeśli efekt nie jest zadowalający. Klasy języka Java nie dostarczają wprawdzie gotowego okna dialogowego dla podglądu wydruku, ale jego zaimplementowanie okazuje się dość proste (patrz rysunek 7.37). Program z listingu 7.14 pokazuje, w jaki sposób można to osiągnąć. Co ważne, zdefiniowana w nim klasa `PrintPreviewDialog` jest zupełnie ogólna i może być wykorzystywana do podglądu dowolnych wydruków.

#### Rysunek 7.37.

Okno dialogowe podglądu wydruku



#### Listing 7.12. `book/BookTestFrame.java`

```
package book;

import java.awt.*;
import java.awt.event.*;
import java.awt.print.*;
import javax.print.attribute.*;
import javax.swing.*;

/**
 * Ramka zawierająca pole tekstowe umożliwiające wprowadzenie
 * tekstu transparentu oraz przyciski wydruku, formatu strony i podglądu wydruku.
 */
public class BookTestFrame extends JFrame
{
    private JTextField text;
    private PageFormat pageFormat;
    private PrintRequestAttributeSet attributes;

    public BookTestFrame()
    {
        text = new JTextField();
        add(text, BorderLayout.NORTH);

        attributes = new HashPrintRequestAttributeSet();

        JPanel buttonPanel = new JPanel();

        JButton printButton = new JButton("Print");
```

```

buttonPanel.add(printButton);
printButton.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent event)
    {
        try
        {
            PrinterJob job = PrinterJob.getPrinterJob();
            job.setPageable(makeBook());
            if (job.printDialog(attributes))
            {
                job.print(attributes);
            }
        }
        catch (PrinterException e)
        {
            JOptionPane.showMessageDialog(BookTestFrame.this, e);
        }
    }
});

JButton pageSetupButton = new JButton("Page setup");
buttonPanel.add(pageSetupButton);
pageSetupButton.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent event)
    {
        PrinterJob job = PrinterJob.getPrinterJob();
        pageFormat = job.pageDialog(attributes);
    }
});

JButton printPreviewButton = new JButton("Print preview");
buttonPanel.add(printPreviewButton);
printPreviewButton.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent event)
    {
        PrintPreviewDialog dialog = new PrintPreviewDialog(makeBook());
        dialog.setVisible(true);
    }
});

add(buttonPanel, BorderLayout.SOUTH);
pack();
}

/**
 * Tworzy obiekt klasy Book zawierający stronę okładki
 * i kolejne strony transparentu.
 */
public Book makeBook()
{
    if (pageFormat == null)
    {
        PrinterJob job = PrinterJob.getPrinterJob();
        pageFormat = job.defaultPage();
    }
}

```

```

        Book book = new Book();
        String message = text.getText();
        Banner banner = new Banner(message);
        int pageCount = banner.getPageCount((Graphics2D) getGraphics(), pageFormat);
        book.append(new CoverPage(message + " (" + pageCount + " pages)", pageFormat);
        book.append(banner, pageFormat, pageCount);
        return book;
    }
}

```

---

**Listing 7.13.** *book/Banner.java*


---

```

package book;

import java.awt.*;
import java.awt.font.*;
import java.awt.geom.*;
import java.awt.print.*;

/**
 * Klasa reprezentująca transparent drukowany na wielu stronach.
 */
public class Banner implements Printable
{
    private String message;
    private double scale;

    /**
     * Tworzy obiekt reprezentujący transparent.
     * @param m tekst transparentu
     */
    public Banner(String m)
    {
        message = m;
    }

    /**
     * Zwraca liczbę stron danego rozdziału.
     * @param g2 kontekst graficzny
     * @param pf format strony
     * @return liczba stron
     */
    public int getPageCount(Graphics2D g2, PageFormat pf)
    {
        if (message.equals("")) return 0;
        FontRenderContext context = g2.getFontRenderContext();
        Font f = new Font("Serif", Font.PLAIN, 72);
        Rectangle2D bounds = f.getStringBounds(message, context);
        scale = pf.getImageableHeight() / bounds.getHeight();
        double width = scale * bounds.getWidth();
        int pages = (int) Math.ceil(width / pf.getImageableWidth());
        return pages;
    }

    public int print(Graphics g, PageFormat pf, int page) throws PrinterException
    {
        Graphics2D g2 = (Graphics2D) g;
        if (page > getPageCount(g2, pf)) return Printable.NO_SUCH_PAGE;
    }
}

```

```

        g2.translate(pf.getImageableX(), pf.getImageableY());

        drawPage(g2, pf, page);
        return Printable.PAGE_EXISTS;
    }

    public void drawPage(Graphics2D g2, PageFormat pf, int page)
    {
        if (message.equals("")) return;
        page--; //uwzględnia okładkę

        drawCropMarks(g2, pf);
        g2.clip(new Rectangle2D.Double(0, 0, pf.getImageableWidth(),
        ↪pf.getImageableHeight()));
        g2.translate(-page * pf.getImageableWidth(), 0);
        g2.scale(scale, scale);
        FontRenderContext context = g2.getFontRenderContext();
        Font f = new Font("Serif", Font.PLAIN, 72);
        TextLayout layout = new TextLayout(message, f, context);
        AffineTransform transform = AffineTransform.getTranslateInstance(0,
        ↪layout.getAscent());
        Shape outline = layout.getOutline(transform);
        g2.draw(outline);
    }

    /**
     * Rysuje znaki wyznaczające obszar drukowania
     * w odległości 1/2 cala od narożników strony.
     * @param g2 kontekst graficzny
     * @param pf format strony
     */
    public void drawCropMarks(Graphics2D g2, PageFormat pf)
    {
        final double C = 36; //znak obszaru drukowania = 1/2 cala
        double w = pf.getImageableWidth();
        double h = pf.getImageableHeight();
        g2.draw(new Line2D.Double(0, 0, 0, C));
        g2.draw(new Line2D.Double(0, 0, C, 0));
        g2.draw(new Line2D.Double(w, 0, w, C));
        g2.draw(new Line2D.Double(w, 0, w - C, 0));
        g2.draw(new Line2D.Double(0, h, 0, h - C));
        g2.draw(new Line2D.Double(0, h, C, h));
        g2.draw(new Line2D.Double(w, h, w, h - C));
        g2.draw(new Line2D.Double(w, h, w - C, h));
    }
}

/**
 * Klasa drukująca stronę okładki zawierającą tytuł.
 */
public class CoverPage implements Printable
{
    private String title;

    /**
     * Tworzy stronę okładki.
     * @param t tytuł
     */

```

```

public CoverPage(String t)
{
    title = t;
}

public int print(Graphics g, PageFormat pf, int page) throws PrinterException
{
    if (page >= 1) return Printable.NO_SUCH_PAGE;
    Graphics2D g2 = (Graphics2D) g;
    g2.setPaint(Color.black);
    g2.translate(pf.getImageableX(), pf.getImageableY());
    FontRenderContext context = g2.getFontRenderContext();
    Font f = g2.getFont();
    TextLayout layout = new TextLayout(title, f, context);
    float ascent = layout.getAscent();
    g2.drawString(title, 0, ascent);
    return Printable.PAGE_EXISTS;
}
}

```

---

**Listing 7.14.** *book/PrintPreviewDialog.java*


---

```

package book;

import java.awt.*;
import java.awt.event.*;
import java.awt.font.*;
import java.awt.print.*;
import javax.swing.*;

/**
 * Klasa implementująca uniwersalne okno dialogowe podglądu wydruku.
 */
class PrintPreviewDialog extends JDialog
{
    private static final int DEFAULT_WIDTH = 300;
    private static final int DEFAULT_HEIGHT = 300;

    private PrintPreviewCanvas canvas;

    /**
     * Tworzy okno dialogowe podglądu wydruku.
     * @param p obiekt typu Printable
     * @param pf format strony
     * @param pages liczba stron obiektu p
     */
    public PrintPreviewDialog(Printable p, PageFormat pf, int pages)
    {
        Book book = new Book();
        book.append(p, pf, pages);
        layoutUI(book);
    }

    /**
     * Tworzy okno podglądu wydruku.
     * @param b obiekt klasy Book
     */
    public PrintPreviewDialog(Book b)

```

```
{
    layoutUI(b);
}

/**
 * Rozmieszcza komponenty okna dialogowego.
 * @param book obiekt klasy Book, którego podgląd będzie prezentowany
 */
public void layoutUI(Book book)
{
    setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);

    canvas = new PrintPreviewCanvas(book);
    add(canvas, BorderLayout.CENTER);

    JPanel buttonPanel = new JPanel();

    JButton nextButton = new JButton("Next");
    buttonPanel.add(nextButton);
    nextButton.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent event)
        {
            canvas.flipPage(1);
        }
    });

    JButton previousButton = new JButton("Previous");
    buttonPanel.add(previousButton);
    previousButton.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent event)
        {
            canvas.flipPage(-1);
        }
    });

    JButton closeButton = new JButton("Close");
    buttonPanel.add(closeButton);
    closeButton.addActionListener(new ActionListener()
    {
        public void actionPerformed(ActionEvent event)
        {
            setVisible(false);
        }
    });

    add(buttonPanel, BorderLayout.SOUTH);
}
}
```

---

**Listing 7.15.** *book/PrintPreviewCanvas.java*

---

```
package book;

import java.awt.*;
import java.awt.geom.*;
```

```

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

/**
 * Komponent prezentacji podglądu wydruku.
 */
class PrintPreviewCanvas extends JComponent
{
    private Book book;
    private int currentPage;

    /**
     * Tworzy obiekt prezentacji podglądu wydruku.
     * @param b obiekt klasy Book, którego podgląd będzie prezentowany
     */
    public PrintPreviewCanvas(Book b)
    {
        book = b;
        currentPage = 0;
    }

    public void paintComponent(Graphics g)
    {
        Graphics2D g2 = (Graphics2D) g;
        PageFormat pageFormat = book.getPageFormat(currentPage);

        double xoff; // przesunięcie wzdłuż osi x początku strony w oknie
        double yoff; // przesunięcie wzdłuż osi y początku strony w oknie
        double scale; // współczynnik przeskalowania strony w oknie
        double px = pageFormat.getWidth();
        double py = pageFormat.getHeight();
        double sx = getWidth() - 1;
        double sy = getHeight() - 1;
        if (px / py < sx / sy) // centruje w poziomie
        {
            scale = sy / py;
            xoff = 0.5 * (sx - scale * px);
            yoff = 0;
        }
        else
        // centruje w pionie
        {
            scale = sx / px;
            xoff = 0;
            yoff = 0.5 * (sy - scale * py);
        }
        g2.translate((float) xoff, (float) yoff);
        g2.scale((float) scale, (float) scale);

        // rysuje obraz strony (ignorując marginesy)
        Rectangle2D page = new Rectangle2D.Double(0, 0, px, py);
        g2.setPaint(Color.white);
        g2.fill(page);
        g2.setPaint(Color.black);
        g2.draw(page);

        Printable printable = book.getPrintable(currentPage);
        try

```

```

        {
            printable.print(g2, pageFormat, currentPage);
        }
        catch (PrinterException e)
        {
            g2.draw(new Line2D.Double(0, 0, px, py));
            g2.draw(new Line2D.Double(px, 0, 0, py));
        }
    }

    /**
     * Przesuwa się o zadaną liczbę stron.
     * @param liczba stron. Wartość ujemna oznacza kierunek wstecz.
     */
    public void flipPage(int by)
    {
        int newPage = currentPage + by;
        if (0 <= newPage && newPage < book.getNumberOfPages())
        {
            currentPage = newPage;
            repaint();
        }
    }
}

```

Aby utworzyć obiekt klasy `PrintPreviewDialog`, należy dostarczyć obiekt typu `Printable` lub klasy `Book` wraz z obiektem klasy `PageFormat`. Okno dialogowe klasy `PrintPreviewDialog` zawiera komponent klasy `PrintPreviewCanvas` (patrz listing 7.15). Gdy użytkownik wybiera przyciski podglądu poprzedniej lub następnej strony, to metoda `paintComponent` wywołuje metodę `print` obiektu `Printable` dla żądanej strony.

Zwykle metoda `print` tworzy rysunek strony, korzystając z kontekstu graficznego drukarki. Jednak tym razem dostarczamy jej kontekst graficzny ekranu przeskalowany, tak by obraz strony znalazł się w prostokącie okna dialogowego.

```

float xoff = . . . ; // współrzędna x lewego górnego wierzchołka strony
float yoff = . . . ; // współrzędna y lewego górnego wierzchołka strony
float scale = . . . ; // współczynnik skalowania strony do okna dialogowego
g2.translate(xoff, yoff);
g2.scale(scale, scale);
Printable printable = book.getPrintable(currentPage);
printable.print(g2, pageFormat, currentPage);

```

W ten sposób metoda `print` nie wie nawet, czy tworzy stronę wydruku, czy też jej podgląd na ekranie. Jest to kolejny przykład przemyślanej konstrukcji modelu tworzenia obrazów zastosowanej przez projektantów Java 2D.

Listing 7.12 zawiera kod programu drukującego transparent wyposażonego w okno podglądu wydruku. Umożliwia on wpisanie dowolnej treści transparentu, następnie podejrzenie jego rozkładu na stronach i na tej podstawie zdecydowanie o jego wydruku.

#### API java.awt.print.PrinterJob 1.2

- `void setPageable(Pageable p)`

instaluje drukowany obiekt typu `Pageable` (na przykład klasy `Book`).



## API java.awt.print.Book 1.2

- void append(Printable p, PageFormat format)
- void append(Printable p, PageFormat format, int pageCount)

Dołączają kolejny rozdział do obiektu klasy Book. Jeśli parametr pageCount jest pominięty, to dodawana jest jedna strona.

- Printable getPrintable(int page)  
pobiera obiekt typu Printable dla podanej strony.

## 7.12.4. Usługi drukowania

Dotychczas pokazaliśmy sposoby drukowania grafiki Java 2D. Jednak Java SE 1.4 posiada o wiele szersze możliwości. Definiuje szereg typów danych, dla których możliwe jest odnalezienie odpowiedniej usługi drukowania. Należą do nich między innymi:

- obrazy w formatach GIF, JPEG i PNG,
- dokumenty w formacie tekstowym, HTML, PostScript i PDF,
- dane przekazywane bez interpretacji wprost do drukarki,
- obiekty klas implementujących interfejsy Printable, Pageable i RenderableImage.

Źródłem tych danych może być strumień wejściowy, lokalizacja o podanym adresie URL bądź tablica. *Rodzaj dokumentu* opisuje kombinację źródła danych i ich typu. Klasa DocFlavor definiuje szereg klas wewnętrznych dla różnych źródeł danych. Każda z tych klas definiuje z kolei stałe określające rodzaj dokumentu. Na przykład stała

```
DocFlavor.INPUT_STREAM.GIF
```

opisuje obraz w formacie GIF dostępny za pomocą strumienia wejścia. Tabela 7.3 przedstawia pozostałe kombinacje.

**Tabela 7.3.** Rodzaje dokumentów dla usług drukowania

Źródło danych	Typ danych	Typ MIME
INPUT_STREAM	GIF	image/gif
URL	JPEG	image/jpeg
BYTE_ARRAY	PNG	image/png
	POSTSCRIPT	application/postscript
	PDF	application/pdf
	TEXT_HTML_HOST	text/html (wykorzystujący kodowanie znaków hosta)
	TEXT_HTML_US_ASCII	text/html; charset=us-ascii

Tabela 7.3. Rodzaje dokumentów dla usług drukowania — ciąg dalszy

Źródło danych	Typ danych	Typ MIME
	TEXT_HTML_UTF_8	text/html; charset=utf-8
	TEXT_HTML_UTF_16	text/html; charset=utf-16
	TEXT_HTML_UTF_16LE	text/html; charset=utf-16le (najpierw bajt mniej znaczący)
	TEXT_HTML_UTF_16BE	text/html; charset=utf-16be (najpierw bajt bardziej znaczący)
	TEXT_PLAIN_HOST	text/plain (wykorzystujący kodowanie znaków hosta)
	TEXT_PLAIN_US_ASCII	text/plain; charset=us-ascii
	TEXT_PLAIN_UTF_8	text/plain; charset=utf-8
	TEXT_PLAIN_UTF_16	text/plain; charset=utf-16
	TEXT_PLAIN_UTF_16LE	text/plain; charset=utf-16le (najpierw bajt mniej znaczący)
	TEXT_PLAIN_UTF_16BE	text/plain; charset=utf-16be (najpierw bajt bardziej znaczący)
	PCL	application/vnd.hp_PCL (Hewlett Packard Printer Control Language)
	AUTOSENSE	application/octet-stream (dane wysyłane bez interpretacji wprost do drukarki)
READER	TEXT_HTML	text/html; charset=utf-16
STRING	TEXT_PLAIN	text/plain; charset=utf-16
CHAR_ARRAY		
SERVICE_FORMATTED	PRINTABLE	nie dotyczy
	PAGEABLE	nie dotyczy
	RENDERABLE_IMAGE	nie dotyczy

Załóżmy, że chcemy wydrukować obraz w formacie GIF zapisany w pliku. Musimy najpierw sprawdzić, czy istnieje *usługa drukowania*, która wykona to zadanie. Metoda statyczna `lookupPrintServices` klasy `PrintServiceLookup` zwraca tablicę obiektów typu `PrintService`, które umożliwiają wydruk danego rodzaju dokumentu.

```
DocFlavor flavor = DocFlavor.INPUT_STREAM.GIF;
PrintService[] services
    = PrintServiceLookup.lookupPrintServices(flavor, null);
```

Drugi z parametrów metody `lookupPrintServices` jest wartością `null`, co wskazuje, że nie chcemy zawęzić zakresu poszukiwania przez wskazanie atrybutów drukarki, które omówimy w kolejnym podrozdziale.

Jeśli tablica zwrócona przez metodę `lookupPrintServices` zawiera więcej niż jeden element, musimy zdecydować, którą z usług chcemy wykorzystać. Metoda `getName` klasy `PrintService` zwraca nazwę drukarki, która posłużyć może jako kryterium wyboru.

Następnie uzyskujemy dla danej usługi obiekt klasy `DocPrintJob`:

```
DocPrintJob job = services[i].createPrintJob();
```

Do drukowania musimy dostarczyć obiekt implementujący interfejs `Doc`. Java udostępnia klasę `SimpleDoc` implementującą ten interfejs. Konstruktor klasy `SimpleDoc` wymaga, aby jako parametrów obiektu reprezentującego źródło danych użyć rodzaju dokumentu `i`, opcjonalnie, zbioru atrybutów. Na przykład:

```
InputStream in = new FileInputStream(fileName);
Doc doc = new SimpleDoc(in, flavor, null);
```

Możemy teraz uruchomić proces drukowania:

```
job.print(doc, null)
```

Podobnie jak wcześniej, wartość `null` może zostać zastąpiona zbiorem atrybutów.

Zwróćmy uwagę, że proces drukowania przebiega teraz inaczej, niż zostało to opisane w poprzednim podrozdziale, ponieważ nie ma miejsca żadna interakcja z użytkownikiem wykorzystująca okna dialogowe. Możemy za to zaimplementować na przykład mechanizm wydruku na serwerze, któremu użytkownicy będą przekazywać prace do wydruku za pośrednictwem formularza na stronie internetowej.

Program z listingu 7.16 demonstruje sposób wykorzystania usług drukowania na przykładzie wydruku pliku graficznego.

#### Listing 7.16. `printService/PrintServiceTest.java`

```
package printService;

import java.io.*;
import java.nio.file.*;
import javax.print.*;

/**
 * Program demonstrujący wykorzystanie usług drukowania.
 * Drukuje obrazek w formacie GIF za pomocą jednej z usług
 * dostępnych dla tego rodzaju dokumentu (wybranej przez użytkownika).
 * @version 1.10 2007-08-16
 * @author Cay Horstmann
 */
public class PrintServiceTest
{
    public static void main(String[] args)
    {
        DocFlavor flavor = DocFlavor.URL.GIF;
```

```

PrintService[] services = PrintServiceLookup.lookupPrintServices(flavor, null);
if (args.length == 0)
{
    if (services.length == 0) System.out.println("No printer for flavor " + flavor);
    else
    {
        System.out.println("Specify a file of flavor " + flavor
            + "\nand optionally the number of the desired printer.");
        for (int i = 0; i < services.length; i++)
            System.out.println((i + 1) + ": " + services[i].getName());
    }
    System.exit(0);
}
String fileName = args[0];
int p = 1;
if (args.length > 1) p = Integer.parseInt(args[1]);
if (fileName == null) return;
try (InputStream in = Files.newInputStream(Paths.get(fileName)))
{
    Doc doc = new SimpleDoc(in, flavor, null);
    DocPrintJob job = services[p - 1].createPrintJob();
    job.print(doc, null);
}
catch (Exception ex)
{
    ex.printStackTrace();
}
}
}

```

**API** `javax.print.PrintServiceLookup` **1.4**

- `PrintService[] lookupPrintServices(DocFlavor flavor, AttributeSet attributes)`

wyszukuje usługi drukowania dla danego rodzaju dokumentu i zbioru atrybutów.

*Parametry:* `flavor`      rodzaj dokumentu,  
                   `attributes`    wymagane atrybuty drukowania lub wartość `null`.

**API** `javax.print.PrintService` **1.4**

- `DocPrintJob createPrintJob()`

zwraca obiekt `DocPrintJob` umożliwiający wydruk obiektu implementującego interfejs `Doc`, na przykład klasy `SimpleDoc`.

**API** `javax.print.DocPrintJob` **1.4**

- `void print(Doc doc, PrintRequestAttributeSet attributes)`

drukuje dokument o podanych atrybutach.

*Parametry:* `doc`              drukowany dokument,  
                   `attributes`    wymagane atrybuty drukowania lub wartość `null`.

**API** javax.print.SimpleDoc 1.4

- SimpleDoc(Object data, DocFlavor flavor, DocAttributeSet attributes)

tworzy obiekt klasy SimpleDoc, który może być wydrukowany za pomocą obiektu klasy DocPrintJob.

*Parametry:*

data	obiekt zawierający dane do wydruku, na przykład strumień wejściowy lub obiekt typu Printable,
flavor	rodzaj dokumentu,
attributes	wymagane atrybuty drukowania lub wartość null.

## 7.12.5. Usługi drukowania za pośrednictwem strumieni

Usługa drukowania tworzy wydruk, wysyłając dane do drukarki. Usługa drukowania za pośrednictwem strumienia tworzy te same dane, ale zamiast wysyłać je do drukarki, umieszcza je w strumieniu, aby odroczyć proces drukowania lub umożliwić zinterpretowanie danych wydruku innemu programowi. Platforma Java udostępnia usługę drukowania za pośrednictwem strumienia, która umożliwia utworzenie danych w formacie PostScript dla obrazów i grafiki 2D. Usługa ta dostępna jest w każdym systemie, nawet jeśli nie posiada on zainstalowanej żadnej drukarki.

Znalezienie odpowiedniej usługi drukowania za pośrednictwem strumienia jest nieco bardziej pracochłonne niż znalezienie zwykłej usługi drukowania. Musimy określić rodzaj drukowanego dokumentu i typ MIME dla strumienia wyjściowego. W rezultacie otrzymamy tablicę StreamPrintServiceFactory zawierającą fabryki.

```
DocFlavor flavor = DocFlavor.SERVICE_FORMATTED.PRINTABLE;
String mimeType = "application/postscript";
StreamPrintServiceFactory[] factories =
    StreamPrintServiceFactory.lookupStreamPrintServiceFactories(flavor, mimeType);
```

Klasa StreamPrintServiceFactory nie posiada żadnej metody, która umożliwiłaby rozróżnienie poszczególnych fabryk, dlatego wybierzemy po prostu pierwszą z nich czyli factories[0]. Wywołamy następnie jej metodę getPrintService z parametrem w postaci strumienia wyjściowego, aby uzyskać obiekt klasy StreamPrintService.

```
OutputStream out = new FileOutputStream(fileName);
StreamPrintService service = factories[0].getPrintService(out);
```

Klasa StreamPrintService jest klasą pochodną klasy PrintService. Aby uzyskać wydruk, należy następnie powtórzyć kroki podane w poprzednim rozdziale.

**API** javax.print.StreamPrintServiceFactory 1.4

- StreamPrintServiceFactory[] lookupStreamPrintServiceFactories(DocFlavor flavor, String mimeType)

wyszukuje fabryki usług drukowania umożliwiające wydruk danego rodzaju dokumentów za pomocą strumienia wyjściowego podanego typu MIME.

- `StreamPrintService` `getPrintService(OutputStream out)`

zwraca usługę drukowania, która wysyła dane do podanego strumienia wyjściowego.

## 7.12.6. Atrybuty drukowania

Usługi drukowania dostarczają bogatego zestawu interfejsów i klas umożliwiających wyspecyfikowanie różnego rodzaju atrybutów. Możemy wyróżnić cztery najistotniejsze grupy atrybutów. Dwie pierwsze pozwalają wyspecyfikować żądania wysyłane do drukarki.

- *Atrybuty żądań wydruku* wykorzystywane są do określenia opcji wydruku (takich jak na przykład druk dwustronny czy format papieru) dla wszystkich dokumentów danego wydruku.
- *Atrybuty dokumentów* dotyczące poszczególnych dokumentów w ramach danego wydruku.

Pozostałe dwie grupy atrybutów dotyczą informacji o drukarce i stanie wydruku.

- *Atrybuty usługi drukowania* zawierają informacje o usłudze drukowania, takie jak marka i model drukarki i określenie, czy przyjmuje ona zlecenia wydruku.
- *Atrybuty wydruku* zawierają informacje o stanie określonego wydruku (na przykład, czy został on wykonany).

Do opisu różnych atrybutów służy interfejs `Attribute` wraz z następującymi interfejsami pochodnymi:

```
PrintRequestAttribute
DocAttribute
PrintServiceAttribute
PrintJobAttribute
SupportedValuesAttribute
```

Klasy poszczególnych atrybutów implementują jeden lub więcej wymienionych interfejsów. Na przykład obiekt klasy `Copies` używany do opisu liczby kopii wydruku implementuje interfejsy `PrintRequestAttribute` i `PrintJobAttribute`. W przypadku pierwszego z nich oznacza to, że żądanie wydruku może określać liczbę kopii. Natomiast implementacja drugiego z interfejsów oznacza, że wydruk zawiera określoną liczbę kopii, która może być mniejsza od żądanej ze względu na ograniczone możliwości drukarki lub brak papieru.

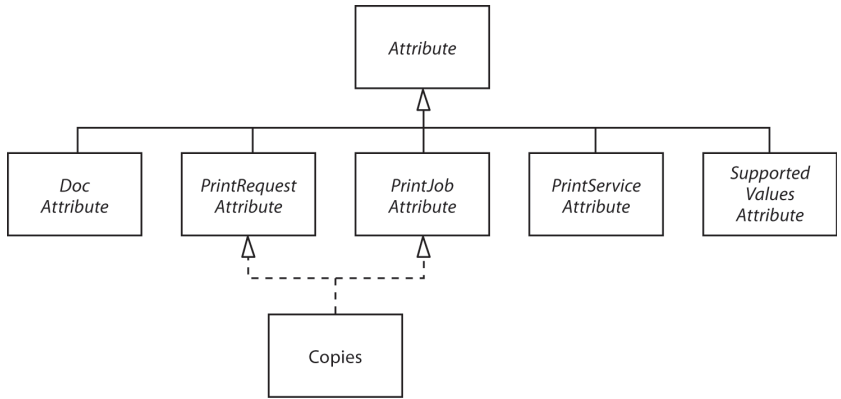
Implementacja przez atrybut interfejsu `SupportedValuesAttribute` wskazuje, że wartość atrybutu nie jest związana z określonym żądaniem lub stanem, ale określa potencjalne możliwości usługi. Istnieje na przykład klasa `CopiesSupported` implementująca interfejs `SupportedValuesAttribute`. Obiekt tej klasy podaje, ile kopii może wydrukować dana drukarka.

Rysunek 7.38 prezentuje diagram hierarchii atrybutów.

Oprócz klas i interfejsów reprezentujących pojedyncze atrybuty zdefiniowano także klasy i interfejsy dla zbiorów atrybutów. Interfejs `AttributeSet` posiada cztery interfejsy pochodne:

```
PrintRequestAttributeSet
DocAttributeSet
PrintServiceAttributeSet
PrintJobAttributeSet
```

**Rysunek 7.38.**  
Hierarchia atrybutów

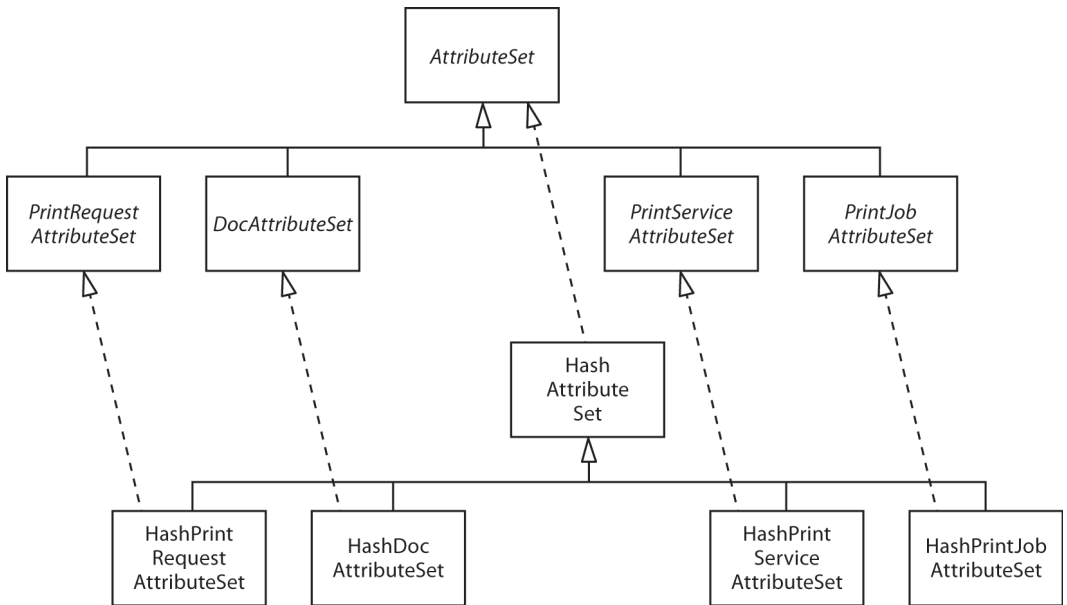


Dla każdego z tych interfejsów dostępna jest klasa implementacji:

```

HashAttributeSet
HashPrintRequestAttributeSet
HashDocAttributeSet
HashPrintServiceAttributeSet
HashPrintJobAttributeSet
  
```

Na rysunku 7.39 przedstawiony został diagram klas dla hierarchii zbiorów atrybutów.



**Rysunek 7.39.** Hierarchia zbiorów atrybutów

Na przykład zbiór atrybutów żądania wydruku tworzymy w następujący sposób:

```

PrintRequestAttributeSet attributes
    = new HashPrintRequestAttributeSet();
  
```

Po utworzeniu zbioru atrybutów nie musimy już posługiwać się przedrostkiem Hash.

W jakim celu utworzono tyle interfejsów? Umożliwiają one kontrolę poprawności użycia atrybutów. Na przykład interfejs `DocAttributeSet` akceptuje jedynie obiekty implementujące interfejs `DocAttribute`. Próba dodania innego atrybutu kończy się błędem wykonania programu.

Zbiór atrybutów jest specjalizowaną odmianą mapy, której klucze są typu `Class`, a wartości należą do klasy implementującej interfejs `Attribute`. Na przykład jeśli w zbiorze atrybutów umieścimy obiekt

```
new Copies(10)
```

to jego kluczem będzie obiekt `Copies.class`. Klucz ten nazywany jest *kategorią* atrybutu. Interfejs `Attribute` definiuje metodę

```
Class getCategory()
```

która zwraca kategorię atrybutu. Klasa `Copies` implementuje tę metodę, tak by zwracała ona obiekt `Copies.class`. Nie jest jednak wymagane, by kategoria atrybutu była równoznaczna z jego klasą.

Gdy umieszczamy atrybut w zbiorze atrybutów, to informacja o jego kategorii pobierana jest automatycznie. Wystarczy więc jedynie umieścić atrybut w zbiorze, jak pokazano poniżej:

```
attributes.add(new Copies(10));
```

Jeśli następnie dodamy do zbioru inny atrybut tej samej kategorii, to zastąpi on poprzedni atrybut.

Aby pobrać atrybut, musimy użyć jego kategorii jako klucza, na przykład:

```
AttributeSet attributes = job.getAttributes();  
Copies copies = (Copies)attribute.get(Copies.class);
```

Atrybuty są także zorganizowane wokół wartości, które mogą przyjmować. Atrybut klasy `Copies` może posiadać dowolną wartość całkowitą i dlatego jego klasa jest pochodną klasy `IntegerSyntax`, podobnie jak klasy innych atrybutów przyjmujących wartości całkowite. Metoda `getValue` umożliwia pobranie wartości atrybutu:

```
int n = copies.getValue();
```

Istnieją także klasy

```
TextSyntax  
DateTimeSyntax  
URISyntax
```

hermetyzujące odpowiednio łańcuch znaków, datę i czas lub identyfikator URI (*Uniform Resource Identifier*).

