

PROGRAMOWANIE

# wJavie

SOLIDNA  
WIEDZA  
W PRAKTYCE

WYDANIE XI



PAUL DEITEL | HARVEY DEITEL

Helion 

Tytuł oryginału: Java How to Program, Early Objects (11th Edition)

Tłumaczenie: Rafał Jońca

ISBN: 978-83-283-3973-6

Authorized translation from the English language edition, entitled: JAVA HOW TO PROGRAM, EARLY OBJECTS, Eleventh Edition; ISBN 0134743350; by Harvey M. Deitel; and by Paul J. Deitel; published by Pearson Education, Inc. Copyright © 2018, 2015, 2012 and 2009 by Pearson Education, Inc., Hoboken, New Jersey 07030.

Ali 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 © 2018.

Java™ and Netbeans™ screenshots ©2017 by Oracle Corporation, all rights reserved. Reprinted with permission.

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

Wydawnictwo HELION

ul. Kościuszki 1c, 44-100 GLIWICE

tel. 32 231 22 19, 32 230 98 63

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

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

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

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- Kup książkę
- Poleć książkę
- Oceń książkę

- Księgarnia internetowa
- Lubię to! » Nasza społeczność



## Spis treści

<b>Przedmowa</b>	<b>29</b>
<b>Wstęp</b>	<b>31</b>
<b>Zanim zaczniesz</b>	<b>49</b>
<b>1. Wprowadzenie do komputerów, internetu i Javy</b>	<b>55</b>
1.1. Wprowadzenie	56
1.2. Sprzęt i oprogramowanie	58
1.2.1. Prawo Moore'a	59
1.2.2. Sposób organizacji komputera	59
1.3. Hierarchia danych	61
1.4. Języki maszynowe, języki assemblerowe i języki wysokiego poziomu	64
1.5. Wprowadzenie do technologii obiektowej	65
1.5.1. Samochód jako obiekt	65
1.5.2. Metody i klasy	66
1.5.3. Tworzenie egzemplarzy	66
1.5.4. Wielokrotne użycie	66
1.5.5. Komunikaty i wywoływanie metod	67
1.5.6. Atrybuty i zmienne instancji	67
1.5.7. Enkapsulacja i ukrywanie informacji	67
1.5.8. Dziedziczenie	67
1.5.9. Interfejsy	67
1.5.10. Obiektowa analiza i projektowanie	68
1.5.11. Język UML	68
1.6. Systemy operacyjne	68
1.6.1. Windows — własnościowy system operacyjny	69
1.6.2. Linux — system operacyjny o otwartym kodzie źródłowym	69
1.6.3. Systemy macOS i iOS firmy Apple dla urządzeń iPhone, iPad i iPod Touch	70
1.6.4. System Android firmy Google	70

1.7.	Języki programowania	71
1.8.	Java	74
1.9.	Typowe środowisko programistyczne wykorzystujące Javę	74
1.10.	Testowanie przykładowej aplikacji napisanej w Javie	78
1.11.	Internet i sieć WWW	82
1.11.1.	Internet, czyli sieć sieci	83
1.11.2.	Sieć WWW — przyjazny internet	83
1.11.3.	Usługi sieciowe i mashupy	83
1.11.4.	Internet rzeczy	84
1.12.	Technologie związane z oprogramowaniem	85
1.13.	Otrzymywanie odpowiedzi na pytania	87

## **2. Wprowadzenie do aplikacji Javy, wejścia – wyjścia i operatorów** **91**

2.1.	Wprowadzenie	92
2.2.	Twój pierwszy program — wyświetlenie wiersza tekstu	92
2.2.1.	Kompilacja aplikacji	97
2.2.2.	Wykonywanie aplikacji	98
2.3.	Modyfikacja pierwszego programu	99
2.4.	Wyświetlanie tekstu metodą printf	101
2.5.	Inna aplikacja — dodawanie liczb całkowitych	102
2.5.1.	Deklaracja import	102
2.5.2.	Deklaracja i utworzenie obiektu Scanner w celu pobrania danych z klawiatury	103
2.5.3.	Prośba o wprowadzenie danych	104
2.5.4.	Deklaracja zmiennej dla liczby całkowitej i pobranie wartości z klawiatury	104
2.5.5.	Pobranie drugiej liczby całkowitej	105
2.5.6.	Użycie zmiennych w obliczeniach	105
2.5.7.	Wyświetlenie wyniku obliczeń	105
2.5.8.	Dokumentacja API Javy	106
2.5.9.	Deklaracja i inicjalizacja zmiennej jako osobne instrukcje	106
2.6.	Zagadnienia dotyczące pamięci	106
2.7.	Operacje arytmetyczne	107
2.8.	Podjęmowanie decyzji — operatory równości i relacji	111
2.9.	Podsumowanie	115