Istnieje także wiele atrybutów, które mogą przyjmować określoną liczbę wartości. Na przykład atrybut `PrintQuality` umożliwia określenie jakości wydruku jako roboczej, zwykłej i wysokiej za pomocą następujących stałych:

```
PrintQuality.DRAFT  
PrintQuality.NORMAL  
PrintQuality.HIGH
```



Klasy atrybutów przyjmujących skończoną liczbę wartości są pochodnymi klasy `EnumSyntax`, która posiada szereg metod umożliwiających zorganizowanie wyliczeń wartości atrybutów. Dzięki temu, korzystając z atrybutu, nie musimy sami tworzyć odpowiedniego mechanizmu do posługiwania się jego wartościami. Wystarczy gdy podamy jedynie nazwę wartości, umieszczając atrybut w zbiorze:

```
attributes.add(PrintQuality.HIGH);
```

Poniżej przedstawiamy sposób sprawdzenia wartości atrybutu.

```
if (attribute.get(PrintQuality.class) == PrintQuality.HIGH)
    . . .
```

Tabela 7.4 prezentuje atrybuty drukowania. Jej druga kolumna zawiera nazwę klasy bazowej dla klasy atrybutu (na przykład `IntegerSyntax` dla `Copies`) lub zbiór wartości w przypadku atrybutów, dla których jest on skończony. Ostatnie cztery kolumny pokazują, czy klasa atrybutu implementuje interfejs `DocAttribute` (DA), `PrintJobAttribute` (PJA), `PrintRequestAttribute` (PRA) i `PrintServiceAttribute` (PSA).

**Tabela 7.4.** Atrybuty drukowania

Atrybut	Klasa bazowa lub wyliczenie wartości atrybutu	DA	PJA	PRA	PSA
Chromaticity	MONOCHROME, COLOR	✓	✓	✓	
ColorSupported	SUPPORTED, NOT_SUPPORTED				✓
Compression	COMPRESS, DEFLATE, GZIP, NONE	✓			
Copies	<code>IntegerSyntax</code>		✓	✓	
DateTimeAtCompleted	<code>DateTimeSyntax</code>		✓		
DateTimeAtCreation	<code>DateTimeSyntax</code>		✓		
DateTimeAtProcessing	<code>DateTimeSyntax</code>		✓		
Destination	<code>UriSyntax</code>		✓	✓	
DocumentName	<code>TextSyntax</code>	✓			
Fidelity	FIDELITY_TRUE, FIDELITY_FALSE		✓	✓	
Finishings	NONE, STAPLE, EDGE_STITCH, BIND, SADDLE_STITCH, COVER, . . .	✓	✓	✓	
JobHoldUntil	<code>DateTimeSyntax</code>		✓	✓	
JobImpressions	<code>IntegerSyntax</code>		✓	✓	
JobImpressionsCompleted	<code>IntegerSyntax</code>		✓		
JobKOctets	<code>IntegerSyntax</code>		✓	✓	
JobKOctetsProcessed	<code>IntegerSyntax</code>		✓		
JobMediaSheets	<code>IntegerSyntax</code>		✓	✓	
JobMediaSheetsCompleted	<code>IntegerSyntax</code>		✓		
JobMessageFromOperator	<code>TextSyntax</code>		✓		
JobName	<code>TextSyntax</code>		✓	✓	

Tabela 7.4. Atrybuty drukowania — ciąg dalszy

Atrybut	Klasa bazowa lub wyliczenie wartości atrybutu	DA	PJA	PRA	PSA
JobOriginatingUserName	TextSyntax		✓		
JobPriority	IntegerSyntax		✓	✓	
JobSheets	STANDARD, NONE		✓	✓	
JobState	ABORTED, CANCELED, COMPLETED, PENDING, PENDING_HELD, PROCESSING, PROCESSING_STOPPED		✓		
JobStateReason	ABORTED_BY_SYSTEM, DOCUMENT_FORMAT_ERROR i wiele innych				
JobStateReasons	HashSet		✓		
MediaName	ISO_A4_WHITE, ISO_A4_TRANSPARENT, NA_LETTER_WHITE, NA_LETTER_TRANSPARENT	✓	✓	✓	
MediaSize	ISO.A0 – ISO.A10, ISO.B0 – ISO.B10, ISO.C0 – ISO.C10, NA.LETTER, NA.LEGAL oraz inne rozmiary papieru i kopert				
MediaSizeName	ISO_A0 – ISO_A10, ISO_B0 – ISO_B10, ISO_C0 – ISO_C10, NA_LETTER, NA_LEGAL oraz inne nazwy rozmiarów papieru i kopert	✓	✓	✓	
MediaTray	TOP, MIDDLE, BOTTOM, SIDE, ENVELOPE, LARGE_CAPACITY, MAIN, MANUAL	✓	✓	✓	
MultipleDocumentHandling	SINGLE_DOCUMENT, SINGLE_DOCUMENT_NEW_SHEET, SEPARATE_DOCUMENTS_COLLATED_COPIES, SEPARATE_DOCUMENTS_UNCOLLATED_COPIES		✓	✓	
NumberOfDocuments	IntegerSyntax		✓		
NumberOfInterveningJobs	IntegerSyntax		✓		
NumberUp	IntegerSyntax	✓	✓	✓	
OrientationRequested	PORTRAIT, LANDSCAPE, REVERSE_PORTRAIT, REVERSE_LANDSCAPE	✓	✓	✓	
OutputDeviceAssigned	TextSyntax		✓		
PageRanges	SetOfInteger	✓	✓	✓	
PagesPerMinute	IntegerSyntax				✓
PagesPerMinuteColor	IntegerSyntax				✓
PDLOverrideSupported	ATTEMPTED, NOT_ATTEMPTED				✓
PresentationDirection	TORIGHT_TOBOTTOM, TORIGHT_TOTOP, TOBOTTOM_TORIGHT, TOBOTTOM_TOLEFT, TOLEFT_TOBOTTOM, TOLEFT_TOTOP, TOTOP_TORIGHT, TOTOP_TOLEFT		✓	✓	
PrinterInfo	TextSyntax				✓
PrinterIsAcceptingJobs	ACCEPTING_JOBS, NOT_ACCEPTING_JOBS				✓

Tabela 7.4. Atrybuty drukowania — ciąg dalszy

Atrybut	Klasa bazowa lub wyliczenie wartości atrybutu	DA	PJA	PRA	PSA
PrinterLocation	TextSyntax				✓
PrinterMakeAndModel	TextSyntax				✓
PrinterMessageFromOperator	TextSyntax				✓
PrinterMoreInfo	UriSyntax				✓
PrinterMoreInfoManufacturer	UriSyntax				✓
PrinterName	TextSyntax				✓
PrinterResolution	ResolutionSyntax	✓	✓	✓	
PrinterState	PROCESSING, IDLE, STOPPED, UNKNOWN				✓
PrinterStateReason	COVER_OPEN, FUSER_OVER_TEMP, MEDIA_JAM i wiele innych				
PrinterStateReasons	HashMap				
PrinterURI	UriSyntax				✓
PrintQuality	DRAFT, NORMAL, HIGH	✓	✓	✓	
QueuedJobCount	IntegerSyntax				✓
ReferenceUriSchemesSupported	FILE, FTP, GOPHER, HTTP, HTTPS, NEWS, NNTP, WAIS				
RequestingUserName	TextSyntax			✓	
Severity	ERROR, REPORT, WARNING				
SheetCollate	COLLATED, UNCOLLATED	✓	✓	✓	
Sides	ONE_SIDED, DUPLEX (=TWO_SIDED_LONG_EDGE), TUMBLE (=TWO_SIDED_SHORT_EDGE)	✓	✓	✓	



Istnieje wiele wyspecjalizowanych atrybutów drukowania. Większość z nich związana jest z protokołem Internet Printing Protocol 1.1 (RFC 2911).



We wcześniejszych wersjach JDK wprowadzono klasy `JobAttributes` i `PageAttribu` `tes`, których przeznaczenie podobne jest do atrybutów omówionych w tym rozdziale. Obecnie klasy te są uważane za przestarzałe.

#### API `javax.print.attribute.Attribute` 1.4

- `Class getCategory()`  
zwraca kategorię atrybutu.
- `String getName()`  
zwraca nazwę atrybutu.

**API** `javax.print.attribute.AttributeSet` 1.4

- `boolean add(Attribute attr)`  
dodaje atrybut do zbioru atrybutów. Jeśli zbiór zawiera już atrybut tej samej kategorii, to zostanie on zastąpiony. Zwraca wartość `true`, jeśli zbiór uległ zmianie na skutek wykonania metody.
- `Attribute get(Class category)`  
pobiera atrybut, którego kluczem jest podana kategoria lub zwraca wartość `null`, jeśli taki atrybut nie istnieje w zbiorze.
- `boolean remove(Attribute attr)`
- `boolean remove(Class category)`  
usuwa ją ze zbioru atrybut lub atrybut danej kategorii. Zwraca wartość `true`, jeśli zbiór uległ zmianie na skutek wykonania metody.
- `Attribute[] toArray()`  
zwraca tablicę zawierającą wszystkie atrybuty danego zbioru.

**API** `javax.print.PrintService` 1.4

- `PrintServiceAttributeSet getAttributes()`  
zwraca atrybuty danej usługi drukowania.

**API** `javax.print.DocPrintJob` 1.4

- `PrintJobAttributeSet getAttributes()`  
zwraca atrybuty danego wydruku.

Na tym kończymy omówienie zagadnień związanych z drukowaniem. Przedstawiliśmy sposób drukowania grafiki 2D i innych rodzajów dokumentów, system wyszukiwania drukarek i usług drukowania za pośrednictwem strumieni oraz metody korzystania z atrybutów drukowania. W dalszej części rozdziału omówimy zagadnienia związane z wykorzystaniem schowka oraz mechanizmu „przeciągnij i upuść”.

## 7.13. Schowek

Mechanizm „wytnij i wklej” jest jednym z najwygodniejszych i bardziej przydatnych elementów graficznych interfejsów użytkownika (takich jak X Window czy Windows). Dzięki niemu możemy zaznaczyć pewne dane programu i wyciąć je lub skopiować do schowka. Następnie zawartość schowka możemy wkleić w zupełnie innym programie. Korzystając z pośrednictwa schowka, możemy przekazywać tekst, grafikę i inne dane z jednego dokumentu do innego, a także z jednego miejsca do innego miejsca w obrębie tego samego dokumentu. Mechanizm ten jest tak naturalny, że większość użytkowników nie zastanawia się nawet nad sposobem jego działania.

Mimo że koncepcja schowka jest bardzo prosta, to jego implementacja jest trudniejsza, niż mogłoby się wydawać. Załóżmy, że do schowka skopiowaliśmy tekst z edytora. Jeśli będziemy chcieli skopiować go do innego dokumentu, to oczywiście informacja o sposobie sformatowania tekstu powinna zostać zachowana. Natomiast umieszczając tekst ze schowka w zwykłym pliku tekstowym, oczekujemy jedynie wstawienia samych znaków tekstu bez dodatkowych kodów formatowania. Aby takie różne zachowania schowka były możliwe, dane muszą być do niego dostarczane w wielu formatach, z których odbiorca wybiera jeden.

Implementacje schowka w systemach Microsoft Windows i Macintosh posiadają podobne możliwości, ale występują też między nimi pewne różnice. Natomiast schowek systemu X Window posiada zdecydowanie mniejsze możliwości — wycinanie i wklejanie danych innych niż zwykły tekst jest rzadko dostępne. Ograniczenia te należy uwzględnić, wypróbowując działanie przykładowych programów zamieszczonych w dalszej części rozdziału.



Informacje o tym, jakiego rodzaju obiekty mogą być przekazywane między schowkiem systemowym a aplikacjami Java, można odnaleźć w pliku `jre/lib/awt.properties`.

Często zdarza się, że programy muszą umożliwiać kopiowanie i wklejanie takich typów danych, których nie obsługuje schowek systemowy. Java umożliwia przekazywanie referencji do dowolnych lokalnych obiektów w obrębie tej samej maszyny wirtualnej. Pomiedzy różnymi maszynami wirtualnymi można przekazywać serializowane obiekty oraz referencje obiektów zdalnych.

Tabela 7.5 podsumowuje możliwości przekazywania danych za pośrednictwem schowka.

**Tabela 7.5.** *Możliwości przekazywania danych na platformie Java*

Rodzaj transferu danych	Typ danych
Pomiędzy programem w języku Java a programem w kodzie macierzystym maszyny.	Tekst, grafika, listy plików, ... (w zależności od platformy).
Pomiędzy dwoma programami w języku Java.	Obiekty serializowane i obiekty zdalne.
W obrębie jednego programu w języku Java.	Dowolne obiekty.

## 7.13.1. Klasy i interfejsy umożliwiające przekazywanie danych

Pakiet `java.awt.datatransfer` zawiera implementację mechanizmu przekazywania danych. Poniżej zamieszczamy przegląd najważniejszych dostępnych w nim klas i interfejsów.

- Obiekty przekazywane za pomocą schowka muszą implementować interfejs `Transferable`.
- Klasa `Clipboard` reprezentuje schowek. W schowku mogą być umieszczane lub pobierane jedynie obiekty implementujące interfejs `Transferable`. Schowek systemowy reprezentowany jest przez obiekt klasy `Clipboard`.
- Klasa `DataFlavor` opisuje rodzaje danych, które mogą być umieszczane w schowku.

- Klasa `StringSelection` jest klasą konkretną implementującą interfejs `Transferable`. Jest ona wykorzystywana do przekazywania łańcuchów znaków.
- Klasa musi implementować interfejs `ClipboardOwner`, jeśli chce być powiadamiana o tym, że zawartość schowka została nadpisana przez inny obiekt. Posiadanie schowka umożliwia odroczenie formatowania skomplikowanych danych. Jeśli program wysyła proste dane, na przykład łańcuch danych, to wystarczy, że umieści go w schowku i może przejść do wykonywania innych zadań. Jeśli jednak program przekazuje za pomocą schowka skomplikowane dane, które mogą być dostępne w różnych formatach, to nie ma sensu, by umieszczał je wszystkie w schowku, dopóki nie będą rzeczywiście potrzebne. Musi wtedy jednak nasłuchiwać żądań pobrania zawartości schowka. O zmianie zawartości schowka zostaje powiadomiony przez wywołanie metody `lostOwnership`, co oznacza, że dane nie będą już potrzebne. W naszych programach ilustrujących działanie schowka nie będziemy wykorzystywać możliwości posiadania schowka.

## 7.13.2. Przekazywanie tekstu

Najlepszym sposobem na zapoznanie się ze sposobem korzystania z klas mechanizmu „wynij-wklej” jest najprostszy przypadek przekazywania tekstu do schowka i z niego. Na początek musimy uzyskać referencję schowka systemowego.

```
Clipboard clipboard =
    Toolkit.getDefaultToolkit().getSystemClipboard();
```

Aby przekazywać łańcuchy znaków za pomocą schowka, należy je obudować obiektami klasy `StringSelection`. Konstruktorowi tej klasy przekazujemy obudowywany łańcuch.

```
String text = . . . ;
StringSelection selection = new StringSelection(text);
```

Przekazanie obiektu do schowka odbywa się za pomocą metody `setContents`, której parametrami są obiekty klasy `StringSelection` i `ClipboardOwner`. Jeśli nie chcemy wyznaczać posiadacza schowka, to jako drugi z parametrów metody przekazujemy wartość `null`.

```
clipboard.setContents(selection, null);
```

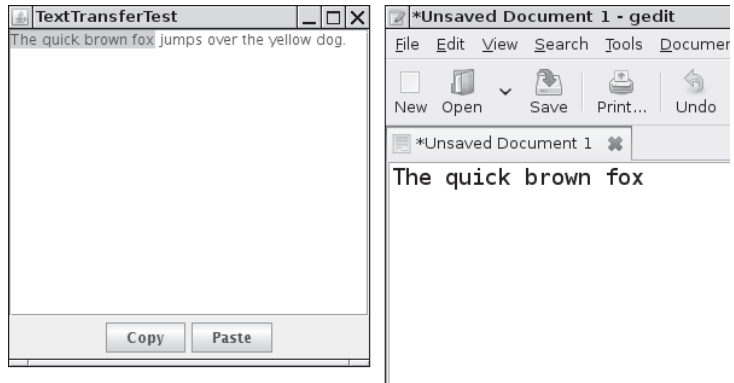
Przyjrzyjmy się operacji odwrotnej, polegającej na odczycie łańcucha znaków ze schowka:

```
DataFlavor flavor = DataFlavor.stringflavor;
if (clipboard.isDataFlavorAvailable(flavor))
    String text = (String) clipboard.getData(flavor);
```

Program, którego tekst źródłowy zawiera listing 7.17, demonstruje sposób korzystania ze schowka systemowego w aplikacjach Java. Wybranie tekstu w oknie programu, a następnie przycisku *Copy* powoduje umieszczenie tekstu w schowku. Rysunek 7.40 pokazuje, że uzyskaną w ten sposób zawartość schowka możemy następnie skopiować do dowolnego edytora tekstu. I na odwrót, jeśli skopiujemy tekst z edytora, możemy wkleić go w naszym przykładowym programie.

**Rysunek 7.40.**

Program  
TextTransferTest  
w działaniu

**Listing 7.17.** *transferText/TextTransferFrame.java*

```
package transferText;

import java.awt.*;
import java.awt.datatransfer.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;

/**
 * Ramka zawierająca obszar tekstowy
 * oraz przyciski umożliwiające kopiowanie do schowka
 * i wklejanie ze schowka.
 */
public class TextTransferFrame extends JFrame
{
    private JTextArea textArea;
    private static final int TEXT_ROWS = 20;
    private static final int TEXT_COLUMNS = 60;

    public TextTransferFrame()
    {
        textArea = new JTextArea(TEXT_ROWS, TEXT_COLUMNS);
        add(new JScrollPane(textArea), BorderLayout.CENTER);
        JPanel panel = new JPanel();

        JButton copyButton = new JButton("Copy");
        panel.add(copyButton);
        copyButton.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                copy();
            }
        });

        JButton pasteButton = new JButton("Paste");
        panel.add(pasteButton);
        pasteButton.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent event)

```

```

        {
            paste();
        }
    }):

    add(panel, BorderLayout.SOUTH);
    pack();
}

/**
 * Kopiuje wybrany tekst do schowka systemowego.
 */
private void copy()
{
    Clipboard clipboard = Toolkit.getDefaultToolkit().getSystemClipboard();
    String text = textArea.getSelectedText();
    if (text == null) text = textArea.getText();
    StringSelection selection = new StringSelection(text);
    clipboard.setContents(selection, null);
}

/**
 * Wkleja tekst ze schowka systemowego.
 */
private void paste()
{
    Clipboard clipboard = Toolkit.getDefaultToolkit().getSystemClipboard();
    DataFlavor flavor = DataFlavor.stringFlavor;
    if (clipboard.isDataFlavorAvailable(flavor))
    {
        try
        {
            String text = (String) clipboard.getData(flavor);
            textArea.replaceSelection(text);
        }
        catch (UnsupportedFlavorException e)
        {
            JOptionPane.showMessageDialog(this, e);
        }
        catch (IOException e)
        {
            JOptionPane.showMessageDialog(this, e);
        }
    }
}
}

```

---

**API** java.awt.Toolkit 1.0

- Clipboard `getSystemClipboard()` 1.1  
zwraca obiekt reprezentujący schowek systemowy.

**API** java.awt.datatransfer.Clipboard 1.1

- Transferable `getContents(Object requester)`



zwraca zawartość schowka.

*Parametry:* requester obiekt pobierający zawartość schowka.  
W rzeczywistości nie jest używany.

- void setContents(Transferable contents, ClipboardOwner owner)  
umieszcza dane w schowku.

*Parametry:* contents obiekt typu Transferable hermetyzujący dane umieszczane w schowku,  
owner obiekt, który będzie powiadamiany (przez wywołanie jego metody lostOwnership), gdy w schowku umieszczone zostaną nowe dane lub wartość null, jeśli powiadamianie nie jest potrzebne.

- boolean isDataFlavorAvailable(DataFlavor flavor) 5.0  
zwraca wartość true, jeśli schowek zawiera dane w podanym formacie.
- Object getData(DataFlavor flavor) 5.0  
zwraca dane w określonym formacie lub zgłasza wyjątek UnsupportedOperationException, gdy dane w tym formacie nie są dostępne.

#### API `java.awt.datatransfer.ClipboardOwner` 1.1

- void lostOwnership(Clipboard clipboard, Transferable contents)  
powiadamia obiekt, że nie jest on już posiadaczem zawartości schowka.  
*Parametry:* clipboard schowek, w którym obiekt umieścił dane,  
contents obiekt, który został umieszczony w schowku.

#### API `java.awt.datatransfer.Transferable` 1.1

- boolean isDataFlavorSupported(DataFlavor flavor)  
zwraca wartość true, jeśli możliwe jest przekazywanie danego formatu danych, wartość false — w przeciwnym razie.
- Object getTransferData(DataFlavor flavor)  
zwraca dane w żądanym formacie. Wyrzuca wyjątek UnsupportedOperationException, jeśli dane nie są dostępne w danym formacie.

### 7.13.3. Interfejs Transferable i formaty danych

Format danych reprezentowany przez obiekt klasy DataFlavor określony jest przez:

- nazwę typu MIME (na przykład "image/gif"),
- klasę reprezentującą dane i umożliwiającą dostęp do nich (na przykład `java.awt.Image`).

Dodatkowo każdy format danych posiada czytelną nazwę (na przykład "GIF Image").

Klasa reprezentująca dane może być podana jako parametr class typu MIME, na przykład

```
image/gif;class=java.awt.Image
```



Powyższy przykład ilustruje jedynie składnię. Nie istnieje standardowy format umożliwiający przekazywanie danych w formacie GIF.

Jeśli nie podamy żadnego parametru class, to domyślną klasą reprezentacji będzie InputStream.

Zdefiniowano trzy typy MIME dla transferu lokalnych, serializowanych i zdalnych obiektów Java.

```
application/x-java-jvm-local-objectref
application/x-java-serialized-object
application/x-java-remote-object
```



Prefiks x- sygnalizuje, że są to tzw. nazwy eksperymentalne, czyli nie są oficjalnie zaakceptowane przez IANA — organizację zajmującą się przydziałem nazw typów MIME.

Standardowy format danych stringFlavor opisany jest przez następujący typ MIME:

```
application/x-java-serialized-object;class=java.lang.String
```

Istnieje możliwość odpytania wszystkich dostępnych formatów danych schowka:

```
DataFlavor[] flavors = clipboard.getAvailableDataFlavors();
```

Możemy również zainstalować obiekt nasłuchujący FlavorListener schowka, który będzie powiadamiany o zmianach formatów danych schowka. Więcej szczegółów na ten temat można odnaleźć w dokumentacji klas i interfejsów przedstawionej poniżej.

#### API java.awt.datatransfer.DataFlavor 1.1

- `DataFlavor(String mimeType, String humanPresentableName)`  
tworzy obiekt reprezentujący format strumienia danych opisany podanym typem MIME.  
*Parametry:*    `mimeType`                      łańcuch znaków opisujący typ MIME,  
                  `humanPresentableName`    czytelna nazwa formatu.
- `DataFlavor(Class class, String humanPresentableName)`  
tworzy obiekt reprezentujący format danych reprezentowany przez klasę Java. Jego typem MIME jest `application/x-java-serialized-object;class=className`.  
*Parametry:*    `class`                                      klasa obiektu zawartego w obiekcie Transferable,  
                  `humanPresentableName`    czytelna nazwa formatu.
- `String getMimeType()`  
zwraca łańcuch znaków reprezentujący typ MIME dla danego formatu.

- `boolean isMimeTypeEqual(String mimeType)`  
sprawdza, czy format danych jest określonego typu MIME.
- `String getHumanPresentableName()`  
zwraca czytelną nazwę formatu danych.
- `Class getRepresentationClass()`  
zwraca obiekt klasy `Class` reprezentujący klasę obiektu, który zwróci obiekt `Transferable`, jeśli zażądamy danego formatu danych. Klasa ta będzie odpowiadać parametrowi `class` typu MIME lub będzie klasą `InputStream`.

**API** `java.awt.datatransfer.Clipboard` 1.1

- `DataFlavor[] getAvailableDataFlavors()` 5.0  
zwraca tablicę dostępnych formatów danych.
- `void addFlavorListener(FlavorListener listener)` 5.0  
instaluje obiekt nasłuchujący schowka powiadamiany o zmianach w zbiorze formatów danych.

**API** `java.awt.datatransfer.Transferable` 1.1

- `DataFlavor[] getTransferDataFlavors()`  
zwraca tablicę dostępnych formatów.

**API** `java.awt.datatransfer.FlavorListener` 5.0

- `void flavorsChanged(FlavorEvent event)`  
metoda wywoływana, gdy zmianie ulega zbiór formatów danych schowka.

## 7.13.4. Przekazywanie obrazów za pomocą schowka

Obiekty przekazywane za pomocą schowka muszą implementować interfejs `Transferable`. Klasa `StringSelection` jest obecnie jedyną publicznie dostępną klasą w standardowej bibliotece Java implementującą ten interfejs. W bieżącym podrozdziale pokażemy, w jaki sposób przekazywać obrazy za pomocą schowka. Ponieważ Java nie dostarcza klasy do tego odpowiedniej, musimy zaimplementować ją sami.

Implementacja takiej klasy jest banalna. Przechowuje ona obiekt klasy `Image` i informuje, że jedynym dostępnym formatem danych jest `DataFlavor.imageFlavor`.

```
class ImageSelection implements Transferable
{
    private Image theImage;

    public ImageSelection(Image image)
    {
```

```

        theImage = image;
    }

    public DataFlavor[] getTransferDataFlavors()
    {
        return new DataFlavor[] { DataFlavor.imageFlavor };
    }

    public boolean isDataFlavorSupported(DataFlavor flavor)
    {
        return flavor.equals(DataFlavor.imageFlavor);
    }

    public Object getTransferData(DataFlavor flavor)
        throws UnsupportedFlavorException
    {
        if(flavor.equals(DataFlavor.imageFlavor))
        {
            return theImage;
        }
        else
        {
            throw new UnsupportedFlavorException(flavor);
        }
    }
}

```

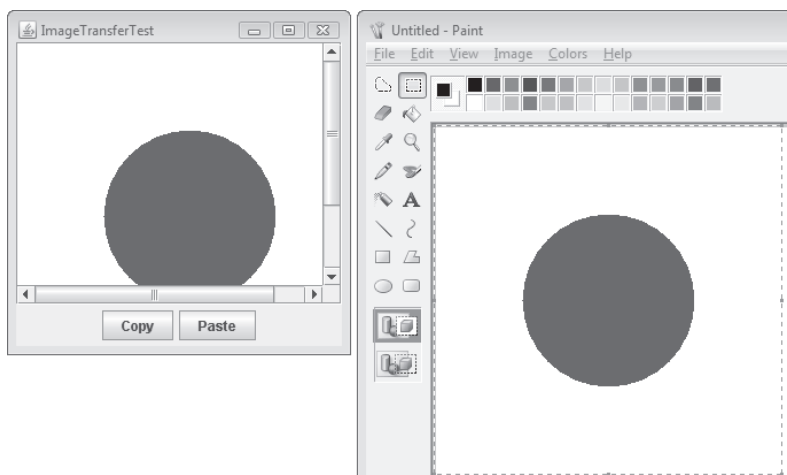


Java SE definiuje stałą `DataFlavor.imageFlavor` i wykonuje konwersję między formatem obrazów w języku Java a formatem schowka. Nie dostarcza jednak klasy obudowującej, która umożliwiłaby umieszczanie obrazów Java w schowku.

Program z listingu 7.18 demonstruje przekazywanie obrazów między aplikacją Java a schowkiem systemowym. Po uruchomieniu tworzy on obraz zawierający czerwone koło. Wybranie przycisku *Copy* powoduje umieszczenie obrazu w schowku, skąd może zostać wklejony do innych aplikacji (patrz rysunek 7.41). Możemy także umieścić w schowku obraz pochodzący z dowolnej innej aplikacji, a następnie wkleić go do okna naszego przykładowego programu, wybierając przycisk *Paste* (patrz rysunek 7.42).

#### Rysunek 7.41.

*Kopiowanie obrazu z aplikacji Java do macierzystej aplikacji systemu*



**Rysunek 7.42.**

Kopiowanie obrazu z macierzystej aplikacji systemu do aplikacji Java

**Listing 7.18.** *imageTransfer/ImageTransferFrame.java*

```
package imageTransfer;

import java.awt.*;
import java.awt.datatransfer.*;
import java.awt.event.*;
import java.awt.image.*;
import java.io.*;
import javax.swing.*;

/**
 * Ramka zawierająca etykietę wyświetlającą obraz
 * i przyciski umożliwiające jego kopiowanie i wklejanie
 * za pośrednictwem schowka systemowego.
 */
class ImageTransferFrame extends JFrame
{
    private JLabel label;
    private Image image;
    private static final int IMAGE_WIDTH = 300;
    private static final int IMAGE_HEIGHT = 300;

    public ImageTransferFrame()
    {
        label = new JLabel();
        image = new BufferedImage(IMAGE_WIDTH, IMAGE_HEIGHT, BufferedImage.TYPE_INT_ARGB);
        Graphics g = image.getGraphics();
        g.setColor(Color.WHITE);
        g.fillRect(0, 0, IMAGE_WIDTH, IMAGE_HEIGHT);
        g.setColor(Color.RED);
        g.fillOval(IMAGE_WIDTH / 4, IMAGE_WIDTH / 4, IMAGE_WIDTH / 2, IMAGE_HEIGHT / 2);

        label.setIcon(new ImageIcon(image));
        add(new JScrollPane(label), BorderLayout.CENTER);
        JPanel panel = new JPanel();

        JButton copyButton = new JButton("Copy");
```

```
panel.add(copyButton);
copyButton.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent event)
    {
        copy();
    }
});

JButton pasteButton = new JButton("Paste");
panel.add(pasteButton);
pasteButton.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent event)
    {
        paste();
    }
});

add(panel, BorderLayout.SOUTH);
pack();
}

/**
 * Kopiuje bieżący obraz do schowka systemowego.
 */
private void copy()
{
    Clipboard clipboard = Toolkit.getDefaultToolkit().getSystemClipboard();
    ImageTransferable selection = new ImageTransferable(image);
    clipboard.setContents(selection, null);
}

/**
 * Wkleja obraz ze schowka systemowego.
 */
private void paste()
{
    Clipboard clipboard = Toolkit.getDefaultToolkit().getSystemClipboard();
    DataFlavor flavor = DataFlavor.imageFlavor;
    if (clipboard.isDataFlavorAvailable(flavor))
    {
        try
        {
            image = (Image) clipboard.getData(flavor);
            label.setIcon(new ImageIcon(image));
        }
        catch (UnsupportedFlavorException exception)
        {
            JOptionPane.showMessageDialog(this, exception);
        }
        catch (IOException exception)
        {
            JOptionPane.showMessageDialog(this, exception);
        }
    }
}
}
```

---

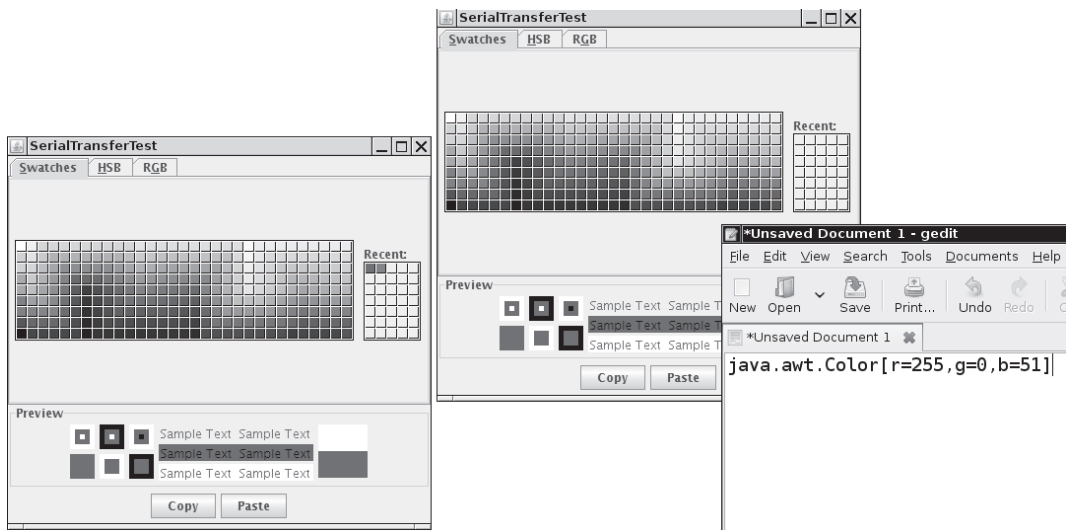
Program przekazywania obrazów powstał jako modyfikacja programu przekazywania tekstów. Formatem danych jest teraz `DataFlavor.imageFlavor`, a klasa `ImageSelection` umożliwia umieszczanie obrazów w schowku systemowym.

## 7.13.5. Wykorzystanie schowka systemowego do przekazywania obiektów Java

Załóżmy, że chcemy kopiować obiekty pomiędzy aplikacjami Java. Możemy to osiągnąć, umieszczając serializowane obiekty Java w schowku systemowym.

Program, którego kod źródłowy zawiera listing 7.19, demonstruje taką możliwość. Program zawiera komponent wyboru kolorów. Wybranie przycisku *Copy* powoduje umieszczenie w schowku systemowym informacji o wybranym kolorze w postaci serializowanego obiektu klasy `Color`. Po wybraniu przycisku *Paste* program sprawdza, czy schowek systemowy zawiera serializowany obiekt klasy `Color`. Jeśli tak, to pobiera go i wykorzystuje jako bieżący wybrany kolor komponentu wyboru kolorów.

Serializowane obiekty możemy przekazywać między aplikacjami Java (rysunek 7.43). W tym celu najlepiej uruchomić dwie kopie przykładowego programu `SerialTransferTest`. W jednej z nich wybierzmy przycisk *Copy*, a następnie w drugiej — przycisk *Paste*. Obiekt klasy `Color` zostanie przekazany w ten sposób z jednej maszyny wirtualnej do drugiej.



Rysunek 7.43. Kopiowanie danych między dwiema instancjami aplikacji Java

Listing 7.19. `serialTransfer/SerialTransferFrame.java`

```
package serialTransfer;

import java.awt.*;
import java.awt.datatransfer.*;
```

```

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

/**
 * Ramka zawierająca komponent wyboru kolorów
 * oraz przyciski operacji kopiowania i wklejania.
 */
class SerialTransferFrame extends JFrame
{
    private JColorChooser chooser;

    public SerialTransferFrame()
    {
        chooser = new JColorChooser();
        add(chooser, BorderLayout.CENTER);
        JPanel panel = new JPanel();

        JButton copyButton = new JButton("Copy");
        panel.add(copyButton);
        copyButton.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                copy();
            }
        });

        JButton pasteButton = new JButton("Paste");
        panel.add(pasteButton);
        pasteButton.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                paste();
            }
        });

        add(panel, BorderLayout.SOUTH);
        pack();
    }

    /**
     * Kopiuje wybrany kolor do schowka systemowego.
     */
    private void copy()
    {
        Clipboard clipboard = Toolkit.getDefaultToolkit().getSystemClipboard();
        Color color = chooser.getColor();
        Serializable selection = new Serializable(color);
        clipboard.setContents(selection, null);
    }

    /**
     * Wkleja kolor ze schowka systemowego do komponentu wyboru koloru.
     */
    private void paste()
    {

```



```

Clipboard clipboard = Toolkit.getDefaultToolkit().getSystemClipboard();
try
{
    DataFlavor flavor = new DataFlavor(
        "application/x-java-serialized-object;class=java.awt.Color");
    if (clipboard.isDataFlavorAvailable(flavor))
    {
        Color color = (Color) clipboard.getData(flavor);
        chooser.setColor(color);
    }
}
catch (ClassNotFoundException e)
{
    JOptionPane.showMessageDialog(this, e);
}
catch (UnsupportedFlavorException e)
{
    JOptionPane.showMessageDialog(this, e);
}
catch (IOException e)
{
    JOptionPane.showMessageDialog(this, e);
}
}
}

/**
 * Klasa obudowująca serializowane obiekty
 * przekazywane za pomocą schowka systemowego.
 */
class SerialTransferable implements Transferable
{
    private Serializable obj;

    /**
     * Tworzy obiekt klasy SerialSelection.
     * @param o dowolny serializowany obiekt
     */
    SerialTransferable(Serializable o)
    {
        obj = o;
    }

    public DataFlavor[] getTransferDataFlavors()
    {
        DataFlavor[] flavors = new DataFlavor[2];
        Class<?> type = obj.getClass();
        String mimeType = "application/x-java-serialized-object;class=" + type.getName();
        try
        {
            flavors[0] = new DataFlavor(mimeType);
            flavors[1] = DataFlavor.stringFlavor;
            return flavors;
        }
        catch (ClassNotFoundException e)
        {
            return new DataFlavor[0];
        }
    }
}

```

```

    }

    public boolean isDataFlavorSupported(DataFlavor flavor)
    {
        return DataFlavor.stringFlavor.equals(flavor)
            || "application".equals(flavor.getPrimaryType())
            && "x-java-serialized-object".equals(flavor.getSubType())
            && flavor.getRepresentationClass().isAssignableFrom(obj.getClass());
    }

    public Object getTransferData(DataFlavor flavor) throws UnsupportedFlavorException
    {
        if (!isDataFlavorSupported(flavor)) throw new UnsupportedFlavorException(flavor);

        if (DataFlavor.stringFlavor.equals(flavor)) return obj.toString();

        return obj;
    }
}

```

Aby możliwe było przemieszczanie obiektów w ten sposób, Java umieszcza w schowku systemowym binarną reprezentację serializowanego obiektu. Inna aplikacja Java, niekoniecznie tego samego typu co aplikacja, która umieściła obiekt w schowku, może pobrać dane ze schowka i odtworzyć obiekt.

Oczywiście aplikacja, która nie została napisana w języku Java, nie będzie wiedzieć, jak zinterpretować dane schowka. Z tego powodu przykładowy program umożliwi pobranie danych ze schowka także w formacie tekstowym. Tekst ten uzyskiwany jest jako wynik zastosowania metody `toString` do obiektu przekazywanego za pomocą schowka. Aby to sprawdzić, wystarczy po umieszczeniu koloru w schowku uruchomić na przykład program edytora tekstu i wkleić do niego informację ze schowka. Łańcuch znaków w postaci

```
java.awt.Color[r=255,g=51,b=51]
```

zostanie wstawiony w dokumencie edytora tekstów.

Przekazywanie serializowanych obiektów za pośrednictwem schowka nie wymaga zbyt wiele programowania. Korzystamy z typu MIME:

```
application/x-java-serialized-object;class=nazwaKlasy
```

Podobnie jak w poprzednich przykładach stworzymy własną klasę obudowującą — szczegóły znajdziesz w kodzie programu.

## 7.13.6. Zastosowanie lokalnego schowka do przekazywania referencji obiektów

Czasami pojawia się potrzeba skopiowania i wklejenia danych, których typ nie jest obsługiwany przez schowek systemowy oraz nie jest serializowalny. Aby przekazać za pośrednictwem schowka dowolną referencję obiektu Java w obrębie tej samej maszyny wirtualnej, używamy następującego typu MIME:

```
application/x-java-jvm-local-objectref;class=nazwaKlasy
```

Dla tego typu musimy sami zdefiniować obiekt obudowujący `Transferable`. Proces ten jest analogiczny jak w przypadku obiektu obudowującego `SerialTransferable` z poprzedniego przykładu.

Ponieważ referencja obiektu ma sens jedynie w obrębie danej maszyny wirtualnej, to do jej przekazywania nie możemy użyć schowka systemowego. Zamiast nim posłużymy się schowkiem lokalnym:

```
Clipboard clipboard = new Clipboard("local");
```

Parametrem konstruktora schowka jest jego nazwa.

Korzystanie z lokalnego schowka ma jedną istotną wadę. Musimy synchronizować schowek lokalny z systemowym, aby użytkownicy ich nie pomylili. Obecnie synchronizacja taka nie jest automatycznie wykonywana na platformie Java.

**API** `java.awt.datatransfer.Clipboard` 1.1

■ `Clipboard(String name)`

tworzy lokalny schowek o podanej nazwie.

## 7.14. Mechanizm „przeciągnij i upuść”

Schowek odgrywa rolę pośrednika, gdy korzystamy z mechanizmu „skopiuj i wklej” do przekazywania danych między programami. Mechanizm „przeciągnij i upuść” pozwala pominąć programom wszelkie pośrednictwo i komunikować się bezpośrednio. Platforma Java 2 oferuje podstawową obsługę tego mechanizmu. Obiekty możemy przeciągać między aplikacjami Java oraz macierzystymi aplikacjami danej platformy. W podrozdziale tym pokażemy, w jaki sposób zaimplementować aplikację w języku Java, która umożliwi upuszczanie obiektów oraz aplikację, która jest źródłem przeciąganych obiektów.

Zanim wnikniemy głębiej w implementację mechanizmu „przeciągnij i upuść” w języku Java, przyjrzymy się najpierw sposobowi jego działania. Zilustrujemy go na przykładzie programów Windows Explorer oraz WordPad — dla innych platform należy eksperymentować z dostępnymi aplikacjami korzystającymi z mechanizmu „przeciągnij i upuść”.

*Operację przeciągnięcia* inicjujemy odpowiednim *gestem* wewnątrz okna *źródła* — zwykle wybierając najpierw jeden lub więcej elementów, a następnie przeciągając je z wyjściowej lokalizacji. Jeśli zwołimy klawisz myszy nad celem, to zapyta on źródło o informacje dotyczące upuszczanych elementów i zainicjuje pewne operacje. Na przykład jeśli przeciągniemy ikonę pliku z menedżera plików i upuścimy ją na ikonę reprezentującą katalog, to plik zostanie przeniesiony do tego katalogu. Jeśli natomiast upuścimy ją w oknie edytora tekstu, to edytor otworzy ten plik. (Oczywiście wymaga to użycia menedżera plików i edytora tekstu, które obsługują mechanizm „przeciągnij i upuść”, na przykład pary Explorer/WordPad w systemie Windows lub Nautilus/gedit w środowisku GNOME).

Jeśli podczas przeciągania przytrzymamy klawisz *Ctrl*, to typ operacji związanej z upuszczeniem zmieni się z *przesunięcia* na *kopiowanie* i kopia pliku zostanie umieszczona w danym katalogu. Jeśli przytrzymamy *oba* klawisze *Shift* i *Ctrl*, to w katalogu umieszczone zostanie *łącze* do pliku (inne platformy mogą wykorzystywać w tym celu inne kombinacje klawiszy).









Istnieją więc trzy rodzaje operacji dostępne za pomocą różnych gestów:

- przesunięcie,
- kopiowanie,
- utworzenie łącza.

Intencją operacji utworzenia łącza jest utworzenie referencji upuszczanego elementu. Operacja taka wymaga zwykle odpowiedniej obsługi ze strony systemu operacyjnego (na przykład tworzenia łączy dla plików lub łączy obiektów w dokumentach) i dlatego nie ma sensu tworzenie przenośnych aplikacji korzystających z tej operacji. Omawiając mechanizm „przeciągnij i upuść”, skoncentrujemy się zatem na operacjach przesunięcia i kopiowania.

Operacja przeciągnięcia zwykle posiada pewien rodzaj graficznej ilustracji. W najprostszym przypadku polega on na zmianie kształtu kursora. Kursor myszy zmienia swój kształt, gdy przemieszczany jest nad celami, w zależności od tego, czy *upuszczenie obiektu* jest możliwe, czy też nie. Jeśli upuszczenie jest możliwe, to kształt kursora ilustruje typ wykonywanej operacji. Tabela 7.6 pokazuje możliwe kształty, które przyjmuje kursor nad celami.

**Tabela 7.6.** *Kształty kursora podczas operacji upuszczania*

Akcja	Ikona Windows	Ikona Gnome
Przesunięcie		
Kopiowanie		
Utworzenie łącza		
Upuszczenie zabronione		

Oprócz ikon plików możemy także przeciągać inne obiekty. W programie WordPad możemy na przykład wybrać fragment tekstu i przeciągnąć go do pewnego celu, aby sprawdzić, jak on się zachowa.



Powyższy eksperyment pokazuje pewną wadę mechanizmu „przeciągnij i upuść”. Użytkownikowi trudno przewidzieć, jakie elementy interfejsu może przeciągać, gdzie je upuszczać i co to spowoduje. Ponieważ operacja przesunięcia powoduje usunięcie oryginału, to wielu użytkowników wyraża uzasadnioną obawę przed eksperymentowaniem z mechanizmem „przeciągnij i upuść”.

## 7.14.1. Przekazywanie danych pomiędzy komponentami Swing

Począwszy od wersji Java SE 1.4, niektóre komponenty biblioteki Swing obsługują mechanizm „przeciągnij i upuść” (patrz tabela 7.7). Wybrany tekst możemy przeciągać z szeregu komponentów i upuszczać na komponenty tekstowe. Ze względu na konieczność zachowania zgodności z wcześniejszymi wersjami przeciąganie należy uaktywnić, wywołując metodę `setDragEnabled`. Natomiast obsługa upuszczania jest zawsze włączona.

**Tabela 7.7.** Komponenty Swing umożliwiające przekazywanie danych

Komponent	Jako źródło	Jako cel
JFileChooser	Przekazuje listę plików.	Niedostępny.
JColorChooser	Przekazuje lokalną referencję do obiektu klasy <code>Color</code> .	Akceptuje dowolny obiekt klasy <code>Color</code> .
JTextField JFormattedTextField	Przekazuje wybrany tekst.	Akceptuje tekst.
JPasswordField	Niedostępny (ze względów bezpieczeństwa).	Akceptuje tekst.
JTextArea JTextPane JEditorPane	Przekazuje wybrany tekst.	Akceptuje tekst i listy plików.
JList JLabel Jtree	Przekazuje tekstowy opis wyboru (tylko kopiowanie).	Niedostępny.



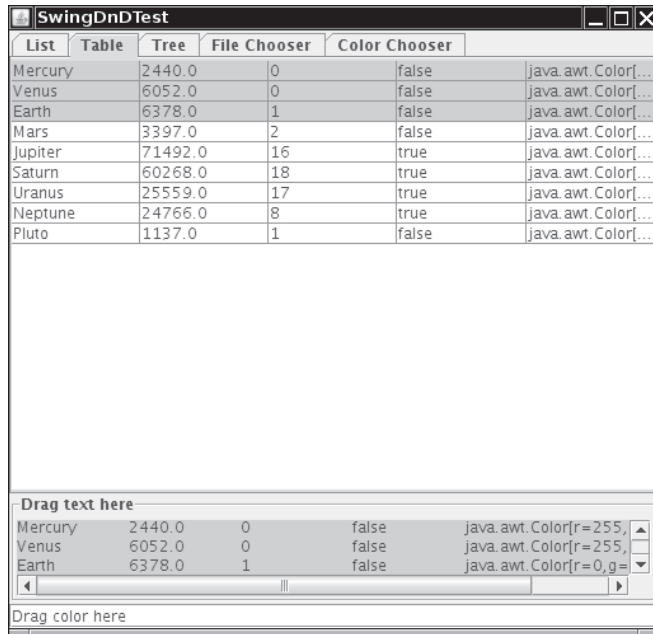
Pakiet `java.awt.dnd` dostarcza niskopoziomowy interfejs mechanizmu „przeciągnij i upuść”, który stanowi bazę dla obsługi tego mechanizmu przez komponenty Swing. Interfejsu tego nie będziemy tutaj omawiać.

Program przedstawiony na listingu 7.20 demonstruje zachowanie komponentów Swing. Zauważmy, że:

- Możemy wybierać wiele elementów list, tabel i drzew (patrz listing 7.21) i przeciągać je.
- Przeciąganie elementów tabeli jest nieco kłopotliwe. Najpierw musimy wybrać dany element za pomocą myszy, po czym należy zwolnić jej klawisz. Następnie jeszcze raz kliknąć i dopiero wtedy możemy przeciągnąć element.
- Upuszczając elementy w obszarze tekstowym, możemy zaobserwować sposób formatowania przeciąganej informacji. Komórki tabeli są oddzielone znakami tabulacji, a każdy wiersz tabeli zostaje umieszczony w osobnym wierszu tekstu (patrz rysunek 7.44).
- Elementy list, tabel, drzew oraz komponentów wyboru plików lub kolorów możemy jedynie kopiować, a nie przesuwać. Usuwanie elementów list, tabel i drzew nie jest możliwe dla wszystkich modeli danych. W następnym podrozdziale pokażemy, w jaki sposób zaimplementować tę możliwość, gdy model danych umożliwia edycję.

**Rysunek 7.44.**

Program  
SwingDnDTest  
w działaniu

**Listing 7.20.** *dnd/SwingDnDTest.java*

```

package dnd;

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

/**
 * Program demonstrujący podstawową obsługę mechanizmu "przeciągnij i upuść"
 * przez komponenty Swing.
 * @version 1.10 2007-09-20
 * @author Cay Horstmann
 */
public class SwingDnDTest
{
    public static void main(String[] args)
    {
        EventQueue.invokeLater(new Runnable()
        {
            public void run()
            {
                JFrame frame = new SwingDnDFrame();
                frame.setTitle("SwingDnDTest");
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.setVisible(true);
            }
        });
    }
}

```

Listing 7.21. *dnd/SampleComponents.java*

```

package dnd;

import java.awt.*;

import javax.swing.*;
import javax.swing.tree.*;

public class SampleComponents
{
    public static JTree tree()
    {
        DefaultMutableTreeNode root = new DefaultMutableTreeNode("World");
        DefaultMutableTreeNode country = new DefaultMutableTreeNode("USA");
        root.add(country);
        DefaultMutableTreeNode state = new DefaultMutableTreeNode("California");
        country.add(state);
        DefaultMutableTreeNode city = new DefaultMutableTreeNode("San Jose");
        state.add(city);
        city = new DefaultMutableTreeNode("Cupertino");
        state.add(city);
        state = new DefaultMutableTreeNode("Michigan");
        country.add(state);
        city = new DefaultMutableTreeNode("Ann Arbor");
        state.add(city);
        country = new DefaultMutableTreeNode("Germany");
        root.add(country);
        state = new DefaultMutableTreeNode("Schleswig-Holstein");
        country.add(state);
        city = new DefaultMutableTreeNode("Kiel");
        state.add(city);
        return new JTree(root);
    }

    public static JList<String> list()
    {
        String[] words = { "quick", "brown", "hungry", "wild", "silent", "huge", "private",
            "abstract", "static", "final" };

        DefaultListModel<String> model = new DefaultListModel<>();
        for (String word : words)
            model.addElement(word);
        return new JList<>(model);
    }

    public static JTable table()
    {
        Object[][] cells = { { "Mercury", 2440.0, 0, false, Color.YELLOW },
            { "Venus", 6052.0, 0, false, Color.YELLOW },
            { "Earth", 6378.0, 1, false, Color.BLUE }, { "Mars", 3397.0, 2, false,
            ↪Color.RED },
            { "Jupiter", 71492.0, 16, true, Color.ORANGE },
            { "Saturn", 60268.0, 18, true, Color.ORANGE },
            { "Uranus", 25559.0, 17, true, Color.BLUE },
            { "Neptune", 24766.0, 8, true, Color.BLUE },
            { "Pluto", 1137.0, 1, false, Color.BLACK } };
    }
}

```

```

        String[] columnNames = { "Planet", "Radius", "Moons", "Gaseous", "Color" };
        return new JTable(cells, columnNames);
    }
}

```

- Nie możemy przeciągać elementów pomiędzy listami, tabelami, drzewami i komponentami wyboru plików.
- Jeśli uruchomimy dwie kopie programu, możemy przeciągnąć kolor pomiędzy komponentami wyboru kolorów.
- Nie możemy przeciągać tekstu z obszaru tekstowego, ponieważ dla tego komponentu nie wywołaliśmy metody `setDragEnabled`.

Pakiet Swing dostarcza potencjalnie użytecznego mechanizmu umożliwiającego przekształcenie komponentu w źródło przeciągania lub cel upuszczania. Wystarczy zainstalować *obiekt obsługi transferu* dla danej właściwości. Nasz przykładowy program używa w tym celu poniższego wywołania:

```
textField.setTransferHandle(new TransferHandler("background"));
```

Dzięki niemu przeciągnięcie koloru do pola tekstowego powoduje zmianę tła pola tekstowego.

Po upuszczeniu obiekt obsługi transferu sprawdza, czy jeden z formatów danych jest reprezentowany przez klasę `Color`. Jeśli tak, wywołuje metodę `setBackground`.

Instalując obiekt obsługi transferu dla pola tekstowego, powodujemy wyłączenie standardowego obiektu obsługi transferu. Oznacza to, że nie możemy już ani przeciągać tekstu z tego pola, ani upuszczać tekstu na to pole, a nawet wycinać i wklejać tekstu. Natomiast możemy wybrać tekst i przeciągnąć go do komponentu wyboru koloru, który pokaże wtedy kolor tła pola tekstowego.

#### API javax.swing.JComponent 1.2

- `void setTransferHandler(TransferHandler handler) 1.4`  
określa procedurę obsługi komponentu związaną z przekazywaniem danych (wycinaniem, kopiowaniem, wklejaniem, przeciąganiem i upuszczaniem).

#### API javax.swing.TransferHandler 1.4

- `TransferHandler(String propertyName)`  
tworzy procedurę obsługi, która odczytuje lub nadaje wartość właściwości `JavaBeans` o podanej nazwie, gdy wykonywana jest operacja przekazywania danych.

```

API javax.swing.JFileChooser 1.2
javax.swing.JColorChooser 1.2
javax.swing.JTextComponent 1.2
javax.swing.JList 1.2
javax.swing.JTable 1.2
javax.swing.JTree 1.2

```



- `void setDragEnabled(boolean b)` 1.4

aktywuje lub dezaktywuje przeciąganie danych z komponentu.

## 7.14.2. Źródła przeciąganych danych

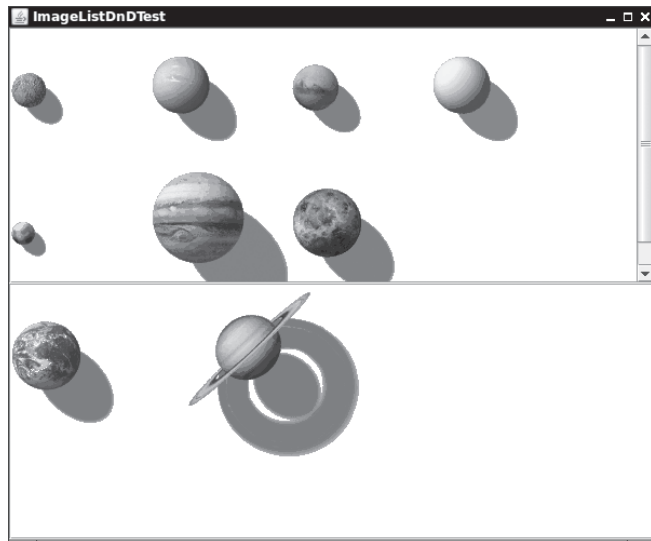
W poprzednim podrozdziale pokazaliśmy zalety podstawowej obsługi mechanizmu „przeciągnij i upuść” przez komponenty Swing. Teraz zajmiemy się konfigurowaniem komponentów jako źródeł przeciąganych danych. W następnym podrozdziale omówimy cele, na które można upuszczać przeciągane dane, i przedstawimy przykładowy komponent, który będzie zarówno źródłem jak i celem mechanizmu „przeciągnij i upuść” dla danych w postaci obrazów.

Aby zindywidualizować działanie mechanizmu „przeciągnij i upuść” dla komponentu Swing, musimy stworzyć klasę pochodną klasy `TransferHandler`. Najpierw zastępujemy metodę `getSourceActions`, która informuje o akcjach (`COPY`, `MOVE`, `LINK`) obsługiwanych przez nasz komponent. Następnie zastępujemy metodę `createTransferable`, która tworzy obiekt `Transferable`, tak samo jak w przypadku implementowanej wcześniej operacji kopiowania do schowka.

W naszym przykładowym programie umożliwimy przeciąganie obrazów z listy `JList` wypełnionej ich ikonami (patrz rysunek 7.45). Poniżej przedstawiamy implementację metody `createTransferable`. Wybrany obraz zostaje po prostu obudowany obiektem `ImageTransferable`.

```
protected Transferable createTransferable(JComponent source)
{
    JList list = (JList) source;
    int index = list.getSelectedIndex();
    if (index < 0) return null;
    ImageIcon icon = (ImageIcon) list.getModel().getElementAt(index);
    return new ImageTransferable(icon.getImage());
}
```

**Rysunek 7.45.**  
Aplikacja *ImageList*  
w działaniu



W naszym przykładzie sprzyja nam fakt, że komponent `JList` obsługuje operację przeciągania. Wystarczy uaktywnić ją za pomocą metody `setDragEnabled`. Jeśli chcemy dodać operację przeciągania dla komponentu, który standardowo jej nie obsługuje, musimy sami zainicjować transfer. Poniżej przedstawiamy przykład takiej inicjacji dla komponentu `JLabel`:

```
label.addMouseListener(new MouseAdapter()
{
    public void mousePressed(MouseEvent evt)
    {
        int mode;
        if ((evt.getModifiers() & (InputEvent.CTRL_MASK | InputEvent.SHIFT_MASK))
            != 0)
            mode = TransferHandler.COPY;
        else mode = TransferHandler.MOVE;
        JComponent comp = (JComponent) evt.getSource();
        TransferHandler th = comp.getTransferHandler();
        th.exportAsDrag(comp, evt, mode);
    }
});
```

W tym przypadku rozpoczynamy transfer, gdy użytkownik kliknie etykietę. Bardziej zaawansowana implementacja wykrywałaby raczej przeciągnięcie kursora myszy.

Gdy użytkownik zakończy operację upuszczania, wywołana zostaje metoda `exportDone` obiektu obsługującego transfer źródła. Metoda ta powinna usunąć przeciągnięty obiekt w przypadku, gdy użytkownik dokonał jego przesunięcia. Oto jej implementacja dla listy obrazków:

```
protected void exportDone(JComponent source, Transferable data, int action)
{
    if (action == MOVE)
    {
        JList list = (JList) source;
        int index = list.getSelectedIndex();
        if (index < 0) return;
        DefaultListModel model = (DefaultListModel) list.getModel();
        model.remove(index);
    }
}
```

Podsumowując, aby przekształcić komponent w źródło przeciąganych danych, dodajemy do niego obiekt obsługi transferu, który specyfikuje:

- Akcje obsługiwane przez komponent.
- Dane przekazywane przez komponent.
- Sposób usunięcia oryginalnych danych po wykonaniu operacji przesunięcia.

Dodatkowo, jeśli nasz komponent nie został wymieniony w tabeli 7.7, musimy sami wykrywać rozpoczęcie operacji przeciągania i inicjować transfer.

#### API javax.swing.TransferHandler 1.4

- `int getSourceActions(JComponent c)`  
metodę tę zastępujemy, aby poinformować o akcjach dopuszczalnych dla danego komponentu, gdy jest on źródłem przeciąganych danych (suma logiczna na bitach stałych `COPY`, `MOVE` i `LINK`).

- `protected Transferable createTransferable(JComponent source)`  
metodę tę zastępujemy, aby stworzyć obiekt dla przeciąganych danych.
- `void exportAsDrag(JComponent comp, InputEvent e, int action)`  
rozpoczyna akcję przeciągnięcia dla danego komponentu. Parametr `action` przyjmuje jedną z wartości `COPY`, `MOVE` lub `LINK`.
- `protected void exportDone(JComponent source, Transferable data, int action)`  
metodę tę zastępujemy, aby skorygować dane źródła po pomyślnym transferze danych.

### 7.14.3. Cele upuszczanych danych

W tym podrozdziale przedstawimy sposób implementacji celu upuszczanych danych. Naszym przykładem będzie ponownie komponent `JList` zawierający ikony obrazków. Wzbogacimy go o obsługę operacji upuszczania, co pozwoli użytkownikom upuszczać obrazki na listę.

Aby przekształcić komponent w cel upuszczanych danych, musimy skonfigurować dla niego obiekt klasy `TransferHandler` i zaimplementować metody `canImport` i `importData`.



Obiekt obsługi transferu możemy dodać do komponentu `JFrame`. Możliwość ta jest najczęściej używana do obsługi upuszczania plików w oknie aplikacji. Dozwolone miejsca upuszczania zawierają dekoracje ramki oraz pasek menu, ale nie komponenty umieszczone w ramce (które mają własne obiekty obsługi transferu).

Metoda `canImport` jest ciągle wywoływana, gdy użytkownik przemieszcza kursor myszy nad komponentem będącym celem upuszczanych danych. Jeśli upuszczenie jest dozwolone, zwraca wartość `true`. Pozwala to zmienić ikonę kursora myszy w taki sposób, aby sygnalizowała użytkownikowi możliwość upuszczenia.

Metoda `canImport` ma parametr typu `TransferHandler.TransferSupport`. Za jego pośrednictwem otrzymujemy informacje o akcji upuszczania wybranej przez użytkownika, miejscu upuszczenia i przekazywanych danych. (W wersjach wcześniejszych od Java SE 6 wywoływana była w tym celu inna metoda `canImport`, która dostarczała jedynie listę formatów danych).

Implementując metodę `canImport`, możemy również zmienić akcję upuszczania wybraną przez użytkownika. Na przykład jeśli użytkownik wybrał akcję przesunięcia, ale oryginał z pewnych względów nie powinien zostać usunięty, możemy wymusić na obiekcie obsługi transferu wykonanie akcji kopiowania.

Oto typowy przykład. Komponent w postaci listy obrazków będzie akceptować upuszczenia list plików oraz obrazków. W przypadku, gdy upuszczona zostanie lista plików, akcja `MOVE` wybrana przez użytkownika zostanie zastąpiona akcją `COPY`, aby zapobiec usunięciu plików obrazków.

```
public boolean canImport(TransferSupport support)
{
    if (support.isDataFlavorSupported(DataFlavor.javaFileListFlavor))
```

```

    {
        if (support.getUserDropAction() == MOVE) support.setDropAction(COPY);
        return true;
    }
    else return support.isDataFlavorSupported(DataFlavor.imageFlavor);
}

```

Bardziej zaawansowana implementacja metody `canImport` sprawdzałaby, czy pliki rzeczywiście zawierają obrazki.

Gdy użytkownik przesuwając kursor nad celem upuszczenia, komponenty `JList`, `JTable`, `JTree` i `JTextComponent` dostarczają wizualnej informacji o pozycji, na którą zostaną wstawione upuszczone dane. Domyślnie odbywa się to poprzez wybór pozycji komponentu (`JList`, `JTable`, `JTree`) lub odpowiednie umiejscowienie kursora (`JTextComponent`). Rozwiązanie takie nie jest szczególnie udane i pozostawiono je jedynie w celu zachowania zgodności z poprzednimi wersjami. Doskonalszej wizualizacji miejsca upuszczenia możemy dostarczyć, wywołując metodę `setDropMode`.

Możemy wybrać, czy upuszczane dane zastąpią istniejącą pozycję komponentu, czy zostaną umieszczone pomiędzy istniejącymi pozycjami. W naszym przykładowym programie wywołujemy

```
setDropMode(DropMode.ON_OR_INSERT);
```

aby umożliwić użytkownikowi zastąpienie istniejącej pozycji poprzez upuszczenie na nią nowych danych lub wstawienie danych pomiędzy istniejącymi pozycjami listy (patrz rysunek 7.46). Tryby upuszczania obsługiwane przez komponenty Swing zostały przedstawione w tabeli 7.8.

#### Rysunek 7.46.

Wizualizacja upuszczenia na istniejący element lub pomiędzy elementy

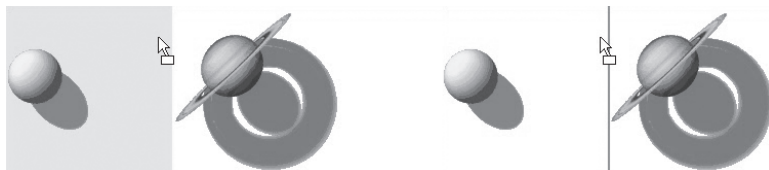


Tabela 7.8. Tryby upuszczania

Komponent	Obsługiwane tryby upuszczania
<code>JList</code> , <code>JTree</code>	<code>ON</code> , <code>INSERT</code> , <code>ON_OR_INSERT</code> , <code>USE_SELECTION</code>
<code>Jtable</code>	<code>ON</code> , <code>INSERT</code> , <code>ON_OR_INSERT</code> , <code>INSERT_ROWS</code> , <code>INSERT_COLS</code> , <code>ON_OR_INSERT_ROWS</code> , <code>ON_OR_INSERT_COLS</code> , <code>USE_SELECTION</code>
<code>JtextComponent</code>	<code>INSERT</code> , <code>USE_SELECTION</code> (w rzeczywistości przesuwając kursor, a nie wybór)

Gdy użytkownik upuści wreszcie dane, wywołana zostaje metoda `importData`. Musimy wtedy pobrać dane ze źródła przeciągnięcia. Wywołując metodę `getTransferable` dla parametru `TransferSupport`, uzyskujemy referencję do obiektu klasy `Transferable`. Jest to ten sam interfejs, który znajduje zastosowanie w przypadku mechanizmu „wytnij i wklej”.

Jednym z typów danych często używanych przez mechanizm „przeciągnij i upuść” jest `DataFlavor.javaFileListFlavor`. Reprezentuje on zbiór plików upuszczanych na komponent celu.

Przekazywane dane są w tym przypadku typu `List<File>`. A oto kod umożliwiający pobranie plików:

```
DataFlavor[] flavors = transferable.getTransferDataFlavors();
if (Arrays.asList(flavors).contains(DataFlavor.javaFileListFlavor))
{
    List<File> fileList = (List<File>)
        ↪transferable.getTransferData(DataFlavor.javaFileListFlavor);
    for (File f : fileList)
    {
        operacje na f;
    }
}
```

Jeśli dane zostały upuszczone na jeden z komponentów wymienionych w tabeli 7.8, musimy poznać dokładne miejsce upuszczenia. W tym celu wywołujemy metodę `getDropLocation` dla parametru `TransferSupport`. Metoda ta zwraca obiekt klasy pochodnej klasy `TransferHandler.DropLocation`. Klasy `JTable`, `JTree` i `JTextComponent` definiują własne klasy pochodne klasy `TransferHandler.DropLocation` określające miejsce upuszczenia w określonym modelu danych. Na przykład miejsce upuszczenia na liście wystarczy określić za pomocą wartości indeksu, ale w przypadku drzewa wymaga podania ścieżki w drzewie. Poniżej przedstawiamy sposób określenia miejsca upuszczenia zastosowany dla naszej listy obrazków:

```
int index;
if (support.isDrop())
{
    JList.DropLocation location = (JList.DropLocation) support.getDropLocation();
    index = location.getIndex();
}
else index = model.size();
```

Klasa pochodna `JList.DropLocation` udostępnia metodę `getIndex` zwracającą indeks upuszczenia. (Klasa `JTree.DropLocation` dysponuje zamiast niej metodą `getPath`).