<b>3. Wprowadzenie do klas, obiektów, metod i tekstów</b>	<b>127</b>
3.1. Wprowadzenie	128
3.2. Zmienne instancji, metody ustawiające i metody pobierające	129
3.2.1. Klasa Account ze zmienną instancji oraz metodą ustawiającą i metodą pobierającą	129
3.2.2. Klasa AccountTest, która tworzy i stosuje obiekt klasy Account	133
3.2.3. Kompilacja i wykonanie aplikacji z wieloma klasami	136
3.2.4. Diagram klas UML dla klasy Account	136
3.2.5. Dodatkowe uwagi na temat klasy AccountTest	137
3.2.6. Inżynieria oprogramowania z prywatnymi zmiennymi instancji i publicznymi metodami dostępowymi	138
3.3. Klasa Account — inicjalizacja obiektów za pomocą konstruktorów	139
3.3.1. Deklaracja konstruktora klasy Account dla własnej inicjalizacji obiektów	140
3.3.2. Klasa AccountTest — inicjalizacja obiektów Account w momencie ich tworzenia	141
3.4. Klasa Account ze stanem konta — liczby zmiennoprzecinkowe	142
3.4.1. Klasa Account ze zmienną instancji balance typu double	143
3.4.2. Klasa AccountTest używająca klasy Account	145
3.5. Typy podstawowe i typy referencyjne	148
3.6. (Opcjonalnie) Studium przypadku GUI i grafiki — prosty interfejs graficzny	149
3.6.1. Czym jest interfejs graficzny?	150
3.6.2. FXML i narzędzie Scene Builder	150
3.6.3. Aplikacja powitalna — wyświetlenie tekstu i obrazka	150
3.6.4. Uruchomienie narzędzia Scene Builder i utworzenie pliku Welcome.fxml	151
3.6.5. Dodanie obrazu do folderu zawierającego plik Welcome.fxml	151
3.6.6. Tworzenie kontenera układu VBox	151
3.6.7. Konfiguracja kontenera układu VBox	151
3.6.8. Dodanie i konfiguracja etykiety	153
3.6.9. Dodanie i konfiguracja ImageView	154
3.6.10. Podgląd wynikowego interfejsu aplikacji	155
3.7. Podsumowanie	156

<b>4. Struktury sterujące, część 1.;</b>	
<b>przypisanie i operatory ++ i --</b>	<b>165</b>
4.1. Wprowadzenie	166
4.2. Algorytmy	166
4.3. Pseudokod	167
4.4. Struktury sterujące	167
4.4.1. Struktury sekwencyjne w Javie	168
4.4.2. Instrukcje wyboru w Javie	169
4.4.3. Instrukcje iteracji w Javie	169
4.4.4. Podsumowanie instrukcji sterujących Javy	170
4.5. Instrukcja pojedynczego wyboru — if	170
4.6. Instrukcja podwójnego wyboru — if...else	171
4.6.1. Zagnieżdżone instrukcje if...else	172
4.6.2. Problem wiszącego else	173
4.6.3. Bloki	173
4.6.4. Operator warunku (?:)	174
4.7. Klasa Student — zagnieżdżone instrukcje if...else	175
4.8. Instrukcja iteracji while	177
4.9. Tworzenie algorytmów — iteracja sterowana licznikiem	179
4.10. Tworzenie algorytmów — iteracja sterowana znacznikiem	183
4.11. Tworzenie algorytmów — zagnieżdżone struktury sterujące	191
4.12. Złożone operatory przypisania	195
4.13. Operatory inkrementacji i dekrementacji	196
4.14. Typy podstawowe	199
4.15. Studium przypadku GUI i grafiki	
— obsługa zdarzeń i rysowanie linii	200
4.15.1. Test ukończonej aplikacji	200
4.15.2. Budowanie GUI aplikacji	201
4.15.3. Przygotowanie do interakcji z GUI w sposób programowy	205
4.15.4. Klasa DrawLinesController	207
4.15.5. Klasa DrawLines — główna klasa aplikacji	209
4.16. Podsumowanie	212
<b>5. Struktury sterujące, część 2.;</b>	
<b>operatory logiczne</b>	<b>229</b>
5.1. Wprowadzenie	230
5.2. Podstawy iteracji sterowanej licznikiem	230
5.3. Instrukcja iteracji for	231
5.4. Przykłady użycia instrukcji for	236
5.4.1. Aplikacja — suma liczb parzystych od 2 do 20	236
5.4.2. Aplikacja — kalkulator procentu składanego	237

5.5.	Instrukcja iteracji do...while	240
5.6.	Instrukcja switch dotycząca wielu wyborów	242
5.7.	Studium przypadku klasy AutoPolicy — tekst jako wartość w instrukcji switch	247
5.8.	Instrukcje break i continue	251
5.8.1.	Instrukcja break	251
5.8.2.	Instrukcja continue	251
5.9.	Operatory logiczne	252
5.9.1.	Operator warunkowy AND (&&)	253
5.9.2.	Operator warunkowy OR (  )	253
5.9.3.	Skrócone obliczenie wartości złożonych warunków	254
5.9.4.	Operatory logiczne AND (&&) i OR (  )	255
5.9.5.	Operator logiczny XOR (^)	255
5.9.6.	Operator negacji logicznej (!)	256
5.9.7.	Przykład użycia operatorów logicznych	256
5.10.	Podsumowanie programowania strukturalnego	258
5.11.	Studium przypadku GUI i grafiki — rysowanie prostokątów i owali	263
5.12.	Podsumowanie	266

## **6. Metody — dokładniejsze spojrzenie 279**

6.1.	Wprowadzenie	280
6.2.	Jednostki programu w Javie	280
6.3.	Metody statyczne, pola statyczne i klasa Math	282
6.4.	Metody z wieloma parametrami	284
6.5.	Uwagi na temat deklarowania i używania metod	288
6.6.	Stos wywołań metod i rekordy aktywacyjne	289
6.6.1.	Stos wywołań metod	289
6.6.2.	Ramki stosu	290
6.6.3.	Zmienne lokalne i ramki stosu	290
6.6.4.	Przepelnienie stosu	290
6.7.	Promocja argumentu i rzutowanie	291
6.8.	Pakiety API Javy	292
6.9.	Studium przypadku — bezpieczne generowanie liczb losowych	294
6.10.	Studium przypadku — gra losowa i wprowadzenie do typów enum	299
6.11.	Zasięg deklaracji	303
6.12.	Przeciążanie metod	306
6.12.1.	Deklarowanie przeciążonych metod	306
6.12.2.	Rozróżnianie przeciążonych metod	307
6.12.3.	Typy zwracane przez metody przeciążone	308
6.13.	(Opcjonalnie) Studium przypadku GUI i grafiki — kolory i wypełnione kształty	308
6.14.	Podsumowanie	312

<b>7. Tablice i obiekty ArrayList</b>	<b>327</b>
7.1. Wprowadzenie	328
7.2. Tablice	329
7.3. Deklaracja i tworzenie tablic	330
7.4. Przykłady użycia tablic	332
7.4.1. Tworzenie i inicjalizacja tablicy	332
7.4.2. Użycie inicjalizatora tablicy	333
7.4.3. Obliczenie wartości elementów tablicy	334
7.4.4. Sumowanie elementów tablicy	335
7.4.5. Graficzne przedstawienie danych z tablicy za pomocą wykresu słupkowego	336
7.4.6. Użycie elementów tablicy jako liczników	337
7.4.7. Użycie tablic do analizy wyników	338
7.5. Obsługa wyjątków — przetworzenie niepoprawnej odpowiedzi	340
7.5.1. Instrukcja try	341
7.5.2. Wykonywanie bloku catch	341
7.5.3. Metoda toString parametru wyjątku	341
7.6. Studium przypadku — tasowanie i rozdawanie kart	342
7.7. Rozszerzona instrukcja for	346
7.8. Przekazywanie tablic do metod	348
7.9. Przekazywanie przez wartość kontra przekazywanie przez referencję	350
7.10. Studium przypadku — klasa GradeBook wykorzystująca tablicę z ocenami	351
7.11. Tablice wielowymiarowe	356
7.11.1. Tablice tablic jednowymiarowych	357
7.11.2. Dwuwymiarowe tablice z wierszami o różnych długościach	357
7.11.3. Tworzenie tablic dwuwymiarowych za pomocą wyrażenia tworzenia tablic	357
7.11.4. Przykład tablicy dwuwymiarowej — wyświetlanie wartości elementów	358
7.11.5. Typowe operacje na tablicach wielowymiarowych wykonywane pętlami for	359
7.12. Studium przypadku — klasa GradeBook używająca tablicy dwuwymiarowej	360
7.13. Lista argumentów o zmiennej długości	365
7.14. Użycie argumentów wiersza poleceń	367
7.15. Klasa Arrays	369
7.16. Wprowadzenie do kolekcji i klasy ArrayList	372
7.17. (Opcjonalnie) Studium przypadku GUI i grafiki — rysowanie łuków	375
7.18. Podsumowanie	378