Metoda `importData` jest również wywoływana na skutek wklejenia danych do komponentu za pomocą kombinacji klawiszy `Ctrl+V`. W tym przypadku metoda `getDropLocation` wyrzuci wyjątek `IllegalStateException`. Dlatego jeśli metoda `isDrop` zwraca wartość `false`, umieszczamy wklejone dane na końcu listy.

W przypadku wstawiania danych na listę, do tabeli lub drzewa musimy sprawdzić także, czy dane mają być umieszczone pomiędzy istniejącymi elementami, czy zastąpić element znajdujący się w miejscu upuszczenia. W przypadku listy wywołujemy w tym celu metodę `isInsert` klasy `JList.DropLocation`. W przypadku innych komponentów sprawdź opis odpowiednich klas umieszczony na końcu tego podrozdziału.

Podsumowując, aby przekształcić komponent w cel upuszczenia, dodajemy do niego obiekt obsługi transferu specyfikujący:

- Sytuacje, gdy upuszczany element może zostać zaakceptowany.
- Sposób importu upuszczonych danych.

Dodatkowo, jeśli dodajemy obsługę upuszczania do komponentów `JList`, `JTable`, `JTree` lub `JTextComponent`, należy określić również tryb upuszczania.

Kompletny kod przykładowego programu został przedstawiony na listingu 7.22. Zwróćmy uwagę, że klasa `ImageList` jest zarówno źródłem przeciągania, jak i celem upuszczania. Uruchom program i spróbuj przeciągnąć obrazki pomiędzy dwiema listami. Możesz również przeciągnąć listę plików obrazków z komponentu wyboru plików innego programu na listę obrazków w naszym programie.

**Listing 7.22.** *dndImage/ImageListDnDFrame.java*

---

```
package dndImage;

import java.awt.*;
import java.awt.datatransfer.*;
import java.io.*;
import java.nio.file.*;
import java.util.*;
import java.util.List;
import javax.imageio.*;
import javax.swing.*;

public class ImageListDnDFrame extends JFrame
{
    private static final int DEFAULT_WIDTH = 600;
    private static final int DEFAULT_HEIGHT = 500;

    private ImageList list1;
    private ImageList list2;

    public ImageListDnDFrame()
    {
        setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);

        list1 = new ImageList(Paths.get(getClass().getPackage().getName(), "images1"));
        list2 = new ImageList(Paths.get(getClass().getPackage().getName(), "images2"));

        setLayout(new GridLayout(2, 1));
        add(new JScrollPane(list1));
        add(new JScrollPane(list2));
    }
}

class ImageList extends JList<ImageIcon>
{
    public ImageList(Path dir)
    {
        DefaultListModel<ImageIcon> model = new DefaultListModel<>();
        try (DirectoryStream<Path> entries = Files.newDirectoryStream(dir))
        {
            for (Path entry : entries)
                model.addElement(new ImageIcon(entry.toString()));
        }
        catch (IOException ex)
        {
            ex.printStackTrace();
        }

        setModel(model);
        setVisibleRowCount(0);
    }
}
```

```

        setLayoutOrientation(JList.HORIZONTAL_WRAP);
        setDragEnabled(true);
        setDropMode(DropMode.ON_OR_INSERT);
        setTransferHandler(new ImageListTransferHandler());
    }
}

class ImageListTransferHandler extends TransferHandler
{
    // Obsługa przeciągania

    public int getSourceActions(JComponent source)
    {
        return COPY_OR_MOVE;
    }

    protected Transferable createTransferable(JComponent source)
    {
        ImageList list = (ImageList) source;
        int index = list.getSelectedIndex();
        if (index < 0) return null;
        ImageIcon icon = list.getModel().getElementAt(index);
        return new ImageTransferable(icon.getImage());
    }

    protected void exportDone(JComponent source, Transferable data, int action)
    {
        if (action == MOVE)
        {
            ImageList list = (ImageList) source;
            int index = list.getSelectedIndex();
            if (index < 0) return;
            DefaultListModel<?> model = (DefaultListModel<?>) list.getModel();
            model.remove(index);
        }
    }

    // Obsługa upuszczania

    public boolean canImport(TransferSupport support)
    {
        if (support.isDataFlavorSupported(DataFlavor.javaFileListFlavor))
        {
            if (support.getUserDropAction() == MOVE) support.setDropAction(COPY);
            return true;
        }
        else return support.isDataFlavorSupported(DataFlavor.imageFlavor);
    }

    public boolean importData(TransferSupport support)
    {
        ImageList list = (ImageList) support.getComponent();
        DefaultListModel<ImageIcon> model = (DefaultListModel<ImageIcon>) list.getModel();

        Transferable transferable = support.getTransferable();
        List<DataFlavor> flavors = Arrays.asList(transferable.getTransferDataFlavors());

        List<Image> images = new ArrayList<>();
    }
}

```

```

try
{
    if (flavors.contains(DataFlavor.javaFileListFlavor))
    {
        @SuppressWarnings("unchecked") List<File> fileList
        = (List<File>) transferable.getTransferData(DataFlavor.javaFileListFlavor);
        for (File f : fileList)
        {
            try
            {
                images.add(ImageIO.read(f));
            }
            catch (IOException ex)
            {
                //obrazek nie został odczytany — pomija
            }
        }
    }
    else if (flavors.contains(DataFlavor.imageFlavor))
    {
        images.add((Image) transferable.getTransferData(DataFlavor.imageFlavor));
    }

    int index;
    if (support.isDrop())
    {
        JList.DropLocation location = (JList.DropLocation) support.getDropLocation();
        index = location.getIndex();
        if (!location.isInsert()) model.remove(index); //miejsce zastapienia
    }
    else index = model.size();
    for (Image image : images)
    {
        model.add(index, new ImageIcon(image));
        index++;
    }
    return true;
}
catch (IOException ex)
{
    return false;
}
catch (UnsupportedFlavorException ex)
{
    return false;
}
}
}

```

---

**API** javax.swing.TransferHandler 1.4

**■ boolean canImport(TransferSupport support) 6**

metodę tę zastępujemy, aby poinformować, czy komponent będący celem upuszczania może zaakceptować dane opisane przez parametr TransferSupport.



- `boolean importData(TransferSupport support)` 6

metodę tę zastępujemy, aby przeprowadzić operację upuszczenia lub wklejenia danych opisaną przez parametr `TransferSupport`. Metoda zwraca wartość `true`, jeśli import danych zakończył się pomyślnie.

#### API javax.swing.JFrame 1.2

- `void setTransferHandler(TransferHandler handler)` 6

określa obiekt obsługi transferu tylko dla operacji upuszczenia i wklejenia.

#### API javax.swing.JList 1.2 javax.swing.JTable 1.2 javax.swing.JTree 1.2 javax.swing.JTextComponent 1.2

- `void setDropMode(DropMode mode)` 6

określa tryb upuszczania dla danego komponentu za pomocą jednej z wartości podanych w tabeli 7.8.

#### API javax.swing.TransferHandler.TransferSupport 6

- `Component getComponent()`

zwraca komponent będący celem tego transferu.

- `DataFlavor[] getDataFlavors()`

zwraca formaty transferowanych danych.

- `boolean isDrop()`

zwraca wartość `true`, jeśli transfer jest wynikiem operacji upuszczenia, `false` — jeśli operacji wklejania.

- `int getUserDropAction()`

zwraca akcję upuszczania wybraną przez użytkownika (`MOVE`, `COPY` lub `LINK`).

- `getSourceDropActions()`

zwraca akcje upuszczania dozwolone dla źródła przeciągania.

- `getDropAction()`

- `setDropAction()`

zwraca lub konfiguruje akcję upuszczania dla danego transferu. Początkowo jest to akcja wybrana podczas upuszczania przez użytkownika, ale może zostać zmieniona przez obiekt obsługi transferu.

- `DropLocation getDropLocation()`

zwraca miejsce upuszczenia lub wyrzuca wyjątek `IllegalStateException`, jeśli dany transfer nie jest skutkiem upuszczenia.

**API** javax.swing.TransferHandler.DropLocation 6

- Point getDropPoint()  
zwraca miejsce upuszczenia dla komponentu będącego celem.

**API** javax.swing.JList.DropLocation 6

- boolean isInsert()  
zwraca wartość true, jeśli dane mają zostać wstawione przed danym elementem listy, false — jeśli zastąpią istniejące dane.
- int getIndex()  
zwraca indeks modelu dla wstawienia nowych danych lub zastąpienia starych.

**API** javax.swing.JTable.DropLocation 6

- boolean isInsertRow()
- boolean isInsertColumn()  
zwraca wartość true, jeśli dane mają zostać wstawione przed danym wierszem lub kolumną.
- int getRow()
- int getColumn()  
zwraca indeks wiersza lub kolumny w modelu danych właściwy dla wstawienia nowych danych lub zastąpienia starych. Gdy upuszczenie miało miejsce nad pustym obszarem komponentu, zwraca wartość -1.

**API** javax.swing.JTree.DropLocation 6

- TreePath getPath()
- int getChildIndex()  
zwraca ścieżkę drzewa lub węzeł podrzędny, które wraz z trybem upuszczenia komponentu będącego celem określają miejsce upuszczenia (patrz tabela 7.9).

**Tabela 7.9.** Obsługa miejsca upuszczenia dla komponentu JTree

Tryb upuszczenia	Akcja
INSERT	Wstawienie jako węzeł podrzędny ścieżki, przed indeksem węzła podrzędnego.
ON lub USE_SELECTION	Zastąpienie danych ścieżki (indeks węzła podrzędnego nie jest używany).
INSERT_OR_ON	Jeśli indeks węzła podrzędnego ma wartość -1, akcja jak w trybie ON, w przeciwnym razie jak w trybie INSERT.

**API** javax.swing.JTextComponent.DropLocation **6**

■ int getIndex()

zwraca indeks pozycji, na której mają być wstawione dane.

## 7.15. Integracja z macierzystą platformą

Rozdział zakończymy omówieniem kilku nowości wprowadzonych w wersji Java SE 6, które sprawiają, że aplikacje tworzone w języku Java coraz bardziej przypominają w użytkowaniu aplikacje poszczególnych systemów. Jedną z nich jest możliwość wyświetlania ekranu powitalnego aplikacji podczas ładowania maszyny wirtualnej. Natomiast klasa `java.awt.Desktop` pozwala uruchamiać macierzyste aplikacje systemu, na przykład domyślną przeglądarkę internetową czy klienta poczty elektronicznej. W wersji Java SE 6 uzyskujemy również dostęp do zasobnika systemowego, w którym, podobnie jak czyni to wiele aplikacji macierzystych, możemy umieszczać własne ikony.

### 7.15.1. Ekran powitalny

Jedną z częściej wymienianych wad aplikacji Java jest ich długi czas uruchamiania. Rzeczywiście maszyna wirtualna potrzebuje trochę czasu, aby załadować wszystkie potrzebne klasy, zwłaszcza w przypadku aplikacji Swing, które wymagają obszernego kodu bibliotek Swing i AWT. Może to być nieco irytujące dla użytkownika, który zniecierpliwiony może nawet próbować ponownie uruchomić aplikację, gdyż nie ma żadnej informacji o tym, czy pierwsze uruchomienie dało jakiś rezultat. Rozwiązaniem tej kłopotliwej sytuacji może okazać się wyświetlenie *ekranu powitalnego* aplikacji, czyli niewielkiego okna, które pojawia się bardzo szybko i informuje użytkownika, że aplikacja jest uruchamiana.

Dotychczas wyświetlenie takiego ekranu przez aplikację Java było co najmniej trudne, jeśli nie niemożliwe. Oczywiście kod aplikacji może wyświetlić takie okno, gdy tylko rozpocznie się wykonywanie metody `main`. Problem w tym, że metoda ta jest uruchamiana dopiero po załadowaniu wszystkich klas potrzebnych do jej działania, co zawsze wymaga pewnego czasu.

W wersji Java SE 6 rozwiązano problem ekranu powitalnego, umożliwiając wyświetlenie go przez maszynę wirtualną natychmiast po jej uruchomieniu. Istnieją dwa mechanizmy określenia obrazu wyświetlanego jako ekran powitalny. Możemy użyć opcji `-splash` w wierszu poleceń:

```
java -splash:myimage.png MyApp
```

Alternatywa polega na podaniu pliku zawierającego ekran powitalny w manifeście pliku JAR:

```
Main-Class: MyApp
SplashScreen-Image: myimage.gif
```

Ekran powitalny rzeczywiście zostaje wyświetlony błyskawicznie i znika automatycznie w momencie wyświetlenia pierwszego okna AWT. Ekran powitalny możemy zapisać w formacie GIF, JPEG lub PNG. Animacje (w formacie GIF) oraz przezroczystość (w formatach GIF i PNG) nie są obsługiwane.

Jeśli aplikacje, które tworzysz, są zawsze gotowe do działania w momencie uruchomienia metody `main`, to możesz pominąć lekturę pozostałej części tego podrozdziału. Wiele aplikacji posiada jednak architekturę modułową, w której niewielki rdzeń ładuje po uruchomieniu zbiór wtyczek. Typowymi przykładami takich aplikacji na platformie Java są Eclipse i NetBeans. W przypadku takiej architektury warto informować użytkownika o postępie ładowania modułów, używając w tym celu ekranu powitalnego.

Istnieją dwa podejścia do tego zagadnienia. Możemy albo rysować bezpośrednio na ekranie powitalnym, albo zastąpić go oknem o identycznej zawartości i bez ramki, a następnie rysować wewnątrz okna. Nasz przykładowy program prezentuje zastosowanie obu technik.

Aby rysować bezpośrednio na ekranie powitalnym, musimy pobrać jego referencję, a następnie kontekst graficzny i rozmiary:

```
SplashScreen splash = SplashScreen.getSplashScreen();
Graphics2D g2 = splash.createGraphics();
Rectangle bounds = splash.getBounds();
```

Samo rysowanie grafiki odbywa się w tradycyjny sposób. Po zakończeniu rysowania wywołujemy metodę `update`, aby spowodować odświeżenie ekranu powitalnego. Nasz przykładowy program rysuje prosty pasek postępu pokazany w lewej części rysunku 7.47.

```
g.fillRect(x, y, width * percent / 100, height);
splash.update();
```

#### **Rysunek 7.47.**

*Początkowy ekran powitalny zostaje zastąpiony oknem o takiej samej zawartości*





Ekran powitalny jest singletonem. Nie możemy utworzyć własnego ekranu powitalnego. Jeśli ekran powitalny nie zostanie określony w wierszu poleceń lub w manifestcie, metoda `getSplashScreen` zwróci wartość `null`.

Jednak rysowanie bezpośrednio na ekranie powitalnym ma swoje wady. Wyznaczanie pozycji pikseli jest żmudne i w efekcie nasz wskaźnik postępu znacznie odstaje w działaniu od podobnych rozwiązań stosowanych w macierzystych aplikacjach systemu. Aby uniknąć tego problemu, możemy zastąpić początkowy ekran powitalny oknem tego samego rozmiaru i o tej samej zawartości, jak tylko rozpocznie się wykonywanie metody `main`. Okno to może zawierać dowolne komponenty Swing.

Zastosowanie tej techniki ilustruje przykładowy program przedstawiony na listingu 7.23. W prawej części rysunku 7.47 widzimy okno pozbawione ramki i zawierające panel, który rysuje ekran powitalny oraz zawiera komponent `JProgressBar`. W ten sposób mamy pełen dostęp do interfejsu programowego Swing i możemy łatwo wyświetlać komunikaty bez zajmowania się szczegółami związanymi z wyznaczaniem położenia pikseli.

### Listing 7.23. `splashScreen/SplashScreenTest.java`

```
package splashScreen;

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

/**
 * Program demonstrujący sposób użycia ekranu powitalnego.
 * @version 1.00 2007-09-21
 * @author Cay Horstmann
 */
public class SplashScreenTest
{
    private static final int DEFAULT_WIDTH = 300;
    private static final int DEFAULT_HEIGHT = 300;

    private static SplashScreen splash;

    private static void drawOnSplash(int percent)
    {
        Rectangle bounds = splash.getBounds();
        Graphics2D g = splash.createGraphics();
        int height = 20;
        int x = 2;
        int y = bounds.height - height - 2;
        int width = bounds.width - 4;
        Color brightPurple = new Color(76, 36, 121);
        g.setColor(brightPurple);
        g.fillRect(x, y, width * percent / 100, height);
        splash.update();
    }

    /**
     * Metoda rysująca na ekranie powitalnym.
     */
    private static void init1()
```

```

    {
        splash = SplashScreen.getSplashScreen();
        if (splash == null)
        {
            System.err.println("Did you specify a splash image with -splash or in the
            ↳manifest?");
            System.exit(1);
        }

        try
        {
            for (int i = 0; i <= 100; i++)
            {
                drawOnSplash(i);
                Thread.sleep(100); // symuluje ładowanie aplikacji
            }
        }
        catch (InterruptedException e)
        {
        }
    }

    /**
     * Metoda wyświetlająca ramkę o tej samej zawartości
     * co ekran powitalny.
     */
    private static void init2()
    {
        final Image img = new ImageIcon(splash.getImageURL()).getImage();

        final JFrame splashFrame = new JFrame();
        splashFrame.setUndecorated(true);

        final JPanel splashPanel = new JPanel()
        {
            public void paintComponent(Graphics g)
            {
                g.drawImage(img, 0, 0, null);
            }
        };

        final JProgressBar progressBar = new JProgressBar();
        progressBar.setStringPainted(true);
        splashPanel.setLayout(new BorderLayout());
        splashPanel.add(progressBar, BorderLayout.SOUTH);

        splashFrame.add(splashPanel);
        splashFrame.setBounds(splash.getBounds());
        splashFrame.setVisible(true);

        new SwingWorker<Void, Integer>()
        {
            protected Void doInBackground() throws Exception
            {
                try
                {
                    for (int i = 0; i <= 100; i++)
                    {

```

```

        publish(i);
        Thread.sleep(100);
    }
}
catch (InterruptedException e)
{
}
return null;
}

protected void process(List<Integer> chunks)
{
    for (Integer chunk : chunks)
    {
        progressBar.setString("Loading module " + chunk);
        progressBar.setValue(chunk);
        splashPanel.repaint(); // ponieważ obrazek został załadowany
    }
}

protected void done()
{
    splashFrame.setVisible(false);

    JFrame frame = new JFrame();
    frame.setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setTitle("SplashScreenTest");
    frame.setVisible(true);
}
}.execute();
}

public static void main(String args[])
{
    init1();

    EventQueue.invokeLater(new Runnable()
    {
        public void run()
        {
            init2();
        }
    });
}
}
}

```

Zwróćmy uwagę, że nie musimy usuwać początkowego ekranu powitalnego. Zostaje on automatycznie usunięty, gdy tylko pojawia się okno, które go zastępuje.



Niestety moment zastąpienia ekranu powitalnego przez okno jest dość łatwy do zauważenia.

## API java.awt.SplashScreen

- `static SplashScreen getSplashScreen()`  
zwraca referencję ekranu powitalnego lub wartość `null`, jeśli nie został skonfigurowany.
- `URL getImageURL()`
- `void setImageURL(URL imageURL)`  
zwraca lub określa adres URL obrazka wyświetlanego jako ekran powitalny. Wywołanie drugiej wersji powoduje aktualizację ekranu powitalnego.
- `Rectangle getBounds()`  
zwraca granice ekranu powitalnego.
- `Graphics2D createGraphics()`  
zwraca kontekst graficzny umożliwiający rysowanie na ekranie powitalnym.
- `void update()`  
aktualizuje wyświetlany ekran powitalny.
- `void close()`  
zamyka ekran powitalny. Ekran powitalny zostaje automatycznie zamknięty w momencie wyświetlenia pierwszego okna AWT.

## 7.15.2. Uruchamianie macierzystych aplikacji pulpitu

Klasa `java.awt.Desktop` pozwala uruchomić domyślną przeglądarkę internetową i program obsługi poczty elektronicznej. Możemy również otwierać, edytować i drukować pliki za pomocą aplikacji, które zostały zarejestrowane dla określonych typów plików.

Posługiwanie się wspomnianą klasą jest bardzo proste. Najpierw wywołujemy metodę statyczną `isDesktopSupported`. Jeśli zwróci ona wartość `true`, to platforma na której działa maszyna wirtualna obsługuje uruchamianie aplikacji pulpitu. Następnie wywołujemy metodę statyczną `getDesktop`, aby uzyskać instancję klasy `Desktop`.

Nie wszystkie środowiska wykorzystujące koncepcję pulpitu obsługują wszystkie operacje. Na przykład pulpit Gnome w systemie Linux umożliwia otwieranie plików, ale nie pozwala na ich drukowanie. Aby sprawdzić czy operacja jest dostępna na danej platformie, wywołujemy metodę `isSupported` przekazując jej wartość zdefiniowaną przez wyliczenie `Desktop.Action`. Nasz program zawiera na przykład poniższy test:

```
if (desktop.isSupported(Desktop.Action.PRINT)) printButton.setEnabled(true);
```

Zanim otworzymy, poddamy edycji lub wydrukujemy plik, musimy najpierw sprawdzić czy dana akcja jest obsługiwana, a dopiero potem wywołać metodę `open`, `edit` lub `print`. Aby uruchomić przeglądarkę, należy przekazać obiekt klasy `URI`. (Więcej informacji na temat klasy `URI` znajdziesz w rozdziale 3.). W tym celu wystarczy wywołać konstruktor klasy `URI` i przekazać mu łańcuch URL rozpoczynający się przedrostkiem `http` lub `https`.



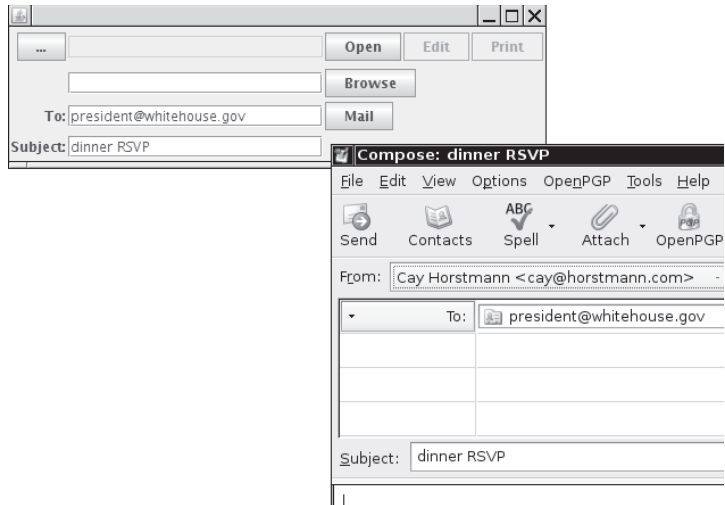
Aby uruchomić domyślny program obsługi poczty elektronicznej, musimy skonstruować obiekt URI w poniższym formacie:

```
mailto:odbiorca?kwerenda
```

Przez odbiorcę rozumiemy w tym przypadku adres poczty elektronicznej, na przykład *president@whitehouse.gov*, a *kwerenda* zawiera pary postaci *nazwa=wartość* rozdzielone znakiem & i zakodowane przy użyciu znaku %. (Kodowanie przy pomocy znaku % jest zasadniczo tożsame z kodowaniem łańcuchów URL omówionym w rozdziale 3., z tą różnicą, że znak spacji jest kodowany jako %20, a nie za pomocą znaku +). Przykładem kwerendy może być łańcuch *subject=dinner%20RSVP&bcc=putin%40kremvax.ru*. Format ten jest opisany szczegółowo w dokumencie RFC 2368 (<http://www.ietf.org/rfc/rfc2368.txt>). Niestety klasa URI nie rozpoznaje łańcuchów *mailto* i wobec tego musimy stworzyć je samodzielnie.

Program przykładowy przedstawiony na listingu 7.24 pozwala użytkownikowi otwierać, edytować i drukować wybrany plik, a także przeglądać podany URL lub uruchamiać klienta poczty elektronicznej (patrz rysunek 7.48).

**Rysunek 7.48.**  
Uruchamianie  
aplikacji pulpitu



**Listing 7.24.** *desktopApp/DesktopAppFrame.java*

```
package desktopApp;

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

class DesktopAppFrame extends JFrame
{
    public DesktopAppFrame()
    {
        setLayout(new GridBagLayout());
        final JFileChooser chooser = new JFileChooser();
        JButton fileChooserButton = new JButton("...");
    }
}
```

```

final JTextField fileField = new JTextField(20);
fileField.setEditable(false);
JButton openButton = new JButton("Open");
JButton editButton = new JButton("Edit");
JButton printButton = new JButton("Print");
final JTextField browseField = new JTextField();
JButton browseButton = new JButton("Browse");
final JTextField toField = new JTextField();
final JTextField subjectField = new JTextField();
JButton mailButton = new JButton("Mail");

openButton.setEnabled(false);
editButton.setEnabled(false);
printButton.setEnabled(false);
browseButton.setEnabled(false);
mailButton.setEnabled(false);

if (Desktop.isDesktopSupported())
{
    Desktop desktop = Desktop.getDesktop();
    if (desktop.isSupported(Desktop.Action.OPEN)) openButton.setEnabled(true);
    if (desktop.isSupported(Desktop.Action.EDIT)) editButton.setEnabled(true);
    if (desktop.isSupported(Desktop.Action.PRINT)) printButton.setEnabled(true);
    if (desktop.isSupported(Desktop.Action.BROWSE)) browseButton.setEnabled(true);
    if (desktop.isSupported(Desktop.Action.MAIL)) mailButton.setEnabled(true);
}

fileChooserButton.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        if (chooser.showOpenDialog(DesktopAppFrame.this) ==
            ↪ JFileChooser.APPROVE_OPTION)
            fileField.setText(chooser.getSelectedFile().getAbsolutePath());
    }
});

openButton.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        try
        {
            Desktop.getDesktop().open(chooser.getSelectedFile());
        }
        catch (IOException ex)
        {
            ex.printStackTrace();
        }
    }
});

editButton.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        try

```

```
        {
            Desktop.getDesktop().edit(chooser.getSelectedFile());
        }
        catch (IOException ex)
        {
            ex.printStackTrace();
        }
    }
}):

printButton.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        try
        {
            Desktop.getDesktop().print(chooser.getSelectedFile());
        }
        catch (IOException ex)
        {
            ex.printStackTrace();
        }
    }
}):

browseButton.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        try
        {
            Desktop.getDesktop().browse(new URI(browseField.getText()));
        }
        catch (URISyntaxException ex)
        {
            ex.printStackTrace();
        }
        catch (IOException ex)
        {
            ex.printStackTrace();
        }
    }
}):

mailButton.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        try
        {
            String subject = percentEncode(subjectField.getText());
            URI uri = new URI("mailto:" + toField.getText() + "?subject=" + subject);

            System.out.println(uri);
            Desktop.getDesktop().mail(uri);
        }
        catch (URISyntaxException ex)
```

```

        {
            ex.printStackTrace();
        }
        catch (IOException ex)
        {
            ex.printStackTrace();
        }
    }
}):

JPanel buttonPanel = new JPanel();
((FlowLayout) buttonPanel.getLayout()).setHgap(2);
buttonPanel.add(openButton);
buttonPanel.add(editButton);
buttonPanel.add(printButton);

add(fileChooserButton, new GBC(0, 0).setAnchor(GBC.EAST).setInsets(2));
add(fileField, new GBC(1, 0).setFill(GBC.HORIZONTAL));
add(buttonPanel, new GBC(2, 0).setAnchor(GBC.WEST).setInsets(0));
add(browseField, new GBC(1, 1).setFill(GBC.HORIZONTAL));
add(browseButton, new GBC(2, 1).setAnchor(GBC.WEST).setInsets(2));
add(new JLabel("To:"), new GBC(0, 2).setAnchor(GBC.EAST).setInsets(5, 2, 5, 2));
add(toField, new GBC(1, 2).setFill(GBC.HORIZONTAL));
add(mailButton, new GBC(2, 2).setAnchor(GBC.WEST).setInsets(2));
add(new JLabel("Subject:"), new GBC(0, 3).setAnchor(GBC.EAST).setInsets(5, 2, 5, 2));
add(subjectField, new GBC(1, 3).setFill(GBC.HORIZONTAL));

pack();
}

private static String percentEncode(String s)
{
    try
    {
        return URLEncoder.encode(s, "UTF-8").replaceAll("[+]", "%20");
    }
    catch (UnsupportedEncodingException ex)
    {
        return null; // UTF-8 jest zawsze obsługiwany
    }
}
}

```

#### API java.awt.Desktop

- static boolean isDesktopSupported()

zwraca wartość true, jeśli platforma obsługuje uruchamianie aplikacji pulpitu.

- static Desktop getDesktop()

zwraca obiekt Desktop umożliwiający wykonywanie operacji pulpitu. Wyrzuca wyjątek `UnsupportedOperationException` jeśli platforma nie obsługuje uruchamianie aplikacji pulpitu.

- `boolean isSupported(Desktop.Action action)`  
zwraca wartość `true` jeśli dana akcja jest obsługiwana. Parametr `action` może przyjmować jedną z wartości `OPEN`, `EDIT`, `PRINT`, `BROWSE` lub `MAIL`.
- `void open(File file)`  
uruchamia aplikację zarejestrowaną do prezentacji zawartości plików danego typu.
- `void edit(File file)`  
uruchamia aplikację zarejestrowaną w celu edycji zawartości plików danego typu.
- `void print(File file)`  
drukuję dany plik.
- `void browse(URI uri)`  
uruchamia domyślną przeglądarkę dla podanego URI.
- `void mail()`
- `void mail(URI uri)`  
uruchamia domyślny program obsługi poczty elektronicznej. Druga z wersji metody może być używana do wypełniania poszczególnych części wiadomości.

### 7.15.3. Zasobnik systemowy

Wiele środowisk wykorzystujących koncepcję pulpitu posiada również specjalny obszar zawierający ikony programów wykonywanych w tle, które okazjonalnie powiadają użytkownika o pewnych zdarzeniach. W systemie Windows obszar ten nosi nazwę *zasobnika systemowego*, a znajdujące się w nim ikony nazwane zostały *ikonami zasobnika*. Na platformie Java przyjęto tę samą terminologię. Typowym przykładem takiego programu jest monitor sprawdzający dostępność nowych wersji oprogramowania. Jeśli pojawi się aktualizacja oprogramowania, monitor zmienia wygląd swojej ikony bądź wyświetla komunikat w jej pobliżu.

Zasobnik systemowy bywa nadużywany przez twórców oprogramowania i użytkownicy zwykle nie są zachwyceni odkryciem jeszcze jednej ikony zasobnika. Nasz przykładowy program nie stanowi wyjątku od tej reguły.

Klasa `java.awt.SystemTray` umożliwia wykorzystanie zasobnika systemowego na różnych platformach. Podobnie jak w przypadku klasy `Desktop` omówionej w poprzednim podrozdziale, najpierw wywołujemy metodę statyczną `isSupported`, aby sprawdzić czy dana platforma obsługuje zasobnik systemowy. Jeśli tak, pobieramy singleton `SystemTray` wywołując w tym celu metodę statyczną `getSystemTray`.

Najważniejszą metodą klasy `SystemTray` jest oczywiście metoda `add` pozwalająca dodawać instancję klasy `TrayIcon`. Instancja taka posiada trzy właściwości:

- Obrazek ikony.
- Podpowiedź wyświetlaną gdy kursor myszy przesuwany nad ikoną.
- Podręczne menu wyświetlane, gdy użytkownik kliknie ikonę prawym klawiszem myszki.

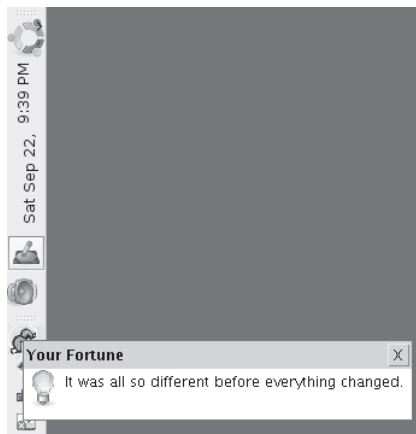
Menu podręczne jest instancją klasy `PopupMenu` z biblioteki AWT, reprezentującej menu macierzystej platformy, a nie menu Swing. Do menu dodajemy instancje klasy `MenuItem` (również z biblioteki AWT), z których każda posiada obiekt nasłuchujący akcji podobnie jak jej odpowiednik z biblioteki Swing.