<b>8. Obiekty i klasy — dokładniejsze spojrzenie</b>	<b>401</b>
8.1. Wprowadzenie	402
8.2. Studium przypadku — klasa Time	402
8.3. Sterowanie dostępem do składowych	407
8.4. Odnoszenie się do składowych aktualnego obiektu referencją <code>this</code>	408
8.5. Studium przypadku klasy Time — przeciążanie konstruktorów	410
8.6. Konstruktory domyślne i bezargumentowe	416
8.7. Uwagi dotyczące metod dostępowych	416
8.8. Kompozycja	418
8.9. Typ <code>enum</code>	420
8.10. Mechanizm odświeżania pamięci	423
8.11. Składowe statyczne klasy	424
8.12. Import statyczny	428
8.13. Zmienne instancji typu <code>final</code>	429
8.14. Dostęp na poziomie pakietu	430
8.15. Użycie klasy <code>BigDecimal</code> do precyzyjnych obliczeń pieniężnych	431
8.16. (Opcjonalnie) Studium przypadku GUI i grafiki — użycie obiektów z grafiką	434
8.17. Podsumowanie	438
<b>9. Programowanie obiektowe — dziedziczenie</b>	<b>447</b>
9.1. Wprowadzenie	448
9.2. Klasy nadrzędne i podklasy	449
9.3. Składowe chronione	451
9.4. Związek między klasami nadrzędnymi i podklasami	452
9.4.1. Tworzenie i wykorzystywanie klasy <code>CommissionEmployee</code>	453
9.4.2. Tworzenie i użycie klasy <code>BasePlusCommissionEmployee</code>	458
9.4.3. Tworzenie hierarchii dziedziczenia <code>CommissionEmployee</code> przez <code>BasePlusCommissionEmployee</code>	462
9.4.4. Tworzenie hierarchii dziedziczenia <code>CommissionEmployee</code> przez <code>BasePlusCommissionEmployee</code> używającej zmiennych chronionych	465
9.4.5. Tworzenie hierarchii dziedziczenia <code>CommissionEmployee</code> przez <code>BasePlusCommissionEmployee</code> używającej zmiennych prywatnych	468
9.5. Konstruktory w podklasach	473

9.6. Klasa Object	473
9.7. Projektowanie klas — kompozycja kontra dziedziczenie	474
9.8. Podsumowanie	476

## 10. Programowanie obiektowe — polimorfizm i interfejsy **483**

10.1. Wprowadzenie	484
10.2. Przykłady polimorfizmu	486
10.3. Przykład zachowania polimorficznego	487
10.4. Metody i klasy abstrakcyjne	490
10.5. Studium przypadku — system płac wykorzystujący polimorfizm	492
10.5.1. Abstrakcyjna klasa nadrzędna Employee	495
10.5.2. Konkretna podklasa SalariedEmployee	496
10.5.3. Konkretna podklasa HourlyEmployee	498
10.5.4. Konkretna podklasa CommissionEmployee	500
10.5.5. Pośrednia konkretna podklasa BasePlusCommissionEmployee	501
10.5.6. Przetwarzanie polimorficzne, operator instanceof i rzutowanie w dół	503
10.6. Umożliwienie przypisywania między zmiennymi klas nadrzędnych i podklas	507
10.7. Metody i klasy finalne	508
10.8. Dokładniejszy opis problemów z wywoływaniem metod z konstruktorów	509
10.9. Tworzenie i stosowanie interfejsów	509
10.9.1. Tworzenie hierarchii Payable	512
10.9.2. Interfejs Payable	512
10.9.3. Klasa Invoice	513
10.9.4. Modyfikacja klasy Employee w celu implementacji interfejsu Payable	515
10.9.5. Użycie interfejsu Payable do polimorficznego przetwarzania klas Invoice i Employee	517
10.9.6. Wspólne interfejsy API Javy	518
10.10. Rozszerzenia interfejsów w Javie SE 8	519
10.10.1. Domyślne metody interfejsu	519
10.10.2. Statyczne metody interfejsu	520
10.10.3. Interfejsy funkcyjne	520
10.11. Prywatne metody interfejsów w Javie SE 9	521
10.12. Konstruktory prywatne	521
10.13. Programowanie do interfejsu, a nie do implementacji	522
10.13.1. Dziedziczenie implementacji działa najlepiej dla niewielkiej liczby ściśle powiązanych klas	522
10.13.2. Dziedziczenie interfejsów jest najlepsze dla elastyczności	522