Ikona zasobnika może wyświetlać powiadomienia (patrz rysunek 7.49). W tym celu wywołujemy metodę `displayMessage` określając nagłówek, tekst powiadomienia i jego typ.

```
trayIcon.displayMessage("Your Fortune", fortunes.get(index),
↳ TrayIcon.MessageType.INFO);
```

### Rysunek 7.49.

Powiadomienie  
wyświetlane przez  
ikonę zasobnika



Listing 7.25 przedstawia kod aplikacji, która umieszcza w zasobniku systemowym ikonę wyświetlającą sentencje, których źródłem jest plik (utworzony za pomocą nieśmiertelnego programu `fortune` z systemu Unix). Poszczególne sentencje są oddzielone wierszem zawierającym znak `%`. Program wyświetla kolejną sentencję co 10 sekund. Na szczęście posiada również menu umożliwiające zakończenie jego pracy. Gdyby tylko wszyscy twórcy ikon zasobnika byli równie przewidujący!

### Listing 7.25. `systemTray/SystemTrayTest.java`

```
package systemTray;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.util.List;
import javax.swing.*;
import javax.swing.Timer;

/**
 * Program demonstrujący wykorzystanie zasobnika systemowego.
 * @version 1.01 2012-01-26
 * @author Cay Horstmann
 */
public class SystemTrayTest
{
    public static void main(String[] args)
```

```

    {
        SystemTrayApp app = new SystemTrayApp();
        app.init();
    }
}

class SystemTrayApp
{
    public void init()
    {
        final TrayIcon trayIcon;

        if (!SystemTray.isSupported())
        {
            System.err.println("System tray is not supported.");
            return;
        }

        SystemTray tray = SystemTray.getSystemTray();
        Image image = new ImageIcon(getClass().getResource("cookie.png")).getImage();

        PopupMenu popup = new PopupMenu();
        MenuItem exitItem = new MenuItem("Exit");
        exitItem.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                System.exit(0);
            }
        });
        popup.add(exitItem);

        trayIcon = new TrayIcon(image, "Your Fortune", popup);

        trayIcon.setImageAutoSize(true);
        trayIcon.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                trayIcon.displayMessage("How do I turn this off?",
                    "Right-click on the fortune cookie and select Exit.",
                    TrayIcon.MessageType.INFO);
            }
        });

        try
        {
            tray.add(trayIcon);
        }
        catch (AWTException e)
        {
            System.err.println("TrayIcon could not be added.");
            return;
        }

        final List<String> fortunes = readFortunes();
        Timer timer = new Timer(10000, new ActionListener())

```

```

        {
            public void actionPerformed(ActionEvent e)
            {
                int index = (int) (fortunes.size() * Math.random());
                trayIcon.displayMessage("Your Fortune", fortunes.get(index),
                    TrayIcon.MessageType.INFO);
            }
        });
        timer.start();
    }

    private List<String> readFortunes()
    {
        List<String> fortunes = new ArrayList<>();
        try (InputStream inStream = getClass().getResourceAsStream("fortunes"))
        {
            Scanner in = new Scanner(inStream);
            StringBuilder fortune = new StringBuilder();
            while (in.hasNextLine())
            {
                String line = in.nextLine();
                if (line.equals("%"))
                {
                    fortunes.add(fortune.toString());
                    fortune = new StringBuilder();
                }
                else
                {
                    fortune.append(line);
                    fortune.append(' ');
                }
            }
        }
        catch (IOException ex)
        {
            ex.printStackTrace();
        }
        return fortunes;
    }
}

```

#### API java.awt.SystemTray

- `static boolean isSupported()`  
zwraca wartość `true`, jeśli dana platforma udostępnia zasobnik systemowy.
- `static SystemTray getSystemTray()`  
zwraca obiekt `SystemTray` umożliwiający dostęp do zasobnika systemowego. Wyrzuca wyjątek `UnsupportedOperationException` jeśli platforma nie udostępnia zasobnika systemowego.
- `Dimension getTrayIconSize()`  
zwraca rozmiary ikony w zasobniku systemowym.



- `void add(TrayIcon trayIcon)`
- `void remove(TrayIcon trayIcon)`  
dodaje lub usuwa ikonę w zasobniku systemowym.

#### API `java.awt.TrayIcon`

- `TrayIcon(Image image)`
- `TrayIcon(Image image, String tooltip)`
- `TrayIcon(Image image, String tooltip, PopupMenu popupMenu)`  
tworzy ikonę zasobnika o podanym obrazku, tekście podpowiedzi i menu podręcznym.
- `Image getImage()`
- `void setImage(Image image)`
- `String getTooltip()`
- `void setTooltip(String tooltip)`
- `PopupMenu getPopupMenu()`
- `void setPopupMenu(PopupMenu popupMenu)`  
zwracają lub określają obrazek ikony, tekst podpowiedzi i menu podręczne.
- `boolean isImageAutoSize()`
- `void setImageAutoSize(boolean autosize)`  
zwracają lub konfigurują właściwość `imageAutoSize`. Jeśli właściwość jest skonfigurowana, obrazek zostaje przeskalowany do obszaru ikony. Jeśli nie (domyślnie) zostaje przycięty (jeśli jest zbyt duży) lub wycentrowany (jeśli jest za mały).
- `void displayMessage(String caption, String text, TrayIcon.MessageType messageType)`  
wyświetla powiadomienie w pobliżu ikony zasobnika. Parametr `messageType` może przyjmować jedną z wartości `INFO`, `WARNING`, `ERROR` lub `NONE`.
- `public void addActionListener(ActionListener listener)`
- `public void removeActionListener(ActionListener listener)`  
dodaje lub usuwa obiekt nasłuchujący akcji zależnej od konkretnej platformy. Typowym przypadkiem jest kliknięcie powiadomienia lub podwójne kliknięcie ikony w zasobniku systemowym.

W ten sposób dotarliśmy do końca tego długiego rozdziału poświęconego zaawansowanym możliwościom AWT. W następnym rozdziale przedstawimy specyfikację JavaBeans i jej wykorzystanie przez narzędzia budowy interfejsów użytkownika.



# Skorowidz

## A

- abort(), 776
- absolute(), 268
- AbstractButton, 673
- AbstractCellEditor, 391, 392
- AbstractListModel, 350
- AbstractTableModel, 363, 385
- accept(), 191, 193
- acceptChanges(), 272, 273
- ActionListener, 835
- ActionListenerFor, 835, 838
- ActionListenerInstaller, 835
  - processAnnotations(), 835
- actionPerformed(), 835
- Activatable, 885, 889
  - exportObject(), 886, 889
  - register(), 886, 889
- ActivationDesc, 890
- ActivationGroup, 890
  - getSystem(), 890
- ActivationGroupDesc, 890
- ActivationSystem, 890
  - registerGroup(), 890
- add(), 403, 610
- addActionListener(), 833, 836
- addAttribute(), 181
- addBatch(), 286
- addCellEditorListener(), 394
- addChangeListener(), 479, 483
- addClass(), 420
- addColumn(), 374, 380
- addElement(), 354, 355
- addFlavorListener(), 617
- addHyperlinkListener(), 462
- addListSelectionListener(), 349
- addPropertyChangeListener(), 680, 682, 699
- addSelectionListener(), 415
- addTab(), 478, 482
- addTableModelListener(), 426, 428
- addVetoableChangeListener(), 495, 500, 676, 681, 682
- adnotacje
  - @ActionListenerFor, 832, 843
  - @BugReport, 844
  - @Deprecated, 841, 842
  - @Documented, 841, 844
  - @Inherited, 841, 844
  - @interface, 832
  - @LogEntry, 852
  - @Override, 841, 842
  - @Persistent, 845
  - @Property, 845
  - @Retention, 832, 841, 843
  - @Serializable, 844
  - @SuppressWarnings, 841, 842
  - @Target, 832, 841, 843
  - @TestCase, 831
- ActionListenerFor, 835, 838
- cykliczne zależności, 839
- deklaracja elementu, 837
- elementy, 831
- format, 837
- interfejs, 831, 837
- kod bajtowy, 851
- kolejność elementów, 837
- metaadnotacje, 843
- metody, 831
- modyfikacja kodu bajtowego podczas ładowania, 857
- obsługa zdarzeń, 832
- pojedyncze wartości, 838
- przetwarzanie, 845
- regularne, 842
- składnia, 837
- skróty, 838
- standardowe, 841

- adnotacje
  - typy elementów, 839, 843
  - zmienne lokalne, 840
  - znacznikowe, 838
- adres internetowy, 189
- adres localhost, 193
- adres URI, 205
- adres URL, 148, 204, 205, 458, 750
- AES, 796, 797, 803
- AffineTransform, 536, 537
  - getRotateInstance(), 537, 538
  - getScaleInstance(), 538
  - getShearInstance(), 538
  - getTranslateInstance(), 538
  - setToRotation(), 538
  - setToScale(), 538
  - setToShear(), 538
  - setToTranslation(), 538
- AffineTransformOp, 572, 578
  - TYPE\_BILINEAR, 572
  - TYPE\_NEAREST\_NEIGHBOR, 572
- afterLast(), 269
- aktualizacje wsadowe, 284
- aktualizowalne zbiory wyników zapytań, 263, 265
- aktywacja obiektów serwera, 884
- alfa, 542
- algorytmy
  - AES, 796, 797, 803
  - DES, 795
  - DSA, 780
  - kryptograficzne, 728
  - MD5, 777, 778
  - RSA, 780, 803
  - SHA1, 777
  - szyfrowania, 795
  - z kluczem symetrycznym, 803
- AllPermission, 752
- AllPermissions, 746, 755
- AlphaComposite, 544
  - getInstance(), 549
- animacje GIF, 562
- AnnotatedElement, 835, 836
  - getAnnotation(), 836
  - getAnnotations(), 836
  - getDeclaredAnnotations(), 836
  - isAnnotationPresent(), 836
- Annotation, 838, 840
  - annotationType(), 840
  - equals(), 840
  - hashCode(), 841
  - toString(), 841
- annotationType(), 840
- antialiasing, 549
- Apache, 288
- aplety, 230, 663, 776
  - lokalizacja, 328
- aplikacje interaktywne, 198
- aplikacje klient-serwer, 230
- aplikacje rozproszone, 861
- append(), 520, 523, 599
- appendChild(), 160, 164
- applets.policy, 792
- appletviewer, 743, 748, 790
- applyPattern(), 319
- apt, 845
- Arc2D, 509, 511
- Arc2D.Double, 522
- architektura JDBC, 228
- architektura klient-serwer, 230
- architektura n-warstwowa, 230
- architektura trójwarstwowa, 230
- ArcMaker, 521
- Area, 523, 524
- ARGB, 567, 571
- arkusz właściwości, 658
- Array, 287
- ARRAY, 287
- ArrayIndexOutOfBoundsException, 923
- ArrayList, 427, 885
- ArrayStoreException, 923
- ASCII, 291
- ASP, 216
- atrybuty, 105, 122
- atrybuty drukowania, 604, 607
  - dokumenty, 604
  - hierarchia, 605
  - klasy, 607
  - usługi drukowania, 604
  - wydruk, 604
  - zbiór, 606
  - żądanie wydruku, 604
- AttributeSet, 604
- ATTLIST, 122, 128
- Attribute, 604, 606, 609
  - getCategory(), 609
  - getName(), 609
- Attributes, 155
  - getLength(), 155
  - getLocalName(), 155
  - getQName(), 156
  - getURI(), 156
  - getValue(), 156
- AttributeSet, 610
  - add(), 610
  - get(), 610
  - remove(), 610
  - toArray(), 610
- AttributesImpl, 181
- addAttribute(), 181

AudioPermission, 752  
 AuthPermission, 752  
 AuthTest.policy, 766  
 AWT  
   drukowanie, 580  
   figury, 508  
   filtrowanie obrazów, 571  
   Graphics, 505, 508  
   obszar przycięcia, 507  
   operacje na obrazach, 565  
   pliki graficzne, 555  
   pola, 523  
   potokowe tworzenie grafiki, 506  
   przeciagnij i upuść, 625  
   przekształcenia układu współrzędnych, 507, 534  
   przezroczystość, 541  
   prycinanie, 539  
   rysowanie figur, 506  
   schowek, 610  
   składanie obrazów, 541  
   ślad pędzla, 507, 524  
   wskazówki operacji graficznych, 549  
   wypełnianie obszaru, 507, 532  
   zasady składania obrazów, 507  
 AWTPermission, 751

## B

Banner, 590  
 Base64, 210  
 base64Encode(), 209  
 Base64Encoder, 214  
 BasicPermission, 752  
 BasicStroke, 524, 525, 530  
 baza danych, 227  
   adres URL, 237  
   aktualizowalne zbiory wyników zapytań, 265  
   JNDI, 288  
   kolumny, 231  
   kursory, 264  
   łączenie tabel, 232, 233  
   metadane, 274  
   model dostępu, 228  
   modyfikacja danych, 235  
   ODBC, 228  
   przewijanie zbioru rekordów, 264  
   rekordy, 231  
   rekordy wstawiania, 266  
   spójność, 283  
   SQL, 227  
   tabele, 231  
   transakcje, 283  
   wstawianie danych, 235  
   wypełnianie, 248  
   zapytania, 232  
   zarządzanie połączeniami, 288  
   zbiory rekordów, 270  
 BCEL, 852  
 BEA WebLogic, 288  
 Bean Builder, 666  
 BeanClass, 846  
 BeanDescriptor, 699  
 BeanInfo, 683, 684, 685, 698  
   getBeanDescriptor(), 698  
   getIcon(), 685  
   getPropertyDescriptors(), 685, 690  
 BeanInfoAnnotationProcessor, 850  
 beforeFirst(), 268  
 bezpieczeństwo  
   hierarchia klas pozwoleń, 745  
   JAAS, 762  
   Java 2, 744  
   java.policy, 748  
   Kerberos, 806  
   klasy pozwoleń, 755  
   kryptografia klucza publicznego, 803  
   ładowanie klas, 728  
   menedżer bezpieczeństwa, 728, 742  
   piaskownica, 744  
   pliki polityki, 744, 747  
   podpis cyfrowy, 776  
   podpisywanie kodu, 788  
   polityka bezpieczeństwa, 745  
   pozwolenia, 742, 745  
   przydzielanie praw, 744  
   skrótów wiadomości, 777  
   SSL, 806  
   szyfrowanie, 795  
   uwierzytelnianie użytkowników, 762  
   uwierzytelnianie wiadomości, 784  
   weryfikacja kodu maszyny wirtualnej, 738  
   źródło kodu, 744  
 biblioteki  
   BCEL, 852  
   DLL, 895  
   JCE, 801  
 BigDecimal, 287  
 bind(), 870, 871  
 BitSet, 713  
 BLOB, 236, 287  
 Book, 598, 599  
   append(), 599  
   getPrintable(), 599  
 boolean, 236, 287  
 breadthFirstEnumeration(), 410, 414  
 Buffer, 198  
 BufferedImage, 533, 545, 565, 570  
   getColorModel(), 570  
   getRaster(), 570

BufferedImageOp, 565, 571, 572, 578  
 filter(), 578  
 buforowane zbiory rekordów, 271  
 ButtonFrame, 835  
 ByteLookupTable, 576, 579

**C**

C, 892  
 łańcuchy znakowe platformy Java, 902  
 obsługa błędów, 923, 927  
 tablice, 919  
 wywoływanie metod języka Java, 912, 917  
 wywoływanie metod statycznych, 916  
 C++, 895  
 CachedRowSet, 270, 271, 273  
 acceptChanges(), 273  
 execute(), 273  
 getTableName(), 273  
 populate(), 273  
 setTableName(), 273  
 Caesar, 736  
 CalendarBean, 661  
 Callback, 773  
 CallbackHandler, 775  
 handle(), 775  
 CallNonVirtualXxxMethod(), 917, 918  
 CallNonVirtualXxxMethodA(), 918  
 CallNonVirtualXxxMethodV(), 918  
 CallStaticObjectMethod(), 916  
 CallStaticXxxMethod(), 916, 918  
 CallStaticXxxMethodA(), 918  
 CallStaticXxxMethodV(), 918  
 CallXxxMethod(), 917  
 CallXxxMethodA(), 917  
 CallXxxMethodV(), 918  
 cancelCellEditing(), 391, 392, 394  
 cancelRowUpdates(), 269  
 canInsertImage(), 561, 564  
 catch, 246  
 CDATA, 106, 123  
 CellEditor, 394  
 addCellEditorListener(), 394  
 cancelCellEditing(), 394  
 getCellEditorValue(), 394  
 isCellEditable(), 394  
 removeCellEditorListener(), 394  
 shouldSelectCell(), 394  
 stopCellEditing(), 394  
 certyfikaty, 744  
 certyfikaty twórców oprogramowania, 793  
 certyfikaty X.509  
 keytool, 781  
 komponenty, 782  
 podpisywanie, 786

składnica kluczy, 782  
 sprawdzanie wiarygodności, 783  
 wydawcy certyfikatów, 783  
 zarządzanie, 781  
 CGI, 216  
 Channels, 204  
 newInputStream(), 204  
 newOutputStream(), 198, 204  
 CHAR, 236, 287  
 CHARACTER, 236  
 CharacterData, 118  
 getData(), 118  
 characters(), 151, 155  
 ChartBean, 846  
 ChartBeanBeanInfo, 688, 845  
 checkExit(), 743, 744, 746  
 checkLogin(), 769  
 checkPermission(), 747, 755, 756  
 children(), 409  
 ChoiceFormat, 321  
 Chromaticity, 607  
 Cipher, 795, 800  
 doFinal(), 801  
 getBlockSize(), 800  
 getInstance(), 800  
 getOutputSize(), 800  
 init(), 800  
 update(), 800  
 CipherInputStream, 802  
 read(), 802  
 CipherOutputStream, 802  
 flush(), 802  
 write(), 802  
 Class, 411, 606, 737, 747, 835  
 getClassLoader(), 737  
 getProtectionDomain(), 747  
 ClassLoader, 729, 733, 737  
 defineClass(), 738  
 findClass(), 738  
 getParent(), 737  
 getSystemClassLoader(), 737  
 ClassNameTreeCellRenderer, 413  
 CLASSPATH, 729  
 ClassTreeFrame, 414  
 CLEAR, 543  
 clearParameters(), 258  
 clip(), 507, 539, 541  
 Clipboard, 611, 614, 617, 625  
 addFlavorListener(), 617  
 getAvailableDataFlavors(), 617  
 getContents(), 614  
 getData(), 615  
 getTransferDataFlavors(), 617  
 isDataFlavorAvailable(), 615  
 setContents(), 615

- ClipboardOwner, 612, 615
    - lostOwnership(), 615
  - Clob, 287
  - CLOB, 236, 287
  - clone(), 884
  - CloneNotSupportedException, 884
  - close(), 191, 193, 243, 244, 245, 467, 474
  - closePath(), 514, 523
  - CodeSource, 747
    - getCertificates(), 747
    - getLocation(), 747
  - CollationKey, 318
    - compareTo(), 318
  - Collator, 312, 317
    - compare(), 317
    - equals(), 317
    - getAvailableLocales(), 317
    - getCollationKey(), 318
    - getDecomposition(), 317
    - getInstance(), 317
    - getStrength(), 317
    - PRIMARY, 312
    - setDecomposition(), 317
    - setStrength(), 317
  - Collections
    - sort(), 312
  - Color, 532, 542, 567, 621
    - getRGB(), 571
  - ColorConvertOp, 577
  - ColorModel, 568, 571
    - getDataElements(), 571
    - getRGB(), 571
  - ColorSupported, 607
  - column(), 245
  - commit(), 284, 286, 776
  - Common Object Request Broker Architecture, 863
  - CommonDialog, 658
  - Comparator, 312
  - compare(), 317
  - compareTo(), 311, 312, 318
  - Compression, 607
  - CONCUR\_READ\_ONLY, 264
  - CONCUR\_UPDATABLE, 264, 266
  - connect(), 188, 207, 215
  - Connection, 245, 258, 268, 282
    - close(), 243
    - commit(), 286
    - createStatement(), 243, 268
    - getAutoCommit(), 285
    - getMetaData(), 282
    - prepareStatement(), 258, 268
    - releaseSavepoint(), 286
    - rollback(), 286
    - setAutoCommit(), 285
    - setSavepoint(), 286
  - Constructor, 835
  - containsAll(), 759
  - ContentHandler, 151, 152, 155
    - characters(), 155
    - endDocument(), 155
    - endElement(), 155
    - startDocument(), 155
    - startElement(), 155
  - Context, 870
    - bind(), 870
    - list(), 871
    - lookup(), 870
    - rebind(), 871
    - unbind(), 870
  - CONTIGUOUS\_TREE\_SELECTION, 415
  - ConvolveOp, 578, 579
  - Copies, 604, 606, 607
  - CopiesSupported, 604
  - CORBA, 863
  - COREJAVA, 236
  - CREATE TABLE, 235, 242, 249
  - createElement(), 160, 163
  - createImageInputStream(), 560, 563
  - createImageOutputStream(), 561, 563
  - createPrintJob(), 602
  - createStatement(), 242, 243, 263, 268, 284, 285
  - createTextNode(), 160, 164
  - CubicCurve2D, 509, 513, 522
  - Currency, 302, 303
    - getCurrencyCode(), 303
    - getDefaultFractionsDigits(), 303
    - getInstance(), 303
    - getSymbol(), 303
    - toString(), 303
  - curveTo(), 513, 514, 522
  - Customizer, 699
    - setObject(), 705
  - Cygwin, 895, 932
  - czas, 304
  - czcionki, 523, 540
    - obrys, 540
- D**
- DA, 607
  - DefaultTreeModel, 404
  - data, 292, 304
    - parsowanie, 308
  - database.properties, 288
  - DatabaseMetaData, 245, 265, 269, 274, 275, 286
    - getJDBCMinorVersion(), 282
    - getJDBCMinorVersion(), 282
    - getMaxConnections(), 282
    - getMaxStatements(), 282

- getTables(), 282
- supportsBatchUpdates(), 286
- supportsResultSetConcurrency(), 270
- supportsResultSetType(), 269
- DataFlavor, 611, 615, 616
  - getHumanPresentableName(), 617
  - getMIMETYPE(), 616
  - getRepresentationClass(), 617
  - imageFlavor, 617, 618
  - isMimeTypeEqual(), 617
- DataSource, 288
- Date, 287
- DATE, 236, 287
- DateFormat, 295, 304, 305
  - DEFAULT, 304
  - format(), 310
  - FULL, 304
  - getAvailableLocales(), 309
  - getCalendar(), 310
  - getDateInstance(), 304, 309
  - getDateTimelInstance(), 309
  - getNumberFormat(), 310
  - getTimeInstance(), 304, 309
  - getTimeZone(), 310
  - isLenient(), 310
  - LONG, 304
  - MEDIUM, 304
  - parse(), 310
  - setCalendar(), 310
  - setLenient(), 310
  - setNumberFormat(), 310
  - setTimeZone(), 310
  - SHORT, 304
- DateFormat.getDateTimelInstance(), 304
- DateTimeAtCompleted, 607
- DateTimeAtCreation, 607
- DateTimeAtProcessing, 607
- DDL, 244
- DEC, 287
- DECIMAL, 236, 287
- decode(), 222
- DefaultCellEditor, 389, 392, 393, 406
- DefaultHandler, 152
- DefaultListModel, 354, 355
  - addElement(), 355
  - removeElement(), 355
- DefaultModelList, 354
- DefaultMutableTreeNode, 396, 401, 402, 410, 414
  - add(), 403
  - breadthFirstEnumeration(), 414
  - depthFirstEnumeration(), 414
  - postOrderEnumeration(), 414
  - preOrderEnumeration(), 414
  - setAllowsChildren(), 403
  - defaultPage(), 588
- DefaultPersistenceDelegate, 711, 713, 725
  - initialize(), 725
  - instantiate(), 725
- DefaultTreeCellRenderer, 412, 413, 414
  - setClosedIcon(), 414
  - setLeafIcon(), 414
  - setOpenIcon(), 414
- DefaultTreeModel, 396, 402, 405, 409, 422
  - insertNodeInto(), 409
  - nodeChanged(), 410
  - reload(), 410
  - removeNodeFromParent(), 410
  - setAsksAllowsChildren(), 402
- defineClass(), 733, 738
- definicja typu dokumentu, 119
- deklaracja typu dokumentu, 104
- delegat trwałości, 710
- DELETE, 242
- deleteRow(), 267, 269
- depthFirstEnumeration(), 410, 414
- depthFirstTraversal(), 411
- DES, 795, 796
- deskryptory aktywacji, 884
- deskryptory wdrożenia, 830
- Destination, 607
- DestroyJavaVM(), 928, 932
- DialogCallbackHandler, 772
- digest(), 779
- DISCONTIGUOUS\_TREE\_SELECTION, 415
- DLL, 895
- doAs(), 763, 764, 767
- doAsPrivileged(), 763, 764, 767
- Doc, 601
- DocAttribute, 604, 607
- DocAttributeSet, 606
- DocFlavor, 599
- DocPrintJob, 601, 602
  - getAttributes(), 610
  - print(), 602
- DOCTYPE, 120, 161
- Document, 108, 111, 117, 163
  - createElement(), 163
  - createTextNode(), 164
  - getDocumentElement(), 117
- DocumentBuilder, 107, 111, 116, 125, 160, 163
  - parse(), 116
  - setEntityResolver(), 125
  - setErrorHandler(), 125
- DocumentBuilderFactory, 116, 124, 126, 149, 150
  - isIgnoringElementContentWhitespace(), 126
  - isNamespaceAware(), 150
  - isValidating(), 126
  - newDocumentBuilder(), 116



- newInstance(), 116
- setIgnoringElementContentWhitespace(), 126
- setNamespaceAware(), 150
- setValidating(), 126
- DocumentName, 607
- doFinal(), 796, 801
- dokumenty XML, 104
  - analiza zawartości, 108
  - atrybuty, 105
  - CDATA, 106
  - deklaracja typu dokumentu, 104
  - DOCTYPE, 120
  - DTD, 106
  - element korzenia, 104
  - elementy, 108
  - elementy podrzędne, 109, 116
  - komentarze, 106
  - kontrola poprawności, 118
  - nagłówki, 104
  - NodeList, 108
  - parsowanie, 107
  - PCDATA, 121
  - pobieranie węzłów, 110
  - przeglądanie atrybutów, 110
  - tworzenie, 160
  - tworzenie drzewa DOM, 160
  - wczytywanie, 107
  - wyszukiwanie informacji, 142
  - XML Schema, 126
  - XPath, 142
- DOM, 107, 120
  - analiza drzewa, 111
  - drzewo, 109
  - tworzenie drzewa, 160
- domena ochronna, 746
- DOMResult, 177, 181
- DOMSource, 165, 176
- DOMTreeModel, 116
- domyślny edytor komórki, 406
- doPost(), 221
- dostęp do składowych, 906
  - poła instancji, 910
  - poła statyczne, 910
- double, 236, 287
- DoubleArrayEditor, 688
- draw(), 507, 508, 509
- draw3DRect(), 508, 509
- drawArc(), 508
- drawLine(), 508
- drawOval(), 508
- drawPolygon(), 508, 509
- drawPolyline(), 508
- drawRect(), 506
- drawRectangle(), 508
- drawRoundRect(), 508
- DriverManager, 242, 288
  - getConnection(), 242
  - setLogWriter(), 242
- DROP TABLE, 242
- drukowanie, 363, 580, 610
  - algorytm rozmieszczenia materiału, 589
  - atrybuty, 604
  - format strony, 582
  - grafika, 580
  - JDK, 580
  - liczba stron, 582
  - marginesy, 583
  - podgląd wydruku, 591
  - PostScript, 603
  - Printable, 580
  - przycięcie kontekstu graficznego, 582
  - rozmieszczenie materiału na stronach, 590
  - transparent, 590
  - usługi, 599
  - wiele stron, 589
  - wymiary strony, 583
  - zadania, 581
- drzewa, 394
  - domyślny edytor komórki, 406
  - dziedziczenie, 411
  - edycja węzłów, 406
  - ikony liści, 401
  - JTree, 394, 395
  - kolejność przeglądania węzłów, 410
  - korzeń, 395, 397
  - las, 395, 400
  - liście, 395
  - model, 396
  - modyfikacja ścieżek, 403
  - modyfikacje, 403
  - nasłuchiwanie zdarzeń, 415
  - obiekt nasłuchujący wyboru, 415
  - obiekt rysujący komórki, 412
  - obiekt użytkownika, 396
  - powiązania między węzłami, 397
  - przeglądanie od końca, 410
  - przeglądanie w głąb, 410
  - przeglądanie węzłów, 410
  - przewijalny panel, 405
  - rozwijanie ścieżek, 405
  - rysowanie węzłów, 412
  - struktura, 427
  - ścieżki, 403
  - tworzenie modeli, 421
  - węzły, 395
  - węzły nadrzędne, 395
  - węzły podrzędne, 395
  - wstawianie węzłów, 406

drzewa  
  wygląd, 398, 412  
  zdarzenia, 415  
  zmiana struktury węzła, 404  
  zwinienie, 400  
DST, 543  
DST\_ATOP, 543  
DST\_IN, 543  
DST\_OUT, 543  
DST\_OVER, 543  
DTD, 106, 119, 124  
DTDHandler, 152  
dynamiczne ładowanie klas, 878  
dziedziczenie, 411

## E

edycja plików polityki, 754  
edytor rejestru, 933  
edytor właściwości, 667, 687  
  graficzny, 694  
  implementacja, 690  
  tworzenie, 687  
  właściwości proste, 690  
  złożone właściwości, 693  
EJB, 658  
Element, 117, 164  
  getAttribute(), 117  
  getTagName(), 117  
  setAttribute(), 164  
  setAttributeNS(), 164  
ELEMENT, 119, 121, 131  
elipsy, 521  
Ellipse2D, 509, 511  
EmployeeReader, 177  
encode(), 222  
Encoder, 724  
  getExceptionListener(), 724  
  getPersistenceDelegate(), 724  
  setExceptionListener(), 724  
  setPersistenceDelegate(), 724  
endDocument(), 151, 155  
endElement(), 151, 155  
Enterprise JavaBeans, 658  
ENTITY, 123  
EntityResolver, 107, 125, 152  
  resolveEntity(), 125  
EntryLogger, 854, 858  
Enumeration, 243  
EnumSyntax, 607  
env, 901, 928  
equals(), 317, 759, 840, 884  
error(), 125, 126  
ErrorHandler, 125, 126, 152  
  error(), 126

  fatalError(), 126  
  warning(), 126  
evaluate(), 143, 147  
Event, 671  
EventHandler, 709, 836  
EventListenerList, 426  
EventObject, 673  
EventSetDescriptor, 684  
ExceptionCheck(), 927  
ExceptionClear(), 927  
ExceptionListener, 724  
  exceptionThrown(), 724  
ExceptionOccured(), 923, 924, 927  
exceptionThrown(), 724  
ExecSQL, 249  
execute(), 244, 271, 273  
executeBatch(), 285, 286  
executeQuery(), 242, 244, 253, 258  
executeUpdate(), 242, 244, 253, 258, 284  
exit(), 743  
exitInternal(), 744  
exportObject(), 867, 886, 889  
Expression, 725  
extern "C", 895

## F

fatalError(), 125, 126  
FeatureDescriptor, 685  
  getDisplayName(), 685  
  getName(), 685  
  getShortDescription(), 686  
  isExpert(), 686  
  isHidden(), 686  
  setDisplayName(), 685  
  setExpert(), 686  
  setHidden(), 686  
  setName(), 685  
  setShortDescription(), 686  
Fidelity, 607  
Field, 835, 915  
figury, 506, 508, 509, 510, 521  
  łuk, 511  
  odcinki, 513  
  prostokąt, 511  
  przycinanie, 539  
  punkty kontrolne, 521  
  rysowanie, 506  
  tworzenie, 521  
  wielokąty, 514  
  wypełnianie, 508, 532  
File, 397  
FileInputStream, 470, 746  
FilePermission, 745, 750  
FileReader, 746

fill(), 507, 508  
 fillOval(), 506  
 Filter, 659  
 filter(), 578  
 FilteredRowSet, 270  
 filtrowanie obrazów, 571  
   interpolacja, 572  
   negatyw, 576  
   rozmycie, 577  
   wykrywanie krawędzi, 578  
 findClass(), 733, 738  
 FindClass(), 910  
 findEditor(), 700  
 Finishings, 607  
 fireIndexedPropertyChange(), 680  
 firePropertyChange(), 675, 680, 682  
 fireVetoableChange(), 677, 682  
 first(), 268  
 FIXED, 123  
 FlavorListener, 616  
   flavorsChanged(), 617  
 flavorsChanged(), 617  
 float, 287  
 FLOAT, 236, 287  
 flush(), 802  
 focusCycleRoot, 667  
 FontType, 128  
 Format, 320  
 format liczby, 297  
 format XML, 864  
 format(), 301, 310, 319, 320  
 formatki, 658, 666  
   tworzenie, 666  
 formatowanie komunikatów, 318  
   warianty, 320  
 formatowanie liczb, 297, 899  
 formularze, 216  
   HTML, 217  
   metoda GET, 218  
   metoda POST, 218  
   nazwy pól, 219  
   odpowiedź serwera, 221  
   przetwarzanie danych, 216  
   serwlety, 216  
   skrypty CGI, 216  
   Submit, 216  
   wysyłanie informacji do serwera, 218  
 fprint(), 924  
 fprintf(), 912  
 fraktale, 568  
 FROM, 234  
 FTP, 209  
 funkcje języka C, 892

## G

GeneralPath, 509, 513, 520, 521, 522, 523, 530, 540  
   append(), 523  
   closePath(), 523  
   curveTo(), 522  
   lineTo(), 522  
   moveTo(), 522  
   quadTo(), 522  
 generateKey(), 797, 801  
 generowanie liczb losowych, 797  
 geometria pól, 523  
 GET, 218  
 get(), 610, 671  
 getAddress(), 189, 190  
 getAdvance(), 541  
 getAllByName(), 189, 190  
 getAllFrames(), 492, 498  
 getAllowsChildren(), 402  
 getAllowUserInteraction(), 214  
 getAnnotation(), 835, 836  
 getAnnotations(), 836  
 GetArrayLength(), 919, 922  
 getAscent(), 541  
 getAsText(), 690, 692, 694, 696  
 getAttribute(), 117, 131  
 getAttributes(), 110, 117, 610  
 getAutoCommit(), 285  
 getAvailableDataFlavors(), 616, 617  
 getAvailableIDs(), 310  
 getAvailableLocales(), 295, 298, 304, 309, 317  
 getBackground(), 356, 358  
 getBeanDescriptor(), 698  
 getBeanInfo(), 684, 685  
 getBlockSize(), 800  
 GetBooleanArrayElements(), 921  
 getBundle(), 324, 325, 326, 328  
 getByName(), 189, 190  
 GetByteArrayElements(), 936  
 getCalendar(), 310  
 getCategory(), 606, 609  
 getCellEditorValue(), 389, 392, 394  
 getCellSelectionEnabled(), 380  
 getCertificates(), 747  
 getChild(), 116, 422, 427, 428  
 getChildAt(), 409  
 getChildCount(), 404, 409, 428  
 getChildNodes(), 117  
 getClassLoader(), 729, 737  
 className(), 871  
 getClip(), 541  
 getCodeSource(), 747  
 getCollationKey(), 313, 318  
 getColorModel(), 566, 570  
 getColumn(), 380

getColumnClass(), 367  
getColumnCount(), 283, 363, 364, 366  
getColumnDisplaySize(), 283  
getColumnLabel(), 283  
getColumnName(), 283, 367  
getColumnNumber(), 126  
getColumnSelectionAllowed(), 379  
getCommand(), 273  
getComponentAt(), 483  
getConcurrency(), 268  
getConnection(), 239, 242, 249, 288  
getConnectTimeout(), 215  
getContent(), 216  
getContentEncoding(), 210, 215  
getContentLength(), 210, 215  
getContentPane(), 485, 499  
getContents(), 614  
getContentType(), 210, 215  
getContextClassLoader(), 738  
getCountry(), 296  
getCurrencyCode(), 303  
getCurrencyInstance(), 297, 302  
getCustomEditor(), 694, 697  
getData(), 110, 118, 615  
getDataElements(), 567, 570, 571  
getDate(), 210, 215  
getDateInstance(), 309  
getDateTimeInstance(), 304, 309  
getDeclaredAnnotations(), 836  
getDecomposition(), 317  
getDefault(), 295, 296, 310  
getDefaultEditor(), 393  
getDefaultFractionsDigits(), 303  
getDefaultName(), 775  
getDefaultRenderer(), 393  
getDefaultToolkit(), 612  
getDescent(), 541  
getDesktopPane(), 499  
getDisplayCountry(), 296  
getDisplayLanguage(), 296  
getDisplayName(), 295, 296, 298, 311, 685  
getDocument(), 459  
getDocumentElement(), 108, 117  
getDoInput(), 214  
getDoOutput(), 214  
getDouble(), 243  
GetDoubleField(), 906  
getElementAt(), 350, 354  
getErrorCode(), 248  
getErrorStream(), 221, 222  
getEventType(), 459  
getExceptionListener(), 724, 725  
getExpiration(), 210, 215  
getFieldDescription(), 420  
GetFieldID(), 907, 910  
getFields(), 427  
getFileSuffixes(), 564  
getFirstChild(), 117  
getFontRenderContext(), 540  
getFontRendererContext(), 541  
getForeground(), 356, 358  
getFormatNames(), 564  
getFrameIcon(), 499  
getHeaderField(), 207, 209, 215  
getHeaderFieldKey(), 207, 209, 215  
getHeaderFields(), 207, 210, 215  
getHeight(), 564, 582, 583, 588  
getHostAddress(), 190  
getHostName(), 190  
getHumanPresentableName(), 617  
getIcon(), 685  
getIconAt(), 483  
getID(), 311  
getIfModifiedSince(), 214  
getImageableHeight(), 583, 588  
getImageableWidth(), 583, 588  
getImageableX(), 589  
getImageableY(), 589  
getImageReadersByFormatName(), 562  
getImageReadersByMIMEType(), 556, 562  
getImageReadersBySuffix(), 556, 562  
getImageWritersByFormatName(), 562  
getImageWritersByMIMEType(), 562  
getImageWritersBySuffix(), 562  
getIndex(), 681  
getIndexedReadMethod(), 687  
getIndexedWriteMethod(), 687  
getIndexOfChild(), 422, 428  
getInputStream(), 187, 191, 208, 219, 221  
getInstance(), 303, 312, 317, 544, 549, 778, 795, 800, 801  
GetIntField(), 906, 936  
getJavaInitializationString(), 697  
getJDBCMinorVersion(), 282  
getJDBCMinorVersion(), 282  
getKeys(), 328  
getLanguage(), 296  
getLastChild(), 110, 117  
getLastModified(), 210, 216  
getLastPathComponent(), 404, 409  
getLastSelectedPathComponent(), 404, 409  
getLayoutOrientation(), 348  
getLayoutPolicy(), 483  
getLeading(), 541  
getLength(), 108, 118, 155  
getLineNumber(), 126  
getListCellRendererComponent(), 358, 359  
getLocale(), 319

getLocalHost(), 189, 190  
 getLocalName(), 150, 155  
 getLocation(), 747  
 getLogger(), 852  
 getMaxConnections(), 282  
 getMaximum(), 473  
 getMaximumFractionDigits(), 302  
 getMaximumIntegerDigits(), 302  
 getMaxStatement(), 245  
 getMaxStatements(), 282  
 getMetaData(), 282, 283  
 GetMethodID(), 917, 919  
 getMimeType(), 616  
 getMIMETypes(), 564  
 getMinimum(), 473  
 getMinimumFractionDigits(), 302  
 getMinimumIntegerDigits(), 302  
 getModel(), 354, 355  
 getName(), 601, 609, 685, 758, 761, 767, 768, 775, 871  
 getNameSpaceURI(), 150  
 getNewValue(), 500, 681  
 getNextException(), 247  
 getNextSibling(), 110, 117  
 getNodeName(), 110, 117, 150  
 getNodeValue(), 110, 117  
 getNumberFormat(), 310  
 getNumberInstance(), 297  
 getNumImages(), 563  
 getNumThumbnails(), 560, 563  
 getObject(), 328  
 GetObjectArrayElement(), 919, 922  
 GetObjectClass(), 907, 910  
 GetObjectField(), 906  
 getOldValue(), 681  
 getOrientation(), 589  
 getOriginatingProvider(), 556, 564, 565  
 getOutline(), 540  
 getOutputSize(), 800  
 getOutputStream(), 187, 191, 208, 218  
 getPageCount(), 590  
 getParent(), 409, 737  
 getParentNode(), 117  
 getPassword(), 273, 775  
 getPath(), 421  
 getPaths(), 421  
 getPathToRoot(), 405  
 getPercentInstance(), 297  
 getPersistenceDelegate(), 724  
 getPixel(), 566, 570, 571  
 getPixels(), 570  
 getPointCount(), 521  
 getPreferredSize(), 356, 358  
 getPreviousSibling(), 117  
 getPrincipals(), 767  
 getPrintable(), 599  
 getPrinterJob(), 581, 588  
 getPrintService(), 604  
 getPrompt(), 775  
 getProperty(), 916  
 getPropertyChangeEvent(), 683  
 getPropertyChangeListeners(), 680  
 getPropertyDescriptors(), 683, 685, 687, 688, 690  
 propertyName(), 500, 671, 681  
 getPropertyType(), 686  
 getProtectionDomain(), 747  
 getPrototypeCell(), 353  
 getQName(), 156  
 getRaster(), 565, 570  
 getReaderFormatNames(), 563  
 getReaderMIMETypes(), 563  
 getReadMethod(), 686  
 getRepresentationClass(), 617  
 getRequestProperties(), 215  
 getResultSet(), 244  
 getRGB(), 567, 571  
 getRoot(), 116, 428  
 getRotateInstance(), 536, 537, 538  
 getRow(), 268  
 getRowCount(), 363, 364, 366  
 getRowHeight(), 379  
 getRowMargin(), 379  
 getRowSelectionAllowed(), 379  
 getSavepointId(), 286  
 getSavepointName(), 286  
 getScaleInstance(), 536, 538  
 getSelectedColumns(), 374  
 getSelectedComponent(), 482  
 getSelectedIndex(), 480, 482  
 getSelectedNode(), 404  
 getSelectedValue(), 349  
 getSelectedValues(), 346, 349  
 getSelectionBackground(), 358  
 getSelectionForeground(), 358  
 getSelectionMode(), 349  
 getSelectionModel(), 370, 379  
 getSelectionPath(), 409, 416, 420, 421  
 getSelectionPaths(), 416, 421  
 getShearInstance(), 536, 538  
 getShortDescription(), 686  
 getSize(), 350, 354  
 getSQLState(), 248  
 GetStaticFieldID(), 910  
 GetStaticMethodID(), 916, 918  
 GetStaticXxxField(), 910  
 getStrength(), 317  
 getString(), 243, 274, 328, 473  
 getStringArray(), 328  
 GetStringChars(), 903

GetStringLength(), 903  
GetStringRegion(), 903  
GetStringUTFChars(), 901, 902, 904, 936  
GetStringUTFLength(), 902  
GetStringUTFRegion(), 903  
getSubject(), 767  
GetSuperClass(), 947  
getSymbol(), 303  
getSystem(), 890  
getSystemClassLoader(), 737  
getSystemClipboard(), 612, 614  
getTabCount(), 483  
getTableCellEditorComponent(), 391, 392, 393  
getTableCellRendererComponent(), 391, 393  
getTableName(), 273  
getTables(), 282  
getTagName(), 108, 117  
getTags(), 692, 697  
getTimeInstance(), 304, 309  
getTimeZone(), 310, 311  
getTitleAt(), 482  
getTransferData(), 615  
getTransferDataFlavors(), 617  
getTranslateInstance(), 536, 538  
getTreeCellRendererComponent(), 413, 414  
getType(), 268  
getUpdateCount(), 244  
getURI(), 156  
getURL(), 272, 460, 463  
getUseCaches(), 214  
getUsername(), 272  
getValue(), 156, 473, 606, 696, 934, 935  
getValueAt(), 363, 364, 367, 389  
getVendorName(), 564  
getVersion(), 556, 564  
getVetoableChangeListeners(), 682  
getVisibleRowCount(), 348  
getWidth(), 564, 582, 583, 588  
getWriteMethod(), 686  
getWriterFormatNames(), 559, 563  
getWriterMIMETypes(), 563  
GetXxxArrayElements(), 922  
GetXxxArrayRegion(), 921, 922  
GetXxxField(), 910  
GIF, 555, 600, 664  
gniazda, 187  
  adresy internetowe, 189  
  kanały, 198  
  limity czasu, 187  
  nawiązanie połączenia, 198  
  otwieranie, 187  
  połączenia częściowo zamknięte, 196  
  zamykanie, 193  
gniazdka, 191

Gnu C, 895  
GradientPaint, 532, 533  
graficzny interfejs użytkownika, 610  
grafika, 505  
  antialiasing, 549  
  drukowanie, 580  
  filtrowanie obrazów, 571  
  Java 2D, 506  
  klasy obiektów graficznych, 511  
  operacje na obrazach, 565  
  potok rysowania, 507  
  potokowe tworzenie, 506  
  prostokąt ograniczający, 509  
  przekształcenia układu współrzędnych, 534  
  przezroczystość, 541  
  przycinanie, 539  
  składanie obrazów, 541  
  ślad pędzla, 524  
  współrzędne, 509  
  wypełnianie obszaru, 532  
  zbiór Mandelbrota, 567  
grant, 749  
Graphics, 505, 506, 508, 541  
  getClip(), 541  
  setClip(), 541  
Graphics2D, 506, 508, 509, 531, 533, 534, 539, 541,  
  542, 549, 554  
  clip(), 541  
  draw(), 508  
  fill(), 508  
  getFontRendererContext(), 541  
  rotate(), 539  
  scale(), 539  
  setComposite(), 549  
  setPaint(), 533  
  setRenderingHint(), 554  
  setRenderingHints(), 554  
  setStroke(), 531  
  setTransform(), 539  
  shear(), 539  
  transform(), 539  
  translate(), 539  
GridBagLayout, 129, 130  
GridBagPane, 132  
GSS, 806  
gzip, 215

## H

handle(), 775  
handleGetObject(), 328  
hashCode(), 841, 884  
HashPrintRequestAttributeSet, 581, 605  
hasła, 769, 772  
hasła dostępu, 209

- hasMore(), 869
  - hasMoreElements(), 934, 935, 945
  - hierarchia klas pozwoleń, 745
  - hiperłącza, 458
  - host, 184
  - HTML, 103, 185, 457
  - HTTP, 186, 230
    - nagłówki żądań, 208
    - odpowiedzi, 210
  - URLConnection, 222
    - getErrorStream(), 222
  - HyperlinkEvent, 459, 460, 463
    - getURL(), 463
  - HyperlinkListener, 459, 463
    - hyperlinkUpdate(), 463
  - hyperlinkUpdate(), 459, 463
- I**
- ICC, 567
  - Icon, 355
  - iconTextGap, 667
  - ID, 123
  - identyfikator URI, 606
  - IDREF, 123
  - IIOImage, 565
  - IIOServiceProvider, 564
    - getVendorName(), 564
    - getVersion(), 564
  - ikony, 684
  - IllegalArgumentException, 924
  - IllegalStateException, 560
  - Image, 659
  - image/gif, 215
  - ImageInputStream, 560
  - ImageIO, 533, 555, 562
    - createImageInputStream(), 563
    - createImageOutputStream(), 563
    - getImageReadersByFormatName(), 562
    - getImageReadersByMIMEType(), 562
    - getImageReadersBySuffix(), 562
    - getImageWritersByFormatName(), 562
    - getImageWritersByMIMEType(), 562
    - getImageWritersBySuffix(), 562
    - getReaderFormatNames(), 563
    - getReaderMIMETypes(), 563
    - getWriterFormatNames(), 563
    - getWriterMIMETypes(), 563
    - read(), 562
    - write(), 562
  - ImageOutputStream, 563
  - ImageReader, 563
    - getHeight(), 564
    - getNumImages(), 563
    - getNumThumbnails(), 563
    - getOriginatingProvider(), 564
    - getWidth(), 564
    - read(), 563
    - readThumbnail(), 563
  - ImageReaderWriterSpi, 564
    - getFileSuffixes(), 564
    - getFormatNames(), 564
    - getMIMETypes(), 564
  - ImageSelection, 621
  - ImageViewerBean, 660, 662, 663, 664, 665, 673, 674, 683
  - ImageViewerBeanBeanInfo, 683
  - ImageWriteParam, 561
  - ImageWriter, 561, 564
    - canInsertImage(), 564
    - getOriginatingProvider(), 565
    - setOutput(), 564
    - write(), 564
    - writeInsert(), 564
  - IMAP, 806
  - IMPLIED, 123
  - implies(), 747, 755, 761
  - import, 732
  - inDaylightTime(), 311
  - IndexedPropertyChangeEvent, 681
    - getIndex(), 681
  - IndexedPropertyDescriptor, 686
    - getIndexedReadMethod(), 687
    - getIndexedWriteMethod(), 687
  - indexOfTab(), 483
  - IndexOutOfBoundsException, 560
  - indywidualizacja ziarnka, 697
    - Customizer, 699
      - getBeanDescriptor(), 698
    - implementacja klasy, 699
    - nazwa klasy, 698
    - PropertyChangeEvent, 700
  - InetAddress, 189, 190, 712
    - getAddress(), 189, 190
    - getAllByName(), 189, 190
    - getByName(), 189, 190
    - getHostAddress(), 190
    - getHostName(), 190
    - getLocalHost(), 189, 190
  - InetSocketAddress, 204
    - isUnresolved(), 204
  - init(), 746, 800, 801
  - InitialContext, 288, 870
  - initialize(), 712, 725, 771, 775
  - InputSource, 120, 125
  - InputStream, 191, 204, 205
  - InputStreamReader, 470
  - INSERT, 235, 242
  - INSERT INTO, 249



- insertNodeInto(), 404, 409
- insertRow(), 267, 269
- insertTab(), 478, 482
- inspektora właściwości, 666
- instalacja JDBC, 235
- instantiate(), 711, 725
- Instrumentation, 857
- int, 236, 287
- IntegerSyntax, 606, 607
- interfejs
  - ActionListener, 835
  - adnotacje, 831
  - AnnotatedElement, 835
  - Annotation, 838
  - AttributeSet, 604
  - Attribute, 604
  - BeanInfo, 683, 684
  - BufferedImageOp, 565, 571
  - CachedRowSet, 271
  - Callback, 773
  - ClipboardOwner, 612
  - Comparator, 312
  - ContentHandler, 151, 152
  - Customizer, 699
  - DataSource, 288
  - Doc, 601
  - EntityResolver, 107
  - ErrorHandler, 125
  - HyperlinkListener, 459
  - Instrumentation, 857
  - JNDI, 288
  - JNI, 898
  - ListCellRenderer, 356
  - ListModel, 354
  - MutableTreeNode, 396
  - Node, 108
  - Pageable, 589
  - Paint, 532
  - Printable, 580, 581
  - PrintRequestAttributeSet, 581
  - PrivilegedExceptionAction, 764
  - PrivilegedAction, 763
  - PropertyChangeListener, 676
  - ResultSet, 270
  - RowSet, 270
  - Shape, 520, 523
  - Source, 176
  - Stroke, 524
  - SupportedValuesAttribute, 604
  - TableCellEditor, 391
  - TableCellRenderer, 383
  - Transferable, 611, 615
  - TreeCellRenderer, 412
  - TreeModel, 116, 396, 422
  - TreeSelectionListener, 415
  - VetoableChangeListener, 495
- interfejs programowy wywołań języka Java, 927, 932
- interfejs użytkownika
  - MDI, 483
- internacjonalizacja
  - czas, 304
  - data, 292, 304
  - ISO, 293
  - języki, 293
  - komplety zasobów, 324
  - komunikaty, 318
  - liczby, 297
  - Locale, 293
  - lokalizatory, 292
  - łańcuchy znaków, 325
  - pliki tekstowe, 322
  - pliki źródłowe programów, 322
  - porządek alfabetyczny, 311
  - waluta, 297, 302
  - zbiory znaków, 322
- internalFrameClosing, 497
- InternalFrameListener, 497
- interpolacja, 572
- interrupt(), 198
- InterruptedException, 188
- Introspector, 684, 685
  - getBeanInfo(), 685
- intValue(), 936
- InverseEditor, 688, 694
- inżynieria kodu bajtowego, 851
  - BCEL, 852
  - modyfikacja kodu podczas ładowania, 857
- IPv6, 189
- isAdjusting(), 346
- isAfterLast(), 269
- isAnnotationPresent(), 836
- IsAssignableFrom(), 936, 946
- isBeforeFirst(), 269
- isCanceled(), 474
- isCellEditable(), 367, 384, 385, 391, 394
- isClosable(), 498
- isClosed(), 189, 499
- isConnected(), 189
- isContinuousLayout(), 477
- isDataFlavorAvailable(), 615
- isDataFlavorSupported(), 615
- isEchoOn(), 775
- isExpert(), 686
- isFirst(), 269
- isGroupingUsed(), 302
- isHidden(), 686
- isIcon(), 499
- isIconifiable(), 499



isIgnoringElementContentWhitespace(), 126  
 isIndeterminate(), 473  
 isInputShutdown(), 198  
 isLast(), 269  
 isLeaf(), 401, 402, 426, 428  
 isLenient(), 310  
 isMaximizable(), 498  
 isMaximum(), 499  
 isMimeTypeEqual(), 617  
 isNamespaceAware(), 150, 154  
 ISO, 293  
 isOneTouchExpandable(), 477  
 isOutputShutdown(), 198  
 isPaintable(), 694, 697  
 isParseIntegerOnly(), 302  
 isPropertyName(), 672  
 isResizable(), 498  
 isRunning(), 672  
 isSelected(), 499  
 isStringPainted(), 473  
 isUnresolved(), 204  
 isValidating(), 126, 154  
 isVisible(), 500  
 Item, 858  
 item(), 118  
 Iterator, 243

## J

JAAS, 762, 767  
   moduł logowania, 768  
   moduły, 767  
   uwierzytelnianie oparte na rolach, 768  
 jaas.config, 766  
 JAR, 205, 663, 729  
   podpisywanie plików, 789  
 jarray, 919, 935  
 jarsigner, 783, 784, 790  
 java, 806  
 Java, 11  
 Java 2, 625, 744  
 Java 2 Enterprise Edition, 230  
 Java 2D, 505, 506  
   figury, 507, 508, 509  
   geometria pól, 523  
   klasy obiektów graficznych, 511  
   krzywe, 509, 512  
   krzywe Beziera, 513  
   linie, 513  
   łuk, 511  
   operacje na polach, 524  
   pola, 523  
   prostokąt, 511  
   przezroczystość, 541  
   punkty kontrolne, 521

składanie obrazów, 541  
 ślad pędzla, 524  
 wartość alfa, 542  
 wskazówki operacji graficznych, 549  
 współrzędne, 509  
 wypełnianie obszaru, 532  
 Java Authentication and Authorization Service, 762  
 java.awt.datatransfer, 611  
 java.beans.Beans, 670  
 java.net.Socket, 187  
 java.nio, 196, 198  
 java.policy, 748  
 java.rmi, 866  
 java.security, 728, 776  
 java.security.policy, 880  
 java.text, 297  
 JavaBeans, 630  
   arkusz właściwości, 658  
   Bean Builder, 666  
   CalendarBean, 661  
   domyślny konstruktor, 660  
   edytor właściwości, 687  
   indywidualizacja ziarenka, 697  
   klasa informacyjna ziarenka, 683  
   kontrolki, 658  
   NetBeans, 663  
   plik manifestu, 663  
   pliki JAR, 663  
   projektowanie, 669  
   trwałość ziarenek, 705, 715  
   tworzenie aplikacji, 662, 666  
   tworzenie ziarenek, 660  
   typy właściwości, 673  
   ziarenka, 658  
   ziarenka, 657  
 javax, 894, 906  
 JavaMail, 210  
 javap, 912  
 Javascript, 458  
 JavaServer Faces, 216, 658  
 JavaServer Pages, 658  
 javax.imageio, 555  
 javax.mail.internet.MimeUtility, 210  
 javax.security.auth.login.LoginContext, 766  
 javax.security.auth.Subject, 767  
 javax.sql.rowset, 270  
 JAXP, 107  
 JBuilder, 657  
 JCE, 801  
 JCheckBox, 389  
 jclass, 906, 916  
 JColorChooser, 630  
 JComboBox, 389, 521

- JComponent, 403, 500, 630, 675, 682
  - addPropertyChangeListener(), 682
  - addVetoableChangeListener(), 500, 682
  - firePropertyChange(), 682
  - fireVetoableChange(), 682
  - putClientProperty(), 403
  - removePropertyChangeListener(), 682
  - removeVetoableChangeListener(), 682
  - setTransferHandle(), 630
- JDBC
  - adres URL baz danych, 237
  - aktualizacja danych, 227
  - aktualizacja wsadowa, 284
  - aktualizowalne zbiory wyników zapytań, 263, 265
  - architektura, 228
  - instalacja, 235
  - JNDI, 288
  - klient, 230
  - klient-serwer, 230
  - menedżer sterowników, 239
  - metadane, 274
  - nawiązywanie połączenia, 237
  - polecenia, 245
  - polecenia przygotowane, 252
  - połączenia krótkotrwałe, 246
  - przewijalne zbiory wyników zapytań, 263
  - przewijanie zbioru rekordów, 264
  - pula połączeń, 289
  - rekordy wstawiania, 266
  - serwer, 230
  - SQL, 227, 231
  - sterowniki, 228
  - transakcje, 283
  - warstwa pośrednia, 230
  - wersje, 227
  - wykonywanie zapytań, 252
  - wypełnianie bazy danych, 248
  - zamykanie zbioru wyników, 245
  - zapytania, 227
  - zarządzanie połączeniami, 245
  - zastosowania, 229
  - zbiory rekordów, 263, 270
  - zbiory wyników, 245
  - źródło danych, 237
- JDBC 3, 288
- JDBC API, 228
- jdbc.password, 249
- jdbc.property, 239
- jdbc.url, 249
- jdbc.username, 249
- JdbcRowSet, 270
- JDesktopPane, 485, 486, 487, 492, 498
  - getAllFrames(), 498
  - setDragMode(), 498
- JDialog, 496
- JEditorPane, 457, 458, 459, 460
  - addHyperlinkListener(), 462
  - setPage(), 462
- język, 293
  - C, 892
  - C++, 895
  - DDL, 244
  - HTML, 103, 185
  - IDL, 863
  - Java, 11
  - Javascript, 458
  - SGML, 103
  - SQL, 227, 231
  - Visual Basic, 658
  - WSDL, 863
  - XML, 101, 103, 864
  - XML Schema, 119, 126
  - XPath, 142
- jfieldID, 906
- JFileChooser, 630
- JFrame, 485, 706
- JInternalFrame, 485, 487, 495, 498, 676
  - getContentPane(), 499
  - getDesktopPane(), 499
  - getFrameIcon(), 499
  - isClosable(), 498
  - isClosed(), 499
  - isIcon(), 499
  - isIconifiable(), 499
  - isMaximizable(), 498
  - isMaximum(), 499
  - isResizable(), 498
  - isSelected(), 499
  - isVisible(), 500
  - moveToBack(), 499
  - moveToFront(), 499
  - reshape(), 499
  - setClosable(), 498
  - setClosed(), 499
  - setContentPane(), 499
  - setFrameIcon(), 499
  - setIcon(), 499
  - setIconifiable(), 499
  - setMaximizable(), 499
  - setMaximum(), 499
  - setResizable(), 498
  - setSelected(), 499
  - setVisible(), 500
  - show(), 500
- JLabel, 358, 412
- JList, 344, 348, 353, 355, 358, 630
  - addListSelectionListener(), 349
  - getBackground(), 358
  - getForeground(), 358

- getLayoutOrientation(), 348
- getModel(), 355
- getPrototypeCell(), 353
- getSelectedValue(), 349
- getSelectedValues(), 346, 349
- getSelectionBackground(), 356, 358
- getSelectionForeground(), 356, 358
- getSelectionMode(), 349
- getVisibleRowCount(), 348
- HORIZONTAL\_WRAP, 345
- isAdjusting(), 346
- konstruktory, 355
- setCellRenderer(), 357, 359
- setFixedCellHeight(), 353
- setFixedCellWidth(), 353
- setLayoutOrientation(), 348
- setPrototypeCell(), 353
- setSelectionMode(), 345
- setVisibleRowCount(), 345, 348
- valueChanged(), 346
- VERTICAL, 345
- VERTICAL\_WRAP, 345
- wektor obiektów, 354
- JNDI, 288
  - nawiązywanie połączenia, 289
  - pula połączeń, 289
  - zarządzanie nazwami użytkowników, 289
  - źródła danych, 288
- JndiLoginModule, 763
- JNI, 898, 900
  - konwencja wywołań, 901
  - wywołania funkcji, 901
- JNI\_CreateJavaVM(), 928, 932
- JNI\_TRUE, 936
- JNICALL, 904
- JNIEXPORT, 904
- JobAttributes, 609
- JobHoldUntil, 607
- JobImpressions, 607
- JobImpressionsCompleted, 607
- jobobject, 906, 935
- JobKOctets, 607
- JobKOctetsProcessed, 607
- JobMediaSheets, 607
- JobMediaSheetsCompleted, 607
- JobMessageFromOperator, 607
- JobName, 607
- JobOriginatingUserName, 608
- JobPriority, 608
- JobSheets, 608
- JobState, 608
- JobStateReason, 608
- JobStateReasons, 608
- JoinRowSet, 270
- JOptionPane, 496
- JPanel, 584
- JPEG, 555, 556
- JProgressBar, 463, 472
  - getMaximum(), 473
  - getMinimum(), 473
  - getString(), 473
  - getValue(), 473
  - isIndeterminate(), 473
  - isStringPainted(), 473
  - setIndeterminate(), 473
  - setMaximum(), 473
  - setMinimum(), 473
  - setString(), 473
  - setStringPainted(), 473
  - setValue(), 473
- JScrollPane, 344, 362
- JSF, 658
- JSP, 658
- JSplitPane, 475, 477, 497
  - HORIZONTAL\_SPLIT, 475
  - isContinuousLayout(), 477
  - isOneTouchExpandable(), 477
  - setBottomComponent(), 477
  - setContinuousLayout(), 477
  - setLeftComponent(), 477
  - setOneTouchExpandable(), 477
  - setRightComponent(), 477
  - setTopComponent(), 477
  - VERTICAL\_SPLIT, 475
- jstring, 900, 916, 935
- JTabbedPane, 478, 482
  - addChangeListener(), 483
  - addTab(), 482
  - getComponentAt(), 483
  - getIconAt(), 483
  - getLayoutPolicy(), 483
  - getSelectedComponent(), 482
  - getSelectedIndex(), 482
  - getTabCount(), 483
  - getTitleAt(), 482
  - indexOfTab(), 483
  - insertTab(), 482
  - removeTabAt(), 482
  - setComponentAt(), 483
  - setIconAt(), 483
  - setSelectedIndex(), 482
  - setTabLayoutPolicy(), 483
  - setTitleAt(), 482
- JTable, 359, 363, 368, 391, 393, 630
  - addColumn(), 380
  - getCellSelectionEnabled(), 380
  - getColumnSelectionAllowed(), 379
  - getDefaultEditor(), 393

JTable  
 getDefaultRenderer(), 393  
 getRowHeight(), 379  
 getRowMargin(), 379  
 getRowSelectionAllowed(), 379  
 getSelectionModel(), 379  
 moveColumn(), 380  
 removeColumn(), 380  
 setAutoResizeMode(), 379  
 setColumnSelectionAllowed(), 380  
 setRowHeight(), 379  
 setRowMargin(), 379  
 setRowSelectionAllowed(), 379  
 JTextArea, 457, 756  
 JTextComponent, 459, 630  
 JTextField, 389, 457  
 JTree, 111, 394, 395, 397, 403, 409, 421, 630  
 getLastSelectedPathComponent(), 409  
 getSelectionPath(), 409, 421  
 getSelectionPaths(), 421  
 makeVisible(), 409  
 scrollPathToVisible(), 409  
 setRootVisible(), 402  
 setShowsRootHandles(), 402  
 jvalue, 919  
 jvm, 928

**K**

kalendarz, 662  
 kalkulator emerytalny, 328  
 kanały, 198  
 SocketChannel, 198  
 Kerberos, 806  
 Kernel, 578, 580  
 KEY\_ALPHA\_INTERPOLATION, 550  
 KEY\_ANTIALIASING, 550  
 KEY\_COLOR\_RENDERING, 550  
 KEY\_DITHERING, 550  
 KEY\_FRACTIONAL\_METRICS, 550  
 KEY\_INTERPOLATION, 550  
 KEY\_RENDERING, 550  
 KEY\_STROKE\_CONTROL, 550  
 KEY\_TEXT\_ANTIALIASING, 550  
 KeyGenerator, 797, 801  
 generateKey(), 801  
 getInstance(), 801  
 init(), 801  
 KeyPairGenerator, 803  
 KeyStoreLoginModule, 763  
 keytool, 781, 782, 787, 790  
 klasy, 732  
 AbstractCellEditor, 391  
 AbstractTableModel, 363  
 abstrakcyjne, 412

ActionListenerInstaller, 835  
 Activatable, 885, 886  
 AffineTransform, 536  
 AffineTransformOp, 572  
 AllPermissions, 746, 755  
 AlphaComposite, 544  
 Arc2D, 511  
 ArcMaker, 521  
 Area, 523  
 Banner, 590  
 Base64, 210  
 Base64Encoder, 214  
 BasicPermission, 752  
 BasicStroke, 524, 525  
 Book, 598  
 BufferedImage, 533, 565  
 ButtonFrame, 835  
 ChartBeanBeanInfo, 688, 845  
 ChoiceFormat, 321  
 Cipher, 795  
 CipherInputStream, 802  
 ClassLoader, 729, 733  
 ClassNameTreeCellRenderer, 413  
 ClassTreeFrame, 414  
 Clipboard, 611  
 Collator, 312  
 Color, 532  
 ColorConvertOp, 577  
 ColorModel, 568  
 ConvolveOp, 578  
 Copies, 604, 606  
 CopiesSupported, 604  
 CubicCurve2D, 513  
 Currency, 302  
 DatabaseMetaData, 265, 275  
 DataFlavor, 611, 615  
 DateFormat, 304, 305  
 DefaultCellEditor, 406  
 DefaultHandler, 152  
 DefaultListModel, 354  
 DefaultModelList, 354  
 DefaultMutableTreeNode, 396  
 DefaultPersistenceDelegate, 713  
 DefaultTreeModel, 396, 405, 422  
 DialogCallbackHandler, 772  
 DocFlavor, 599  
 DocPrintJob, 601  
 Document, 108  
 DocumentBuilder, 107, 160  
 domena ochronna, 746  
 DOMResult, 177  
 DOMSource, 176  
 DOMTreeModel, 116  
 DriverManager, 288