10.13.3. Modyfikacja hierarchii Employee	523
10.14. Studium przypadku GUI i grafiki — rysowanie z użyciem polimorfizmu	524
10.15. Podsumowanie	526

## **11. Obsługa wyjątków — dokładniejsze spojrzenie 535**

11.1. Wprowadzenie	536
11.2. Przykład — dzielenie przez 0 bez obsługi wyjątków	537
11.3. Przykład — obsługa wyjątków ArithmeticException i InputMismatchException	540
11.4. Kiedy używać obsługi wyjątków	545
11.5. Hierarchia wyjątków Javy	546
11.6. Blok finally	549
11.7. Rozwijanie stosu i uzyskiwanie informacji z wyjątku	554
11.8. Wyjątki łańcuchowe	557
11.9. Deklarowanie nowych rodzajów wyjątków	559
11.10. Warunki wstępne i warunki końcowe	560
11.11. Asercje	560
11.12. Instrukcja try z zasobami — automatyczne zwalnianie zasobów	562
11.13. Podsumowanie	563

## **12. Graficzny interfejs użytkownika JavaFX, część 1. 569**

12.1. Wprowadzenie	570
12.2. Narzędzie Scene Builder dla JavaFX	571
12.3. Struktura okna aplikacji JavaFX	572
12.4. Aplikacja powitalna — wyświetlenie tekstu i obrazka	573
12.4.1. Uruchomienie narzędzia Scene Builder i utworzenie pliku Welcome.fxml	574
12.4.2. Dodanie obrazu do folderu zawierającego plik Welcome.fxml	574
12.4.3. Tworzenie kontenera układu VBox	575
12.4.4. Konfiguracja kontenera układu VBox	575
12.4.5. Dodanie i konfiguracja etykiety	575
12.4.6. Dodanie i konfiguracja ImageView	576
12.4.7. Podgląd wynikowego interfejsu aplikacji	577
12.5. Aplikacja do wyliczania napiwków — wprowadzenie do obsługi zdarzeń	577
12.5.1. Testowanie kalkulatora napiwków	579
12.5.2. Przedstawienie technologii	580
12.5.3. Budowanie GUI aplikacji	583
12.5.4. Klasa TipCalculator	589
12.5.5. Klasa TipCalculatorController	591

12.6. Funkcjonalności opisywane w pozostałych rozdziałach dotyczących JavaFX	596
12.7. Podsumowanie	597

## **13. Graficzny interfejs użytkownika JavaFX, część 2. 605**

13.1. Wprowadzenie	606
13.2. Układanie węzłów w grafie	606
13.3. Aplikacja Painter — przyciski opcji, zdarzenia myszy i kształty	608
13.3.1. Omówienie technologii	608
13.3.2. Utworzenie pliku Painter.fxml	611
13.3.3. Tworzenie GUI	611
13.3.4. Podklasa Painter klasy Application	614
13.3.5. Klasa PainterController	614
13.4. Aplikacja do wybierania kolorów	618
13.4.1. Omówienie technologii	618
13.4.2. Tworzenie GUI	620
13.4.3. Podklasa ColorChooser klasy Application	622
13.4.4. Klasa ColorChooserController	622
13.5. Aplikacja do przeglądania okładek	625
13.5.1. Omówienie technologii	625
13.5.2. Dodanie obrazków do folderu aplikacji	626
13.5.3. Budowanie interfejsu graficznego aplikacji	626
13.5.4. Podklasa CoverViewer klasy Application	627
13.5.5. Klasa CoverViewerController	628
13.6. Aplikacja do przeglądania okładek — dostosowanie komórek ListView	630
13.6.1. Omówienie technologii	630
13.6.2. Kopiowanie przeglądarki okładek	631
13.6.3. Klasa własnej fabryki komórek — ImageTextCell	631
13.6.4. Klasa CoverViewerController	633
13.7. Dodatkowe możliwości JavaFX	634
13.8. JavaFX 9 — aktualizacja JavaFX w Javie SE 9	636
13.9. Podsumowanie	638