Ellipse2D, 511  
 EntryLogger, 858  
 EnumSyntax, 607  
 File, 397  
 FilePermission, 745  
 GeneralPath, 513  
 GradientPaint, 532  
 Graphics, 505, 508  
 Graphics2D, 506  
 GridBagLayout, 129  
 HashPrintRequestAttributeSet, 581  
 HyperlinkEvent, 460  
 Icon, 355  
 ImageInputStream, 560  
 ImageIO, 555  
 ImageViewerBean, 660  
 ImageWriter, 561  
 InetAddress, 189  
 informacyjne ziarnka, 683, 830  
 InputStream, 205  
 IntegerSyntax, 606  
 InverseEditor, 694  
 JComboBox, 521  
 JDesktopPane, 485, 487  
 JEditorPane, 457  
 JFrame, 485  
 JInternalFrame, 485, 487  
 JLabel, 358, 412  
 JList, 349  
 JPanel, 584  
 JProgressBar, 463  
 JSplitPane, 497  
 JTabbedPane, 478  
 JTable, 368  
 JTextArea, 756  
 JTree, 394  
 Kernel, 578  
 KeyGenerator, 797  
 KeyPairGenerator, 803  
 komplety zasobów, 326  
 Line2D, 511  
 ListResourceBundle, 326  
 Locale, 293, 295, 297  
 LoginContext, 762  
 LookupOp, 576, 577  
 ładowanie, 728  
 MessageDigest, 778  
 MessageFormat, 318, 321  
 MimeUtility, 210  
 Number, 297  
 NumberFormat, 297, 298, 304  
 PageFormat, 582  
 Permission, 755  
 PersistenceDelegate, 712  
 Point2D, 509  
 Policy, 745  
 pozwolenia, 755  
 PreparedStatement, 254  
 PrinterJob, 581  
 PrintPreviewDialog, 591, 598  
 PrintService, 601, 603  
 PrintServiceLookup, 600  
 PrintWriter, 218  
 PrivilegedAction, 763  
 ProgressMonitor, 463, 466  
 ProgressMonitorInputStream, 463, 469  
 Properties, 102  
 PropertyChangeSupport, 675  
 PropertyDescriptor, 687  
 PropertyEditorSupport, 691  
 Random, 797  
 Raster, 567  
 Reader, 176  
 Rectangle2D, 511  
 RenderingHints, 551  
 RescaleOp, 576  
 ResourceBundles, 326  
 ResultSetMetaData, 275  
 RetinaScanCallback, 773  
 RoundRectangle2D, 511  
 rozwiązywanie, 728  
 Runtime, 743  
 SAXSource, 177  
 Scanner, 198  
 SecureRandom, 797  
 SecurityManager, 746  
 ServerSocket, 191, 193  
 ShapeMaker, 521  
 ShapePanel, 521  
 SimpleBeanInfo, 684  
 SimpleDateFormat, 319  
 SimpleLoginModule, 771  
 SimplePrincipal, 769  
 SimulatedActivity, 466  
 Socket, 187  
 SocketChannel, 198  
 StreamPrintService, 603  
 StreamPrintServiceFactory, 603  
 StreamSource, 176  
 String, 312  
 StringBuffer, 472  
 StringSelection, 612, 617  
 szyfrowanie plików, 737  
 TableColumn, 368  
 TableColumnModel, 368  
 TextLayout, 540  
 TexturePaint, 532, 533  
 ThreadedEchoHandler, 194

- klasy
    - TreeNode, 403
    - TreePath, 404
    - TreeSelectionModel, 415
    - UnicastRemoteObject, 867
    - UnixNumericGroupPrincipal, 763
    - UnixPrincipal, 762
    - URI, 205
    - URL, 205
    - URLConnection, 205
    - URLConnection, 205
    - VetoableChangeSupport, 677
    - WordCheckPermission, 756
    - WritableRaster, 567
    - XPath, 143
  - klient, 185, 230, 862
  - klucz prywatny, 780
  - klucz publiczny, 780
  - klucze, 796
  - kod ASCII, 291
  - kod bajtowy, 851
  - kod języków, 293
  - kod macierzysty, 891
  - kod Unicode, 291
  - kodowanie znaków, 322
  - kolory, 542
    - RGB, 541, 566
  - kolumny, 231
  - kompilator, 728
    - Gnu C, 895
  - komplety zasobów, 324
    - implementacja klas, 326
    - klasy, 326
    - ładowanie, 324
    - pliki właściwości, 325
  - komponenty
    - formatowanie tekstu, 429
    - JavaBeans, 658
    - JEditorPane, 457
    - JList, 344
    - JProgressBar, 463
    - JTextArea, 457
    - JTextField, 457
    - JTree, 394
    - Metal, 484
    - organizatory, 474
    - panele dzielone, 475
    - rozmieszczenie, 487
  - komunikacja, 862
  - komunikaty, 318
    - formatowanie z wariantami, 320
    - indeks znacznika, 319
  - konfiguracja wywołania zdalnych metod, 866
  - konstruktory, 917
    - kontekst graficzny, 535, 583, 584
    - kontekst tworzenia czcionki, 540
    - kontrola dostępu, 727
    - kontrola poprawności dokumentów XML, 118
      - atributy, 122
      - ATTLIST, 122
      - byty, 123
      - CDATA, 123
      - definicja typu dokumentu, 119
      - DOCTYPE, 120
      - DTD, 119
      - ELEMENT, 119, 121
      - parser XML, 122
      - reguły zawartości elementów, 121
      - skróty, 123
      - typy atrybutów, 122
      - wartości domyślne atrybutów, 123
      - warunki kontroli, 119
      - XML Schema, 119, 126
    - kontrola pozwoleń, 747
    - kontrolki, 658
      - CommonDialog, 658
      - Image, 658
    - kończenie pracy maszyny wirtualnej, 744
    - korzeń, 395
    - Krb5LoginModule, 763
    - kryptografia klucza publicznego, 780, 803
    - krzywe, 509, 512, 521
      - Beziera, 513
      - drugiego stopnia, 513
      - punkty kontrolne, 512
      - trzeciego stopnia, 513
    - kursory, 264
- L**
- las, 395, 400
  - last(), 268
  - layoutPages(), 590
  - LD\_LIBRARY\_PATH, 932
  - LDAP, 806
  - liczby, 297
    - formatowanie, 297, 899
    - formaty, 298
    - lokalizatory, 297
    - losowe, 797
  - LIKE, 234
  - Line2D, 508, 509, 511
  - lineTo(), 513, 514, 522
  - linie, 513
  - Linux, 234, 931
  - list(), 871
  - ListCellRenderer, 356, 359
    - getListCellRendererComponent(), 359

ListModel, 354  
 getElementAt(), 354  
 getSize(), 354  
 ListResourceBundle, 326  
 ListSelectionListener, 349  
 ListSelectionModel, 381  
 setSelectionMode(), 381  
 listy, 343  
 JList, 344  
 kolor tła komórki, 358  
 modele, 349  
 obiekt odrysowujący zawartość komórek, 355  
 odrysowywanie zawartości, 355  
 powiadomienia, 346  
 prezentacja elementów, 345  
 przewijanie zawartości, 344  
 rozwijalne, 344  
 tworzenie, 344  
 usuwanie elementów, 354  
 wizualizacja danych, 350  
 wstawianie elementów, 354  
 zaznaczanie elementów, 345  
 liście, 395  
 loadClass(), 733  
 loadImage(), 668, 684, 685  
 loadLibrary(), 896, 897  
 Locale, 293, 294, 295, 296, 297  
 getCountry(), 296  
 getDefault(), 296  
 getDisplayCountry(), 296  
 getDisplayLanguage(), 296  
 getDisplayName(), 296  
 getLanguage(), 296  
 setDefault(), 296  
 toString(), 296  
 localFile, 750  
 localhost, 868  
 LoggingPermission, 752  
 logika biznesowa, 764  
 login(), 766, 776  
 LoginContext, 762, 766, 771  
 getSubject(), 767  
 login(), 766  
 logout(), 766  
 LoginModule, 775  
 abort(), 776  
 commit(), 776  
 initialize(), 775  
 login(), 776  
 logout(), 776  
 logout(), 766, 776  
 logowanie, 763, 774  
 lokalizacja, 293, 328  
 kod, 744  
 zasoby, 324

lokalizatory, 292  
 czas, 304  
 data, 304  
 domyślny, 295  
 języki, 294  
 liczby, 297  
 lookup(), 288, 870, 871  
 LookupOp, 576, 577, 579  
 lookupPrintServices(), 600, 601  
 LookupTable, 576  
 lostOwnership(), 615

## L

ładowanie klas, 728  
 ClassLoader, 733  
 implementacja procedury ładującej, 733  
 klasy systemowe, 729  
 maszyna wirtualna, 728  
 procedura rozszerzona, 729  
 procedura systemowa, 729  
 procedury, 730  
 przestrzeń nazw, 732  
 rozszerzenia maszyny wirtualnej, 729  
 URLClassLoader, 729  
 łańcuch zaufania, 784  
 łańcuchy, 321  
 łączenie tabel, 232  
 łuk, 511, 521

## M

macierz przekształceń, 536  
 mailto, 205  
 main(), 728  
 makeShape(), 521  
 makeVisible(), 405, 409  
 mapy bitowe, 684  
 MarshalledObject, 885, 889  
 get(), 890  
 MD5, 777  
 MDI, 483, 484  
 mechanizm przeciagnij i upuść, 625  
 MediaName, 608  
 MediaSize, 608  
 MediaSizeName, 608  
 MediaTray, 608  
 menedżer bezpieczeństwa, 728, 742, 749, 879  
 checkExit(), 743  
 domyślny, 744  
 pliki polityki, 749  
 pozwolenia, 742  
 SecurityManager, 746  
 MessageDigest, 778, 779  
 digest(), 779  
 reset(), 779  
 update(), 779



- MessageFormat, 318, 319, 321
    - applyPattern(), 319
    - format(), 318, 320
    - getLocale(), 319
    - setLocale(), 319
  - Messenger, 850
  - metaadnotacje, 843
  - metadane, 274
    - DatabaseMetaData, 275
    - pozyskiwanie, 274
    - ResultSetMetaData, 275
  - Metal, 398
  - Method, 835, 915
  - MethodDescriptor, 684
  - metody
    - main(), 728
    - sygnatury, 911
  - metody macierzyste
    - alternatywne sposoby wywoływania metod, 917
    - dostęp do pól instancji, 906, 910
    - dostęp do pól statycznych, 910
    - funkcje języka C, 892
    - implementacja, 895
    - interfejs programowy wywołań języka Java, 927
    - jarray, 919
    - język C++, 895
    - JNI, 898
    - konstruktory, 917
    - łańcuchy znaków, 900
    - obsługa błędów, 923, 927
    - parametry numeryczne, 898
    - przeciążanie identyfikatorów, 893
    - rejestr systemu Windows, 932
    - składowe obiektu, 906
    - sygnatury, 911
    - tablice, 919, 922
    - wartości zwracane, 898
    - wyrzucanie wyjątków, 923
    - wywoływanie metod języka Java, 912, 917
    - wywoływanie metod obiektów, 912
    - wywoływanie metod statycznych, 916
  - Microsoft ASP, 216
  - MIME, 555, 556, 616
  - MimeUtility, 210
  - MissingResourceException, 325
  - model drzewa, 396
  - model kolorów, 566
  - model tabeli, 359, 363
  - model-widok-nadzorca, 349
  - moduł JAAS, 767
  - moduł logowania, 763, 768
    - hasła, 769, 772
    - logowanie, 774
    - nazwa użytkownika, 772
    - options, 771
    - SimpleLoginModule, 769
  - moduł uwierzytelniania, 763
  - modyfikacja kodu bajtowego podczas ładowania, 857
  - modyfikatory, 831
  - monitorowanie postępu strumieni wejścia, 469
  - monitory postępu, 466
  - mostek JDBC/ODBC, 228
  - moveColumn(), 380
  - moveTo(), 514, 522
  - moveToBack(), 499
  - moveToCurrentRow(), 267, 269
  - moveToFront(), 499
  - moveToInsertRow(), 267, 269
  - MultipleDocumentHandling, 608
  - MutableTreeNode, 396, 402
    - setUserObject(), 402
- ## N
- nagłówki żądań HTTP, 208
  - NameCallback, 773, 775
    - getDefaultName(), 775
    - getName(), 775
    - getPrompt(), 775
    - setName(), 775
  - NameClassPair, 869, 871
    - getClassName(), 871
    - getName(), 871
  - NamedNodeMap, 110, 118
    - getLength(), 118
    - item(), 118
  - namiastka, 863, 864
    - klienta, 862
    - serwera, 862
    - szeregowanie parametrów, 865
  - Naming, 871
    - bind(), 871
    - list(), 871
    - lookup(), 871
    - rebind(), 871
    - unbind(), 871
  - native2ascii, 323
  - negatyw obrazu, 576
  - NetBeans, 657, 663, 665
    - edytor właściwości, 667, 687
    - implementacja klasy indywidualizacji, 699
    - inspektor właściwości, 669
    - tworzenie formatki, 666
    - tworzenie projektu, 665
    - właściwości, 667
    - zdarzenia, 668
  - NetPermission, 751
  - NewByteArray(), 935
  - newDocument(), 160



newDocumentBuilder(), 116  
 newInputStream(), 204  
 newInstance(), 116, 147, 154, 164  
 NewObject(), 919, 923  
 NewObjectA(), 919  
 NewObjectV(), 919  
 newOutputStream(), 198, 204  
 newSAXParser(), 154  
 NewString(), 903  
 NewStringUTF(), 902, 935  
 newPath(), 147  
 NewXxxArray(), 921  
 next(), 244  
 nextElement(), 934, 935, 945  
 NMTOKEN, 123  
 NMTOKENS, 123  
 Node, 108, 117, 150  
   appendChild(), 164  
   getAttributes(), 117  
   getChildNodes(), 117  
   getFirstChild(), 117  
   getLastChild(), 117  
   getLocalName(), 150  
   getNameSpaceURI(), 150  
   getNextSibling(), 117  
   getNodeName(), 117  
   getNodeValue(), 117  
   getParentNode(), 117  
   getPreviousSibling(), 117  
 nodeChanged(), 405, 410  
 NodeList, 108, 118  
   getLength(), 118  
   item(), 118  
 NOT LIKE, 234  
 NTLoginModule, 763  
 NullPointerException, 924  
 Number, 297  
 NumberFormat, 297, 298, 301, 304  
   format(), 301  
   getAvailableLocales(), 298  
   getCurrencyInstance(), 302  
   getMaximumFractionDigits(), 302  
   getMaximumIntegerDigits(), 302  
   getMinimumFractionDigits(), 302  
   getMinimumIntegerDigits(), 302  
   isGroupingUsed(), 302  
   isParseIntegerOnly(), 302  
   parse(), 301  
   setGroupingUsed(), 302  
   setMaximumFractionDigits(), 302  
   setMaximumIntegerDigits(), 302  
   setMinimumFractionDigits(), 302  
   setMinimumIntegerDigits(), 302  
   setParseIntegerOnly(), 302

NumberOfDocuments, 608  
 NumberOfInterveningJobs, 608  
 NumberUp, 608  
 NUMERIC, 236, 287  
 numeryczne parametry metod, 898

## 0

obiekty  
   dostęp do składowych, 906  
   nasłuchujące, 674  
   odrysowujące zawartość komórek listy, 355  
   serializacja, 706  
   użytkownika, 396  
 obrazy  
   dostęp do danych, 565  
   filtrowanie, 571  
   model kolorów, 566  
   modyfikacja pikseli, 565  
   negatyw, 576  
   obrót, 572  
   operacje, 565  
   próbki piksela, 566  
   przejrzystość, 541  
   przyrostowe tworzenie, 565  
   rozmycie, 577  
   sekwencje, 560  
   wczytywanie, 555  
   wykrywanie krawędzi, 578  
   zapisywanie, 555  
   zbiór Mandelbrota, 567  
 obrót, 534, 572  
 obrys czcionek, 540  
 obrys figury, 506  
 obrys tekstu, 540  
 obsługa błędów, 923  
 obsługa zdarzeń  
   adnotacje, 832  
 ODBC, 228  
 odcinki, 513  
 odcisk palca, 777  
 odszyfrowywanie danych, 801  
 odwołanie transakcji, 283  
 okna dialogowe, 478  
   drukowanie, 587  
   ramki wewnętrzne, 496  
   wybór plików, 659  
 openConnection(), 207, 214  
 openInputStream(), 216  
 openOutputStream(), 216  
 openStream(), 205, 208, 214  
 operacje na obrazach, 565  
 ORDER BY, 243  
 organizatory komponentów, 474  
 OrientationRequested, 608

OutOfMemoryError, 924  
OutputDeviceAssigned, 608  
OutputStream, 191, 204

## P

Package, 835  
Pageable, 589  
PageAttributes, 609  
pageDialog(), 583, 588  
PageFormat, 582, 587, 588  
    getHeight(), 588  
    getImageableHeight(), 588  
    getImageableWidth(), 588  
    getImageableX(), 589  
    getImageableY(), 589  
    getOrientation(), 589  
    getWidth(), 588  
PageRanges, 608  
PagesPerMinute, 608  
PagesPerMinuteColor, 608  
Paint, 532  
paint(), 506  
paintComponent(), 358, 506, 530, 584, 598  
paintValue(), 694, 697  
pakiety, 732  
panele dzielone, 475  
    JSplitPane, 475  
panele pulpitu, 483  
panele z zakładkami, 478  
    JTabbedPane, 478  
parse(), 116, 155, 181, 297, 301, 310  
ParseException, 298  
parser  
    SAX, 150  
parser XML, 107  
    DOM, 107  
    implementacja, 122  
    SAX, 107  
parsowanie dokumentów XML, 107, 112  
    DOM, 107, 109  
    DOMTreeModel, 116  
    SAX, 107  
    TreeModel, 116  
    XML Schema, 129  
pasek postępu, 463  
    nieokreślony, 466  
    usawianie wartości, 463  
pasek przewijania, 362  
PasswordCallback, 773, 775  
    getPassword(), 775  
    getPrompt(), 775  
    isEchoOn(), 775  
    setPassword(), 775  
pathFromAncestorEnumeration(), 411  
PCDATA, 121, 124, 131  
PDLOverrideSupported, 608  
Permission, 755, 761  
    getName(), 761  
    implies(), 761  
PersistenceDelegate, 712, 725  
    initialize(), 725  
piaskownica, 744  
piksel, 541  
    docelowy, 542  
PJA, 607  
PKCS#5, 796  
pliki, 753  
    class, 728  
    JAR, 205, 663, 664, 729  
    java.policy, 748  
    pomocnicze, 830  
    pozwoleń, 745  
    rt.jar, 730  
    tekstowe, 322  
    właściwości, 102, 325  
    XML, 102, 707  
    XML Schema, 119  
    zasoby, 324  
    ZIP, 729  
    źródłowe, 322  
pliki graficzne, 555  
    animacje GIF, 562  
    formaty, 555  
    GIF, 555  
    ImageIO, 555  
    interfejs dostawcy, 556  
    JPEG, 555, 556  
    MIME, 555, 556  
    obiekt odczytu, 556  
    obiekty, 555  
    sekwencje obrazów, 560  
    wczytywanie, 555  
    zapisywanie, 555, 559  
pliki polityki, 744, 747, 880  
    baza kodu, 750  
    edycja, 754  
    grant, 749  
    implementacja klasy pozwoleń, 756  
    java.policy, 748  
    klasy pozwoleń, 755  
    menedżer bezpieczeństwa, 749  
    niezależne od platformy systemowej, 754  
    odczyt, 754  
    operacje sieciowe, 753  
    położenie, 748  
    pozwolenia, 750, 758  
    składnica kluczy, 790  
    system plików, 752

- testowanie aplikacji, 749
- właściwości systemowe, 753
- zapis, 754
- PNG, 555
- pochylenie, 535
- poczta elektroniczna, 183, 222
  - SMTP, 222
  - wysyłanie, 222
- podgląd wydruku, 591
- podpis cyfrowy, 776, 785
  - aplety, 776
  - generator liczb losowych, 797
  - MD5, 777
  - MessageDigest, 778
  - podpisywanie wiadomości, 779
  - SHA1, 777
  - skrót wiadomości, 777
  - weryfikacja, 781
- podpisywanie certyfikatów, 786
- podpisywanie kodu, 728, 788
  - aplety dostępne przez Internet, 789
  - aplety intranetowe, 789
  - certyfikaty twórców oprogramowania, 793
  - jar signer, 790
  - pliki JAR, 789
- podpisywanie wiadomości, 779
- Point2D, 509, 521, 540
- pola, 523
  - add, 523
  - exclusiveOr, 523
  - intersect, 523
  - operacje, 524
  - subtract, 523
  - śląd pędzla, 524
  - tworzenie, 523
- Policy, 745
- policytool, 754
- polityka bezpieczeństwa, 744, 745
  - pliki, 747
- połączenia bazodanowe, 227
  - JNDI, 288
- połączenia sieciowe
  - częściowo zamknięte, 196
  - gniazda, 187
  - HTTP, 186
  - klient, 185
  - porty, 184
  - serwer, 183
  - strumienie, 187
  - TCP, 187
  - telnet, 183
  - UDP, 187
  - URL, 204
  - wysyłanie danych do formularzy, 216
- populate(), 273
- porównywanie łańcuchów, 311
- porty, 184
- porządek alfabetyczny, 311
- POST, 218
- postOrderEnumeration(), 414
- postOrderTraversal(), 411
- PostScript, 603
- potok rysowania, 507
- potokowe tworzenie grafiki, 506
- pozwolenia, 742, 745, 758
  - implementacja klasy, 756
  - klasy, 755
  - operacje sieciowe, 753
  - parametry, 750, 751, 752
  - pliki polityki, 750
  - stos wywołań metod, 746
  - system plików, 752
- PRA, 607
- premain(), 857
- preOrderEnumeration(), 414
- preOrderTraversal(), 411
- PreparedStatement, 253, 254, 258
  - clearParameters(), 258
  - executeQuery(), 258
  - executeUpdate(), 258
  - setXxx(), 258
- prepareStatement(), 258, 263, 268
- PresentationDirection, 608
- Previous(), 268
- Principal, 767
  - getName(), 767
- Print, 584
- print(), 580, 583, 587, 588, 602, 913
- Printable, 580, 581, 582, 587
- printDialog(), 581, 588
- PrinterException, 581
- PrinterInfo, 608
- PrinterIsAcceptingJobs, 608
- PrinterJob, 581, 582, 583, 588, 589, 598
  - defaultPage(), 588
  - getPrinterJob(), 588
  - pageDialog(), 588
  - print(), 588
  - printDialog(), 588
  - setPageable(), 598
  - setPrintable(), 588
- PrinterLocation, 609
- PrinterMakeAndModel, 609
- PrinterMessageFromOperator, 609
- PrinterMoreInfo, 609
- PrinterMoreInfoManufacturer, 609
- PrinterName, 609
- PrinterResolution, 609

- PrinterState, 609
- PrinterStateReason, 609
- PrinterStateReasons, 609
- PrinterURI, 609
- printf(), 892, 911
  - formatowanie liczb, 899
- PrintJob, 581
- PrintJobAttribute, 604, 607
- PrintPreviewCanvas, 598
- PrintPreviewDialog, 591, 598
- PrintQuality, 606, 609
- PrintRequestAttribute, 604, 607
- PrintRequestAttributeSet, 581
- PrintService, 601, 602, 603, 610
  - createPrintJob(), 602
  - getAttributes(), 610
- PrintServiceAttribute, 604, 607
- PrintServiceLookup, 600, 602
- PrintWriter, 191, 218, 912, 913
- PrivilegedAction, 763, 767
- PrivilegedExceptionAction, 764, 767
- PrivilegedAction, 763, 764, 773
- problem uwierzytelniania, 784
- procedury ładowania klas
  - implementacja, 733
- procesor XSLT, 174
- processAnnotations(), 835
- profil ICC, 567
- ProgressMonitor, 463, 466, 474
  - close(), 474
  - isCanceled(), 474
  - setNote(), 474
  - setProgress(), 474
- ProgressMonitorInputStream, 463, 469, 474
- Properties, 102
- Property, 671
- PropertyChange, 674
- propertyChange(), 676, 680
- PropertyChangeEvent, 496, 500, 676, 681, 700
  - getNewValue(), 500, 681
  - getOldValue(), 681
  - getPropertyName(), 500, 681
- PropertyChangeListener, 676, 680
  - propertyChange(), 680
- PropertyChangeSupport, 675, 680
  - addPropertyChangeListener(), 680
  - fireIndexedPropertyChange(), 680
  - firePropertyChange(), 680
  - getPropertyChangeListeners(), 680
  - removePropertyChangeListener(), 680
- PropertyDescriptor, 683, 686, 687, 690
  - getPropertyType(), 686
  - getReadMethod(), 686
  - getWriteMethod(), 686
  - setPropertyEditorClass(), 690
- PropertyEditor, 694
- PropertyEditorSupport, 691, 696
  - getAsText(), 696
  - getCustomEditor(), 697
  - getJavaInitializationString(), 697
  - getTags(), 697
  - getValue(), 696
  - isPaintable(), 697
  - paintValue(), 697
  - setAsText(), 696
  - setValue(), 696
  - supportsCustomEditor(), 697
- PropertyPermission, 750
- PropertyVetoException, 493, 494, 495, 496, 500, 676, 677, 682
  - getPropertyChangeEvent(), 683
- prostokąt, 511, 521
  - ograniczający, 509
- ProtectionDomain, 747
  - getCodeSource(), 747
  - implies(), 747
- protokoły
  - HTTP, 208
  - Kerberos, 806
  - LDAP, 806
  - SMTP, 222
  - TCP, 187
  - UDP, 187
- prywatne metody macierzyste, 744
- przeciąganie zarysu ramki, 497
- przeciągnij i upuść, 625
  - gesty, 626
  - inicjacja operacji przeciągnięcia, 625
  - kopiowanie, 626
  - przesunięcie, 626
  - tworzenie łącza, 626
  - upuszczenie obiektu, 626
- przeglądarka klas, 415
- przekazywanie parametrów zdalnym metodom, 876
  - equals(), 884
  - hashCode(), 884
  - zdalne obiekty, 884
- przekształcenia afiniczne, 536, 572
- przekształcenia figur, 506
- przekształcenia układu współrzędnych, 507, 534
  - macierz przekształceń, 536
  - obrót, 534
  - pochylenie, 535
  - przesunięcie, 535
  - skalowanie, 534
  - składanie przekształceń, 535
- przekształcenia XSL
  - Java, 176

style, 173  
 szablon przekształcenia, 173  
 XSLT, 174

przebieg pliki źródłowe, 323

przestrzeń nazw, 148, 732

alias, 149  
 definiowanie, 149  
 identyfikatory, 148  
 URI, 148  
 XML, 148

przesunięcie, 535

przetwarzanie adnotacji, 845

przetwarzanie dokumentów XML, 107

przewijalne zbiory wyników zapytań, 263

przezroczystość, 541

przycinanie, 506, 539

określanie obszaru, 540

przyrostowe tworzenie obrazów, 565

PSA, 607

public, 831

PUBLIC, 120

pula połączeń, 289

punkty kontrolne transakcji, 284

putClientProperty(), 399, 403

## Q

QuadCurve2D, 509, 513

QuadCurve2D.Double, 522

quadTo(), 513, 522

QueuedJobCount, 609

## R

ramki wewnętrzne, 483

obiekt nasłuchujący weta zmiany, 495

odrysowywanie zawartości, 497

okna dialogowe, 496

przeciąganie zarysu ramki, 497

rozmieszczenie, 487

zamknięcie, 495

zgłaszanie weta zmiany właściwości, 495

Random, 797

RandomAccessFile, 563

Raster, 567, 570

getDataElements(), 570

getPixel(), 570

getPixels(), 570

read(), 198, 555, 560, 562, 563, 802

ReadableByteChannel, 198

Reader, 176

readObject(), 724

readThumbnail(), 561, 563

REAL, 236, 287

rebind(), 871

Rectangle2D, 508, 509, 511

Rectangle2D.Double, 711

ReferenceUriSchemesSupported, 609

ReflectPermission, 751

refleksja, 671, 843

REG\_BINARY, 935

REG\_DWORD, 935

REG\_SZ, 935

register(), 889

registerGroup(), 890

rejestr systemu Windows, 932, 933

edytor rejestru, 933

funkcje kontroli typów, 946

interfejs dostępu, 934

klucze, 934

metody macierzyste, 935

pobieranie wartości, 935

przeglądanie nazw kluczy, 936

uchwyt klucza, 936

węzły, 934

Win32RegKey, 934

Win32RegKeyNameEnumeration, 936

zapisywanie wartości, 936

rejestracja aktywności RMI, 874

rejestratory RMI, 875

rekordy, 231

wstawiania, 266

RELATIVE, 130

relative(), 264, 268

relativize(), 207

Relax NG, 119

releaseSavepoint(), 284, 286

ReleaseStringChars(), 903

ReleaseStringUTFChars(), 902, 904

ReleaseXxxArrayElements(), 922

reload(), 405, 410

REMAINDER, 130

REMARKS, 282

RemoteException, 866, 867, 884

remove(), 610

removeCellEditorListener(), 394

removeColumn(), 374, 380

removeElement(), 354, 355

removeNodeFromParent(), 404, 410

removePropertyChangeListener(), 680, 682, 699

removeTabAt(), 478, 482

removeTreeModelListener(), 426, 428

removeVetoableChangeListener(), 676, 681, 682

RenderingHints, 551, 554

RequestingUserName, 609

REQUIRED, 123

RescaleOp, 576, 579

reset(), 779

reshape(), 486, 499

- resolve(), 207
- resolveEntity(), 125
- ResourceBundle, 324, 327
  - getBundle(), 324, 328
  - getKeys(), 328
  - getObject(), 328
  - getString(), 328
  - getStringArray(), 328
  - handleGetObject(), 328
- ResourceBundles, 326
- Result, 176
- ResultSet, 242, 244, 245, 263, 270, 283
  - absolute(), 268
  - afterLast(), 269
  - beforeFirst(), 268
  - cancelRowUpdates(), 269
  - close(), 245
  - column(), 245
  - CONCUR\_READ\_ONLY, 264
  - CONCUR\_UPDATABLE, 266
  - deleteRow(), 269
  - first(), 268
  - getConcurrency(), 268
  - getMetaData(), 283
  - getRow(), 268
  - getType(), 268
  - getXxx(), 245
  - insertRow(), 269
  - isAfterLast(), 269
  - isBeforeFirst(), 269
  - isFirst(), 269
  - isLast(), 269
  - last(), 268
  - moveToCurrentRow(), 269
  - moveToInsertRow(), 269
  - next(), 244
  - previous(), 268
  - relative(), 268
  - TYPE\_SCROLL\_INSENSITIVE, 264, 266
  - updateRow(), 269
  - updateXxx(), 269
- ResultSetMetaData, 275, 283
  - getColumnCount(), 283
  - getColumnDisplaySize(), 283
  - getColumnLabel(), 283
  - getColumnName(), 283
- RetentionPolicy, 843
  - SOURCE, 845
- RetinaScanCallback, 773
- return, 923
- RGB, 541, 566
- RMI, 863, zob. także wywoływanie zdalnych metod
  - rejestracja aktywności, 874
  - rejestratory, 875
- RMIClassLoader, 875
- RMISecurityManager, 880
- role, 768
- rollback(), 284, 286
- rotate(), 535, 539
- RoundRectangle2D, 508, 509, 511
- RoundRectangle2D.Double, 522
- RowSet, 270, 272
  - execute(), 273
  - getCommand(), 273
  - getPassword(), 273
  - getURL(), 272
  - getUsername(), 272
  - setCommand(), 273
  - setPassword(), 273
  - setURL(), 272
  - setUsername(), 273
- rozmieszczenie
  - kaskadowe, 487
  - sąsiadujące, 487
- rozmycie obrazu, 577
- rozszerzenia maszyny wirtualnej, 729
- rozwiązywanie klasy, 728
- RSA, 780, 803
  - klucze, 803
- rt.jar, 729, 730
- RTF, 457
- run(), 767
- Runtime, 743
- RuntimePermission, 751
- rysowanie
  - antialiasing, 550
  - figur, 506
  - potok, 508
  - węzłów, 412
- rysunki, 505

**S**

- SASL, 806
- Savepoint, 286
  - getSavepointId(), 286
  - getSavepointName(), 286
- SAX, 107, 150
  - wyszukiwanie elementów, 152
- SAXParseException, 126
  - getColumnNumber(), 126
  - getLineNumber(), 126
- SAXParser, 152
  - parse(), 155
- SAXParserFactory, 152, 153, 154
  - isNamespaceAware(), 154
  - isValidating(), 154
  - newInstance(), 154

- newSAXParser(), 154
- setNamespaceAware(), 154
- setValidating(), 154
- SAXSource, 176, 177, 181
- scale(), 534, 535, 539
- Scanner, 191, 198
- schowek, 610
  - Clipboard, 611
  - Copy, 612
  - formaty danych, 615
  - interfejsy, 611
  - klasy, 611
  - przekazywanie danych, 611
  - przekazywanie obiektów Java, 621
  - przekazywanie obrazów, 617
  - przekazywanie tekstu, 612
  - rodzaje danych, 611
  - Transferable, 611, 615
- scrollPathToVisible(), 405, 409
- SecretKeySpec, 801
- SecureRandom, 797
- SecurityException, 744, 746
- SecurityManager, 746, 747
  - checkPermission(), 747
- SecurityPermission, 752
- sekwencje obrazów, 560
- SELECT, 233
  - FROM, 234
  - LIKE, 234
  - NOT LIKE, 234
  - WHERE, 234
- Serializable, 844
- SerializablePermission, 751
- serializacja, 706, 864
- ServerSocket, 191, 193
  - accept(), 193
  - close(), 193
- serwer, 183, 230, 862
  - aplikacji, 230, 288
  - FTP, 209
  - gniazda, 191
  - HTTP, 191
  - implementacja, 191
  - obsługa wielu klientów, 194
  - odpowiedź, 221
  - połączenia, 183
  - wątki, 194
  - Web, 183, 216
  - wysyłanie danych do formularzy, 216
- serwlety, 216, 288
- set(), 507, 671
- setAllowsChildren(), 401, 403
- setAllowUserInteraction(), 208, 214
- setAsksAllowsChildren(), 401, 402
- setAsText(), 690, 692, 696
- setAsynchronousLoadPriority(), 459
- setAttribute(), 160, 164
- setAttributeNS(), 164
- setAutoCommit(), 283, 285
- setAutoResizeMode(), 379
- setBottomComponent(), 477
- SetByteArrayRegion(), 935
- setCalendar(), 310
- setCellEditor(), 393
- setCellRenderer(), 357, 359, 393
- setCellSelectionEnabled(), 370
- setClip(), 539, 541, 583
- setClosable(), 498
- setClosed(), 495, 496, 499, 676
- setClosedIcon(), 414
- setColumnSelectionAllowed(), 370, 380
- setCommand(), 271, 273
- setComponentAt(), 483
- setComposite(), 507, 544, 549
- setConnectTimeout(), 215
- setContentHandler(), 181
- setContentPane(), 499
- setContents(), 612, 615
- setContextClassLoader(), 738
- setContinuousLayout(), 475, 477
- setDataElements(), 567, 570
- setDecomposition(), 317
- setDefault(), 296
- setDefaultRenderer(), 384
- setDisplayName(), 685
- setDoInput(), 208, 214
- setDoOutput(), 208, 214, 218, 221
- SetDoubleField(), 906
- setDragEnabled(), 631
- setDragMode(), 497, 498
- setEditable(), 406, 458
- setEntityResolver(), 125
- setErrorHandler(), 125
- setExceptionListener(), 724
- setExpert(), 686
- setFileName(), 668, 671
- setFixedCellHeight(), 353
- setFixedCellWidth(), 353
- setFrameIcon(), 485, 499
- setGroupingUsed(), 302
- setHeaderRenderer(), 384, 393
- setHeaderValue(), 384, 393
- setHidden(), 686
- setIcon(), 499
- setIconAt(), 483
- setIconifiable(), 499
- setIfModifiedSince(), 214
- setIgnoringElementContentWhitespace(), 124, 126



setIndeterminate(), 466, 473  
setInput(), 560  
SetIntField(), 906, 936  
setLayoutOrientation(), 348  
setLeafIcon(), 414  
setLeftComponent(), 477  
setLenient(), 310  
setLocale(), 295, 319  
setLogWriter(), 242  
setMaximizable(), 499  
setMaximum(), 463, 473, 493, 499  
setMaximumFractionDigits(), 302  
setMaximumIntegerDigits(), 302  
setMaxWidth(), 368, 380  
setMillisToDecidePopup(), 469  
setMillisToPopup(), 469  
setMinimum(), 463, 473  
setMinimumFractionDigits(), 302  
setMinimumIntegerDigits(), 302  
setMinWidth(), 368, 380  
setModifiedSince(), 208  
setName(), 685, 775  
setNamespaceAware(), 129, 149, 150, 153, 154  
setNote(), 474  
setNumberFormat(), 310  
setObject(), 699, 700, 705  
SetObjectArrayElement(), 919, 923  
SetObjectField(), 906  
setOneTouchExpandable(), 475, 477  
setOpenIcon(), 414  
setOutput(), 564  
setOutputProperty(), 164  
setPage(), 458, 460, 462  
setPageable(), 589, 598  
setPaint(), 507, 532, 533  
setParseIntegerOnly(), 302  
setPassword(), 271, 273, 775  
setPersistenceDelegate(), 724  
setPixel(), 565, 566, 571  
setPixels(), 566, 571  
setPreferredWidth(), 368, 380  
setPrintable(), 588  
setProgress(), 467, 474  
setProperty(), 880  
setPropertyEditorClass(), 687, 690  
setPropertyName(), 671  
setPrototypeCell(), 353  
setPrototypeCellValue(), 353  
setReadTimeout(), 215  
setRenderHints(), 551  
setRenderingHint(), 549, 550, 554  
setRenderingHints(), 506, 554  
setRequestProperty(), 208, 209, 215  
setResizable(), 368, 381, 498  
setRightComponent(), 477  
setRootVisible(), 400, 402  
setRowHeight(), 370, 379  
setRowMargin(), 370, 379  
setRowSelectionAllowed(), 370, 379  
setRunning(), 672  
setSavepoint(), 284, 286  
setSecurityManager(), 749  
setSeed(), 798  
setSelected(), 499  
setSelectedIndex(), 478, 482  
setSelectionMode(), 345, 370, 381  
setShortDescription(), 686  
setShowsRootHandles(), 400, 402  
setSoTimeout(), 188  
SetStaticXxxField(), 910  
setStrength(), 317  
setString(), 253, 464, 473  
setStringPainted(), 464, 473  
setStroke(), 507, 524, 525, 530, 531  
setTabLayoutPolicy(), 479, 483  
setTableName(), 273  
setText(), 459, 666  
setTimeZone(), 310  
setTitle(), 701  
setTitleAt(), 482  
setTopComponent(), 477  
setToRotation(), 537, 538  
setToScale(), 537, 538  
setToShear(), 537, 538  
setToTranslation(), 537, 538  
setTransferHandle(), 630  
setTransform(), 537, 539  
setURL(), 271, 272  
setUseCaches(), 208, 214  
setUsername(), 271, 273  
setUserObject(), 397, 402  
setValidating(), 126, 154  
setValue(), 473, 696, 934, 935, 936  
setValueAt(), 367, 392  
setVisible(), 486, 500  
setVisibleRowCount(), 345, 348  
setWidth(), 369, 381  
setXxx(), 258  
SetXxxArrayRegion(), 921, 922  
SetXxxField(), 910  
Severity, 609  
SGML, 103  
SHA1, 777  
Shape, 520, 523  
ShapeMaker, 521  
ShapePanel, 521  
shear(), 535, 539  
SheetCollate, 609



- short, 287
- ShortLookupTable, 576
- shouldSelectCell(), 391, 392, 394
- show(), 500
- shutdownInput(), 198
- shutdownOutput(), 198
- Sides, 609
- sieć, 183
  - hasła dostępu, 209
  - poczta elektroniczna, 222
  - przesyłanie danych, 191
- SimpleBeanInfo, 684, 685
  - loadImage(), 685
- SimpleDateFormat, 319
- SimpleDoc, 601, 603
- SimpleLoginModule, 769, 771
- SimplePrincipal, 769
- SimulatedActivity, 466
- SINGLE\_TREE\_SELECTION, 415
- skalowanie, 534
- składanie obrazów, 541
  - CLEAR, 543
  - DST, 543
  - DST\_ATOP, 543
  - DST\_IN, 543
  - DST\_OUT, 543
  - DST\_OVER, 543
  - piksel docelowy, 542
  - projektowanie reguły, 542
  - reguły Portera-Duffa, 544
  - reguły składania, 542
  - SRC, 543
  - SRC\_ATOP, 543
  - SRC\_IN, 543
  - SRC\_OUT, 543
  - SRC\_OVER, 543
  - XOR, 543
- składanie przekształceń, 535
- składnica kluczy, 782, 783, 790
- składowe obiektu, 906
- skrót wiadomości, 777
- skrypty CGI, 216
- SMALLINT, 236, 287
- SMTP, 222
- Socket, 187, 188, 198
  - connect(), 188
  - getInputStream(), 187
  - getOutputStream(), 187
  - isClosed(), 189
  - isConnected(), 189
  - isInputShutdown(), 198
  - isOutputShutdown(), 198
  - setSoTimeout(), 188
  - shutdownInput(), 198
  - shutdownOutput(), 198
- SocketChannel, 198, 204
- SocketPermission, 750
- SocketTimeoutException, 215
- sort(), 312
- sortowanie
  - porządek alfabetyczny, 311
- Source, 176
- splot, 577
- spójność bazy danych, 283
- SQL, 227, 231
  - aktualizowalne zbiory wyników zapytań, 265
  - ARRAY, 287
  - CREATE TABLE, 235, 242
  - DDL, 244
  - DROP TABLE, 242
  - FROM, 233
  - funkcje, 235
  - INSERT, 235
  - LIKE, 234
  - łączenie tabel, 232
  - metadane, 274
  - modyfikacja danych, 235
  - NOT LIKE, 234
  - polecenia, 245
  - polecenia przygotowane, 252
  - SELECT, 233
  - słowa kluczowe, 233
  - tworzenie tabel, 235
  - typy danych, 236, 287
  - WHERE, 234
  - wstawianie danych, 235
  - wykonywanie zapytań, 252
  - wynik zapytania, 232
  - wypełnianie bazy danych, 248
  - zapytania, 233
  - zarządzanie połączeniami, 245
  - zbiór wyników, 245
- SQLException, 247, 284
  - getErrorCode(), 248
  - getNextException(), 247
  - getSQLState(), 248
- SQLPermission, 752
- SRC, 543
  - SRC\_ATOP, 543
  - SRC\_IN, 543
  - SRC\_OUT, 543
  - SRC\_OVER, 543
- sRGB, 567
- SSL, 806
- startDocument(), 151, 155
- startElement(), 151, 152, 153, 155
- stateChanged(), 480

- Statement, 245, 263, 286, 725
  - addBatch(), 286
  - close(), 244, 245
  - createStatement(), 242
  - execute(), 244
  - executeBatch(), 286
  - executeQuery(), 244
  - executeUpdate(), 242, 244
  - getResultSet(), 244
  - getUpdateCount(), 244
- static, 831
- stdarg.h, 919
- sterowniki JDBC, 228
  - typ 1, 228
  - typ 2, 229
  - typ 3, 229
  - typ 4, 229
- stopCellEditing(), 391, 392, 394
- StreamPrintService, 603
- StreamPrintServiceFactory, 603
  - getPrintService(), 604
- StreamResult, 165
- StreamSource, 176, 177, 180
- strefy czasowe, 310
- String, 287, 312, 913
- StringBuffer, 472
- StringSelection, 612, 617
- Stroke, 524
- strona WWW, 216
- struktura dokumentu XML, 104
- struktury danych
  - nieskończone, 428
- strumienie, 187, 219
  - monitorowanie postępu, 469
  - tworzenie, 470
  - usługi drukowania, 603
- strumienie szyfrujące, 801
  - CipherInputStream, 802
- Subject, 767
  - doAs(), 767
  - doAsPrivileged(), 767
  - getPrincipals(), 767
- sun.misc.Base64Encoder, 214
- SupportedValuesAttribute, 604
- supportsBatchUpdates(), 286
- supportsCustomEditor(), 694, 697
- supportsResultSetConcurrency(), 265, 270
- supportsResultSetType(), 265, 269
- SVG, 162
- Swing
  - drzewa, 394
  - filtr strumieni, 469
  - formatowanie tekstu, 429
  - JDesktopPane, 485
  - JEditorPane, 457
  - JFrame, 485
  - JInternalFrame, 485
  - JList, 344
  - JProgressBar, 463
  - JScrollPane, 344
  - JSplitPane, 475
  - JTabbedPane, 478
  - JTextArea, 457
  - JTextField, 457
  - JTree, 394
  - listy, 343
  - monitory postępu, 466
  - organizatory komponentów, 474
  - panele dzielone, 475
  - panele pulpitu, 483
  - panele z zakładkami, 478
  - pasek przewijania, 362
  - przekazywanie danych, 627
  - ramki wewnętrzne, 483
  - rozmieszczenie komponentów, 487
  - tabele, 359
  - tekst, 457
  - wskaźnik postępu, 463
- Swng, 675
- sygnatury metody, 911
- SyncProviderException, 272, 273
- System, 897
  - loadLibrary(), 896, 897
  - setSecurityManager(), 749
- szeregowanie parametrów, 864, 865
- szyfr Cezara, 736
- szyfrowanie, 736, 795
  - AES, 796, 797, 803
  - algorytmy, 795
  - Cipher, 795
  - DES, 795, 796
  - dopełnienie ostatniego bloku, 796
  - klucze, 796, 797
  - pliki klas, 737
  - RSA, 780, 803
  - strumienie, 801
  - symetryczne, 795
  - tryb pracy algorytmu, 796
  - z kluczem publicznym, 803

## §

- ścieżki drzewa, 403
- śląd pędzla, 507, 524
  - połączenia, 525
  - przerwany wzór, 526
  - szerokość, 524
  - wartość graniczna, 525
  - zakończenia, 524

## T

- tabele, 231, 359
  - drukowanie, 363
  - edycja zawartości komórek, 367
  - implementacja edytora komórek, 390
  - JTable, 359
  - kolumny, 368
  - model, 359, 363
  - model wyboru, 370
  - obiekt rysujący, 367
  - prezentacja danych, 367
  - przesuwanie kolumny, 362
  - przewijanie zawartości, 362
  - rysowanie komórek, 367
  - szerokość kolumn, 362, 368
  - tworzenie, 235
  - tworzenie edytorów, 390
  - ukrywanie kolumn, 374
  - widoki, 362
  - wybór kolumn, 370
  - wybór komórek, 370
  - wybór wierszy, 370
  - wysokość komórek, 370
  - wyświetlanie kolumn, 374
- TABLE\_CAT, 282
- TABLE\_NAME, 282
- TABLE\_SCHEM, 282
- TABLE\_TYPE, 282
- TableCellEditor, 391, 392, 393
  - getTableCellEditorComponent(), 393
- TableCellRenderer, 383, 393
  - getTableCellRendererComponent(), 393
- TableColumn, 368, 380, 393
  - setMaxWidth(), 380
  - setMinWidth(), 380
  - setPreferredWidth(), 380
  - setResizable(), 381
  - setWidth(), 381
- TableColumnModel, 368, 380
  - getColumn(), 380
  - setCellEditor(), 393
  - setCellRenderer(), 393
  - setHeaderRenderer(), 393
  - setHeaderValue(), 393
- TableModel, 366
  - getColumnCount(), 366
  - getColumnName(), 367
  - getRowCount(), 366
  - getValueAt(), 367
  - isCellEditable(), 367
  - setValueAt(), 367
- tablice, 916, 919
  - C, 919
  - C++, 920
  - jarray, 919
  - rozmiar, 919
- TCP, 187
- tekst, 457
  - hiperłącza, 459
- telnet, 183
- testowanie, 830
- Text, 110
- text/plain, 215
- TextField, 297
- TextLayout, 540, 541
  - getAdvance(), 541
  - getAscent(), 541
  - getDescent(), 541
  - getLeading(), 541
- TexturePaint, 532, 533, 534
- Thread, 738
  - getContextClassLoader(), 738
  - setContextClassLoader(), 738
- ThreadedEchoHandler, 194
- Throw(), 923, 927
- ThrowNew(), 923, 927
- throws, 839
- Time, 287
- TIME, 236, 287
- Timestamp, 287
- TIMESTAMP, 236, 287
- TimeZone, 310
  - getAvailableIDs(), 310
  - getDefault(), 310
  - getDisplayName(), 311
  - getID(), 311
  - getTimeZone(), 311
  - inDaylightTime(), 311
  - useDaylightTime(), 311
- TitlePositionEditor, 688
- toArray(), 610
- Tomcat, 288
- Toolkit, 614
  - getSystemClipboard(), 614
- toString(), 296, 303, 396, 521, 841
- transakcje, 283
  - aktualizacje wsadowe, 284
  - automatyczne zatwierdzanie, 283
  - odwołanie, 283
  - punkty kontrolne, 284
  - tworzenie, 283
- Transferable, 611, 615
  - getTransferData(), 615
  - isDataFlavorSupported(), 615
- TransferHandler, 630
- transform(), 164, 176, 507, 537, 539
- Transformer, 164
  - setOutputProperty(), 164
  - transform(), 164

- TransformerFactory, 164, 180
    - newInstance(), 164
  - translate(), 535, 539, 590
  - TreeCellRenderer, 412, 414
    - getTreeCellRendererComponent(), 414
  - TreeModel, 116, 396, 402, 422, 428
    - addTreeModelListener(), 428
    - getChild(), 428
    - getChildCount(), 428
    - getIndexOfChild(), 428
    - getRoot(), 428
    - isLeaf(), 402, 428
    - removeTreeModelListener(), 428
    - valueForPathChanged(), 429
  - TreeModelEvent, 429
  - TreeModelListener, 429
    - treeNodesChanged(), 429
    - treeNodesInserted(), 429
    - treeNodesRemoved(), 429
    - treeStructureChanged(), 429
  - TreeNode, 396, 402, 403, 409
    - children(), 409
    - getAllowsChildren(), 402
    - getChildAt(), 409
    - getChildCount(), 409
    - getParent(), 409
    - isLeaf(), 402
  - treeNodesChanged(), 429
  - treeNodesInserted(), 429
  - treeNodesRemoved(), 429
  - TreePath, 404, 409, 416
    - getLastPathComponent(), 409
  - TreeSelectionEvent, 416, 421
    - getPath(), 421
    - getPaths(), 421
  - TreeSelectionListener, 415, 421
    - valueChanged(), 421
  - TreeSelectionModel, 415
  - treeStructureChanged(), 429
  - trim(), 110, 298
  - trwałość ziarenek JavaBeans, 705, 715
    - dowolne dane, 709
    - implementacja delegatu trwałości, 710
    - kodowanie obiektu, 711
    - odtworzenie stanu obiektów, 712
    - serializacja, 706
    - tworzenie obiektu na podstawie właściwości, 711
    - tworzenie obiektu przez metodę fabryki, 712
    - właściwości tymczasowe, 713
    - XMLEncoder, 709, 710
  - try, 246
  - tunelowanie HTTP, 875
  - tworzenie
    - certykatów, 781
    - dokumentów XML, 160
    - drzewa DOM, 160
    - grafiki, 506
    - klas pozwolen, 755
    - nazw metod, 672
    - plików JAR, 663
    - ziarenek JavaBeans, 660
  - TYPE\_BILINEAR, 572
  - TYPE\_BYTE\_GRAY, 568
  - TYPE\_FORWARD\_ONLY, 264
  - TYPE\_INT\_ARGB, 566
  - TYPE\_SCROLL\_INSENSITIVE, 264, 266
  - TYPE\_SCROLL\_SENSITIVE, 264
  - typy danych C, 898
  - typy danych Java, 287, 898
  - typy danych SQL, 236, 287
  - typy MIME, 616
- ## U
- UDP, 187
  - układ współrzędnych, 534
  - unbind(), 870, 871
  - UnicastRemoteObject, 867
    - exportObject(), 867
  - Unicode, 291, 323
  - uniform resource identifiers, 205
  - uniform resource locator, 205
  - uniform resource name, 205
  - UnixLoginModule, 763
  - UnixNumericGroupPrincipal, 763
  - UnixPrincipal, 762
  - UnknownHostException, 187
  - UPDATE, 242, 253
  - update(), 779, 796, 800
  - updateDouble(), 266
  - updateRow(), 266, 267, 269
  - uporządkowanie łańcuchów, 312
  - uporządkowanie słownikowe, 311
  - URI, 148, 205, 206, 606
    - absolutny, 205
    - hierarchiczny, 206
    - identyfikatory, 206
    - nieprzenikalny, 205
    - relatywizacja, 206
    - rozwiązywanie, 206
    - specyfikacja, 205
    - względny, 205
  - URL, 148, 204, 205, 214, 749
    - nagłówki żądań, 208
    - openConnection(), 214
    - openStream(), 214
    - pobieranie informacji, 207
  - URLClassLoader, 729
  - URLConnection, 205, 207, 208, 214, 218, 221
    - connect(), 215

getAllowUserInteraction(), 214  
 getConnectTimeout(), 215  
 getContent(), 216  
 getContentEncoding(), 215  
 getContentLength(), 215  
 getContentType(), 215  
 getDate(), 215  
 getDoInput(), 214  
 getDoOutput(), 214  
 getExpiration(), 215  
 getHeaderField(), 215  
 getHeaderFieldKey(), 215  
 getHeaderFields(), 215  
 getIfModifiedSince(), 214  
 getLastModified(), 216  
 getRequestProperties(), 215  
 getUseCaches(), 214  
 openConnection(), 207  
 openInputStream(), 216  
 openOutputStream(), 216  
 setAllowUserInteraction(), 208, 214  
 setConnectTimeout(), 215  
 setDoInput(), 208, 214  
 setDoOutput(), 208, 214  
 setIfModifiedSince(), 214  
 setModifiedSince(), 208  
 setReadTimeout(), 215  
 setRequestProperty(), 208, 215  
 setUseCaches(), 208, 214  
 URLDecoder, 222  
   decode(), 222  
 URLEncoder, 222  
   encode(), 222  
 URN, 205  
 useDaylightTime(), 311  
 usługi drukowania, 599  
   GIF, 600  
   rodzaje dokumentów, 599, 600  
   strumienie, 603  
   źródło danych, 599, 600  
 usługi rejestru początkowego RMI, 868  
 usługi sieciowe, 183  
 usługi uwierzytelniania, 762  
 UTF-16, 323, 900, 902  
 UTF-32, 902  
 UTF-8, 323, 900  
 uwierzytelnianie  
   oparte na rolach, 768  
 uwierzytelnianie użytkowników, 762  
   grant, 763  
   JAAS, 762  
   moduł logowania, 763  
   moduły, 763  
   nadzorca, 763

podmiot, 763  
 pozwolenia, 764  
 uwierzytelnianie wiadomości, 784  
   klucze, 784  
   łańcuch zaufania, 784  
   podpis zaufanego pośrednika, 785  
   zaufany pośrednik, 785

## V

va\_list, 919  
 valueChanged(), 346, 415, 421  
 valueForPathChanged(), 426, 429  
 VARCHAR, 236, 287  
 Variable, 427  
 Verisign, Inc., 785  
 vetoableChange(), 496, 500, 681  
 VetoableChangeListener, 495, 500, 676, 681  
   vetoableChange(), 500, 681  
 VetoableChangeSupport, 676, 677, 681  
   addVetoableChangeListener(), 681  
   fireVetoableChange(), 682  
   getVetoableChangeListeners(), 682  
   removeVetoableChangeListener(), 681  
 Visual Basic, 658  
 vm\_args, 928

## W

W3C, 152  
 waluta, 297, 302  
   formatowanie, 302  
   identyfikatory, 303  
 warning(), 125, 126  
 warstwa pośrednia, 230  
 wartość alfa, 542  
 wątki, 194  
 wdrażanie aplikacji RMI, 871  
 Web, 183, 216  
 WebRowSet, 270  
 wektor obiektów, 354  
 weryfikacja dokumentu XML, 124  
 weryfikacja kodu maszyny wirtualnej, 738  
 weryfikacja podpisu cyfrowego, 781  
 weryfikacja podpisu plików JAR, 783  
 weryfikator kodu, 738, 741  
 węzły, 395  
   nadrzędne, 395  
   podrzędne, 395  
   przeglądanie, 410  
   rysowanie, 412  
 WHERE, 234  
 wielokąty, 509, 514, 521  
 Win32RegKey, 934  
 Win32RegKeyNameEnumeration, 936

- WindowListener, 497
- Windows, 931, 933
  - rejestr, 933
- wizualne projektowanie aplikacji, 671
- właściwości, 667
  - indeksowane, 674
  - ograniczone, 676
  - powiązane, 674
  - proste, 673
  - ziarnka, 673
- WordCheckPermission, 756
- WritableByteChannel, 198
- WritableRaster, 565, 567, 570
  - setDataElements(), 570
  - setPixel(), 571
  - setPixels(), 571
- write(), 198, 555, 562, 564, 802
- writeInsert(), 561, 564
- writeObject(), 724
- writeStatement(), 724
- WSDL, 863
- wskazówki operacji graficznych, 506, 549
  - antialiasing, 549
  - KEY\_ALPHA\_INTERPOLATION, 550
  - KEY\_ANTIALIASING, 550
  - KEY\_COLOR\_RENDERING, 550
  - KEY\_DITHERING, 550
  - KEY\_FRACTIONAL\_METRICS, 550
  - KEY\_INTERPOLATION, 550
  - KEY\_RENDERING, 550
  - KEY\_STROKE\_CONTROL, 550
  - KEY\_TEXT\_ANTIALIASING, 550
- wskaźnik postępu, 463
  - JProgressBar, 463
  - monitory postępu, 466
  - pasek postępu, 463
  - ProgressMonitor, 466
  - strumienie wejścia, 469
- współrzędne, 509
  - ekranowe, 534
  - użytkownika, 534
- wyjątki
  - ArrayIndexOutOfBoundsException, 923
  - ArrayStoreException, 923
  - CloneNotSupportedException, 884
  - IllegalStateException, 560
  - IndexOutOfBoundsException, 560
  - MissingResourceException, 325
  - ParseException, 298
  - PrinterException, 581
  - PropertyVetoException, 676
  - RemoteException, 866, 867, 884
  - SecurityException, 744, 746
  - SQLException, 284
  - SyncProviderException, 272
  - UnknownHostException, 187
  - wykonywanie zapytań SQL, 252
  - wykres, 688
  - wykrywanie krawędzi, 578
  - wynik zapytania, 232
  - wypełnianie obszaru, 506, 507, 508, 532
    - gradient, 532
    - kolory, 533
    - obrazek wzorca, 533
    - prostokąt zakotwiczenia, 533
  - wrażenia XPath, 142
  - wyrzucanie wyjątków, 923
  - wysyłanie danych do formularzy, 216
  - wysyłanie poczty elektronicznej, 222
  - wytnij i wklej, 610
  - wywoływanie funkcji języka C, 892
    - parametry, 895
    - printf(), 892
  - wywoływanie metod języka Java, 912, 917
  - wywoływanie metod obiektów, 912
  - wywoływanie metod statycznych, 916
  - wywoływanie zdalnych metod, 861, 864
    - aktywacja obiektów serwera, 884
    - deskryptory aktywacji, 884
    - dynamiczne ładowanie klas, 878
    - identyfikator grupy, 886
    - implementacje, 866
    - interfejsy, 866
    - menedżer bezpieczeństwa, 879
    - namiastka, 864
    - plik polityki, 880
    - przekazywanie parametrów, 876
    - przygotowanie wdrożenia, 871
    - rejestracja aktywności, 874
    - rejestratory, 875
    - RMISecurityManager, 880
    - szeregowanie parametrów, 864
    - UnicastRemoteObject, 867
    - usługa rejestru początkowego, 868
    - wdrażanie aplikacji, 871
    - wyszukiwanie obiektów serwera, 868
  - wzorce model-widok-nadzorca, 395
  - wzorce nazw, 671
  - wzorce projektowe, 671

**X**

- X Window, 610
- XML, 102, 864
  - atrybuty, 104, 105
  - ATTLIST, 122
  - CDATA, 106, 123
  - deklaracja typu dokumentu, 104
  - DOM, 107

- element korzenia, 104
  - instrukcje przetwarzania, 106
  - JAXP, 107
  - komentarze, 106
  - kontrola poprawności dokumentów, 118
  - mieszana zawartość, 105
  - parser, 107
  - parsowanie dokumentów, 107
  - PCDATA, 121
  - pliki, 102
  - przestrzeń nazw, 148
  - przetwarzanie dokumentów, 107
  - referencje bytów, 106
  - referencje znaków, 106
  - SAX, 107, 150
  - struktura dokumentu, 104
  - wartości atrybutów, 104
  - wartości domyślne atrybutów, 123
  - wielkość znaków, 103
  - XPath, 142
  - znaczniki, 103
  - XML Schema, 119, 126, 148
    - atrybuty, 128
    - definicje elementów, 128
    - parsowanie dokumentu XML, 129
    - powtórzenia elementów, 128
    - przestrzeń nazw, 127
    - typ elementu, 127
    - typy proste, 127
    - typy złożone, 127
  - XMLDecoder, 724
    - getExceptionListener(), 725
    - readObject(), 724
    - setExceptionListener(), 724
  - XMLEncoder, 706, 707, 709, 710, 724
    - writeObject(), 724
    - writeStatement(), 724
  - xmlns, 148
  - xmlns:alias, 149
  - XMLReader, 181
    - parse(), 181
    - setContentHandler(), 181
  - XPath, 142, 147, 173
    - evaluate(), 147
    - funkcje, 143
    - wartość wyrażenia, 143
    - wyrażenia, 142
    - wyznaczanie wyrażzeń, 143
    - zbiór węzłów, 143
  - XPathConstants, 143
    - NODE, 143
    - NUMBER, 143
  - XPathFactory, 143, 147
    - newInstance(), 147
    - newXPath(), 147
  - xsd:attribute, 128
  - xsd:boolean, 127
  - xsd:choice, 128
  - xsd:int, 127
  - xsd:schema, 128
  - xsd:sequence, 128
  - xsd:string, 127
  - XSL Schema Definition, 127
  - xsl:apply-templates, 173
  - xsl:output, 173
  - xsl:template, 173
  - xsl:value-of, 174
  - XSLT, 161, 173, 174
- ## Z
- zadania drukowania, 581
  - zakładki, 479
  - zapytania SQL, 233
    - przygotowane, 252
    - wykonywanie, 252
  - zasada Gödla, 739
  - zasada składania obrazów, 506, 507
  - zasoby, 324
    - lokalizacja, 324
  - zbiory atrybutów drukowania, 606
  - zbiory pozwoleń, 744
  - zbiory rekordów, 263, 266, 270
    - aktualizowalne, 265
    - buforowane, 271
    - CachedRowSet, 271
    - modyfikacja, 272
    - przewijalne, 263
    - RowSet, 270
    - sprawdzanie, 272
  - zbiory znaków, 322
  - zbiór Mandelbrota, 567
  - zdalne obiekty, 884
  - zdalne wywoływanie metod, 863
  - zdarzenia, 668
    - PropertyChange, 674
  - zestawy znaków, 322
  - ziarnka, 657, 658
    - edytor właściwości, 687
    - EJB, 658
    - formatki, 667
    - indywidualizacja, 697
    - JavaBeans, 658
    - klasa informacyjna, 671, 683
    - korzystanie, 664
    - metody, 672
    - obiekty nasłuchujące, 674, 675
    - plik manifestu, 663
    - pliki JAR, 663

ziarnka

- projektowanie, 669
- trwałość, 705
- tworzenie, 662
- tworzenie aplikacji, 662
- typy właściwości, 673
- właściwości indeksowane, 674
- właściwości ograniczone, 676
- właściwości powiązane, 674
- właściwości proste, 673
- wzorce nazw właściwości, 669
- wzorce nazw zdarzeń, 669
- zdarzenia, 672

ZIP, 729

zmiennie

- LD\_LIBRARY\_PATH, 932
- znaczniki XML, 103
- znaki, 322

## Ż

źródło danych, 288

źródło danych JDBC, 237

źródło danych JNDI, 288

źródło kodu, 744

  certyfikaty, 744

  lokalizacja kodu, 744



# 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>

## WYKORZYSTAJ W PEŁNI MOŻLIWOŚCI JĘZYKA JAVA 7!

Zastanawiasz się, dlaczego język Java zdobył taką popularność? Przyczyn jest co najmniej kilka: automatyczne zarządzanie pamięcią, możliwość uruchamiania kodu na różnych platformach, ogrom dodatkowych narzędzi oraz wyjątkowo aktywna społeczność. Nie bez znaczenia jest również świetna dokumentacja, wsparcie dla usług sieciowych oraz aktywny rozwój, którego dowodem są kolejne wersje tego języka.

Dziewiąte wydanie bestsellerowej pozycji *Java. Techniki zaawansowane* zostało zaktualizowane i uzupełnione o nowinki z najnowszej wersji języka Java oznaczonej numerem 7. W trakcie lektury dowiesz się, jak wydajnie korzystać ze strumieni, wyrażeń regularnych oraz baz danych. Java 7 to całkowicie nowy, mocno rozbudowany dostęp do plików — opis wszystkich niuansów znajdziesz w tej publikacji. Co jeszcze? Tworzenie aplikacji dla różnych języków i lokalizacji, zaawansowane wykorzystanie biblioteki Swing oraz dystrybucja stworzonych aplikacji. To tylko niektóre z zagadnień poruszonych w tej wyjątkowej książce, która musi się znaleźć na półce każdego programisty języka Java.

### POZNAJ:

- nowości języka Java 7
- zaawansowane techniki korzystania ze strumieni
- całkowicie nowe metody dostępu do plików
- techniki łączenia się z bazą danych
- potencjał języka Java

 PRENTICE HALL  
PEARSON EDUCATION

**helion.pl**  
księgarnia  
internetowa

Nr katalogowy: 15689

 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/nawosci>

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



cena: 149,00 zł

ISBN 978-83-246-7762-7



9 788324 677627

Informatyka w najlepszym wydaniu