## **14. Tekst, znaki i wyrażenia regularne 649**

14.1. Wprowadzenie	650
14.2. Podstawy znaków i tekstów	650
14.3. Klasa String	651
14.3.1. Konstruktory klasy String	651
14.3.2. Metody length, charAt i getChars	652
14.3.3. Porównywanie tekstów	653
14.3.4. Znajdowanie położenia znaków i fragmentów w tekstach	658

14.3.5.	Wydobywanie fragmentu tekstu	660
14.3.6.	Łączenie tekstów	661
14.3.7.	Inne metody klasy String	661
14.3.8.	Metoda valueOf klasy String	663
14.4.	Klasa StringBuilder	664
14.4.1.	Konstruktory StringBuilder	665
14.4.2.	Metody length, capacity, setLength i ensureCapacity klasy StringBuilder	665
14.4.3.	Metody charAt, setCharAt, getChars i reverse klasy StringBuilder	667
14.4.4.	Metody append klasy StringBuilder	668
14.4.5.	Metody wstawiania i usuwania klasy StringBuilder	670
14.5.	Klasa Character	671
14.6.	Tokenizacja tekstów	675
14.7.	Wyrażenia regularne, klasy Pattern i Matcher	676
14.7.1.	Zastępowanie fragmentów tekstu i podział tekstu	682
14.7.2.	Klasy Pattern i Matcher	683
14.8.	Podsumowanie	685

## 15. Pliki, strumienie wejścia – wyjścia, NIO i serializacja XML

697

15.1.	Wprowadzenie	698
15.2.	Pliki i strumienie	698
15.3.	Użycie klas i interfejsów NIO do pobrania informacji o pliku lub folderze	700
15.4.	Sekwencyjne pliki tekstowe	704
15.4.1.	Tworzenie sekwencyjnego pliku tekstowego	704
15.4.2.	Odczyt danych z sekwencyjnego pliku tekstowego	707
15.4.3.	Studium przypadku — program sprawdzający salda płatności klientów	709
15.4.4.	Aktualizacja plików sekwencyjnych	713
15.5.	Serializacja XML	713
15.5.1.	Tworzenie sekwencyjnego pliku używającego serializacji XML	714
15.5.2.	Odczyt i deserializacja danych z pliku sekwencyjnego	719
15.6.	Okna dialogowe FileChooser i DirectoryChooser	720
15.7.	(Opcjonalnie) Dodatkowe klasy java.io	726
15.7.1.	Interfejsy i klasy dotyczące wejścia – wyjścia danych bajtowych	727
15.7.2.	Interfejsy i klasy dla znakowych operacji wejścia – wyjścia	729
15.8.	Podsumowanie	729

<b>16. Ogólne kolekcje</b>	<b>739</b>
16.1. Wprowadzenie	740
16.2. Omówienie kolekcji	740
16.3. Klasy otoczkowe	742
16.4. Automatyczne pakowanie i rozpakowywanie	742
16.5. Interfejs Collection i klasa Collections	743
16.6. Listy	743
16.6.1. ArrayList i Iterator	744
16.6.2. Klasa LinkedList	747
16.7. Metody klasy Collections	751
16.7.1. Metoda sort	752
16.7.2. Metoda shuffle	755
16.7.3. Metody reverse, fill, copy, max i min	757
16.7.4. Metoda binarySearch	759
16.7.5. Metody addAll, frequency i disjoint	761
16.8. Klasa PriorityQueue i interfejs Queue	763
16.9. Zbiory	764
16.10. Odwzorowania	767
16.11. Synchronizowane kolekcje	771
16.12. Kolekcje niezienne	772
16.13. Implementacje abstrakcyjne	772
16.14. Java SE 9 — wygodne metody fabryczne dla niezmiennych kolekcji	773
16.15. Podsumowanie	776
<b>17. Lambdy i strumienie</b>	<b>783</b>
17.1. Wprowadzenie	784
17.2. Strumienie i redukcja	786
17.2.1. Sumowanie liczb od 1 do 10 pętlą for	786
17.2.2. Iteracja zewnętrzna za pomocą pętli for jest narażona na błędy	787
17.2.3. Sumowanie za pomocą strumienia i redukcji	787
17.2.4. Iteracja wewnętrzna	789
17.3. Odwzorowania i lambdy	789
17.3.1. Wyrażenia lambda	790
17.3.2. Składnia lambda	791
17.3.3. Operacje pośrednie i operacje kończące	792
17.4. Filtrowanie	793
17.5. Jak elementy poruszają się po potoku strumienia?	795
17.6. Referencje do metod	796
17.6.1. Tworzenie strumienia IntStream liczb losowych	797
17.6.2. Wykonywanie zadań dla każdego elementu strumienia za pomocą metody forEach i referencji do metody	797

17.6.3.	Odwzorowanie liczb całkowitych na obiekty String za pomocą metody mapToObj	798
17.6.4.	Łączenie tekstów metodą collect	799
17.7.	Operacje IntStream	799
17.7.1.	Tworzenie obiektu IntStream i wyświetlanie jego wartości	800
17.7.2.	Operacje kończące count, min, max, sum i average	801
17.7.3.	Operacja kończąca reduce	802
17.7.4.	Sortowanie wartości IntStream	804
17.8.	Interfejsy funkcyjne	804
17.9.	Lambdy — dokładniejsze spojrzenie	806
17.10.	Obsługa Stream<Integer>	807
17.10.1.	Tworzenie Stream<Integer>	808
17.10.2.	Sortowanie strumienia i zbieranie wyników	808
17.10.3.	Filtrowanie strumienia i przechowywanie wyników w celu ich późniejszego użycia	809
17.10.4.	Filtrowanie i sortowanie strumienia i zbieranie wyników	809
17.10.5.	Sortowanie zebranych wyników	810
17.11.	Obsługa Stream<String>	810
17.11.1.	Mapowanie tekstów na duże litery	811
17.11.2.	Filtrowanie tekstów i sortowanie ich rosnąco bez uwzględniania wielkości liter	812
17.11.3.	Filtrowanie tekstów i sortowanie ich malejąco bez uwzględniania wielkości liter	812
17.12.	Obsługa Stream<Employee>	812
17.12.1.	Tworzenie i wyświetlenie List<Employee>	814
17.12.2.	Odfiltrowywanie pracowników z wynagrodzeniem mieszczącym się w określonym przedziale	815
17.12.3.	Sortowanie pracowników na podstawie wielu pól	818
17.12.4.	Odwzorowanie pracowników na unikatowe nazwiska	819
17.12.5.	Grupowanie pracowników według działów	821
17.12.6.	Zliczenie pracowników poszczególnych działów	822
17.12.7.	Suma i średnia zarobków pracowników	822
17.13.	Utworzenie Stream<String> na podstawie pliku	823
17.14.	Strumienie wartości losowych	827
17.15.	Strumienie nieskończone	829
17.16.	Obsługa zdarzeń za pomocą lambda	831
17.17.	Dodatkowe uwagi na temat interfejsów Javy SE 8	831
17.18.	Podsumowanie	832

## **18. Rekurencja** **849**

18.1.	Wprowadzenie	850
18.2.	Pojęcie rekurencji	851
18.3.	Przykład użycia rekurencji — silnia	852

18.4.	Implementacja FactorialCalculator z użyciem klasy BigInteger	855
18.5.	Przykład użycia rekurencji — ciąg Fibonacciego	856
18.6.	Rekurencja i stos wywołań metod	859
18.7.	Rekurencja kontra iteracja	860
18.8.	Wieża Hanoi	862
18.9.	Fraktale	864
18.9.1.	Krzywa Kocha	865
18.9.2.	(Opcjonalnie) Studium przypadku — pióro Lo	866
18.9.3.	(Opcjonalnie) Interfejs graficzny rysujący fraktal	868
18.9.4.	(Opcjonalnie) Klasa FractalController	870
18.10.	Rekurencyjne nawracanie	875
18.11.	Podsumowanie	876

## 19. Wyszukiwanie, sortowanie i notacja dużego O 885

19.1.	Wprowadzenie	886
19.2.	Wyszukiwanie liniowe	887
19.3.	Notacja dużego O	890
19.3.1.	Algorytmy $O(1)$	890
19.3.2.	Algorytmy $O(n)$	890
19.3.3.	Algorytmy $O(n^2)$	891
19.3.4.	Duże O dla wyszukiwania liniowego	891
19.4.	Wyszukiwanie binarne	892
19.4.1.	Implementacja wyszukiwania binarnego	893
19.4.2.	Wydajność wyszukiwania binarnego	896
19.5.	Algorytmy sortujące	896
19.6.	Sortowanie przez wybieranie	897
19.6.1.	Implementacja sortowania przez wybieranie	897
19.6.2.	Wydajność sortowania przez wybieranie	900
19.7.	Sortowanie przez wstawianie	900
19.7.1.	Implementacja sortowania przez wstawianie	901
19.7.2.	Wydajność sortowania przez wstawianie	903
19.8.	Sortowanie przez scalanie	903
19.8.1.	Implementacja algorytmu sortowania przez scalanie	904
19.8.2.	Wydajność sortowania przez scalanie	908
19.9.	Podsumowanie dużego O algorytmów wyszukiwania i sortowania z tego rozdziału	908
19.10.	Duże zrównoleglenie i algorytmy równoległe	908
19.11.	Podsumowanie	909



<b>20. Uogólnione klasy i metody</b>	
<b>— dokładniejsze spojrzenie</b>	<b>917</b>
20.1. Wprowadzenie	918
20.2. Po co nam metody uogólnione?	918
20.3. Metody uogólnione — implementacja i przekształcenia na etapie kompilacji	920
20.4. Dodatkowy problem przekształcenia w trakcie kompilacji	923
20.5. Przeciążanie metod uogólnionych	927
20.6. Klasy uogólnione	927
20.7. Wieloznaczność w metodach akceptujących parametry typu	934
20.8. Podsumowanie	938
<b>21. Własne uogólnione struktury danych</b>	<b>943</b>
21.1. Wprowadzenie	944
21.2. Klasy odnoszące się do samych siebie	945
21.3. Dynamiczna alokacja pamięci	946
21.4. Listy jednokierunkowe	946
21.4.1. Lista jednokierunkowa	947
21.4.2. Implementacja uogólnionej klasy List	947
21.4.3. Klasy uogólnione ListNode i List	950
21.4.4. Klasa ListTest	950
21.4.5. Metoda insertAtFront klasy List	952
21.4.6. Metoda insertAtBack klasy List	952
21.4.7. Metoda removeFromFront klasy List	954
21.4.8. Metoda removeFromBack klasy List	954
21.4.9. Metoda print klasy List	956
21.4.10. Tworzenie własnych pakietów	956
21.5. Stosy	960
21.6. Kolejki	963
21.7. Drzewa	966
21.8. Podsumowanie	972
Podrozdział specjalny — tworzenie własnego kompilatora	984
<b>22. Grafika, animacje i wideo w JavaFX</b>	<b>997</b>
22.1. Wprowadzenie	998
22.2. Sterowanie czcionkami poprzez kaskadowe arkusze stylów	999
22.2.1. CSS, który określa styl GUI	999
22.2.2. FXML, który definiuje GUI — wprowadzenie do XML	1002
22.2.3. Referencja do pliku CSS w dokumencie FXML	1005
22.2.4. Określenie klasy dla stylów w VBox	1006
22.2.5. Programowe wczytywanie CSS	1006

22.3.	Wyświetlanie kształtów dwuwymiarowych	1006
22.3.1.	Definiowanie dwuwymiarowych kształtów za pomocą FXML	1007
22.3.2.	CSS, który zapewnia stylowanie dwuwymiarowych kształtów	1009
22.4.	Klasy Polyline, Polygon i Path	1012
22.4.1.	GUI i CSS	1012
22.4.2.	Klasa PolyShapesController	1014
22.5.	Przekształcenia	1017
22.6.	Odtwarzanie materiału wideo	1019
22.6.1.	Interfejs graficzny aplikacji	1020
22.6.2.	Klasa VideoPlayerController	1022
22.7.	Animacje Transition	1026
22.7.1.	Plik TransitionAnimations.fxml	1026
22.7.2.	Klasa TransitionAnimationsController	1028
22.8.	Animacje Timeline	1031
22.9.	Animacja klatka po klatce za pomocą klasy AnimationTimer	1034
22.10.	Rysowanie na kanwie	1037
22.11.	Kształty trójwymiarowe	1042
22.12.	Podsumowanie	1045

## 23. Współbieżność 1063

23.1.	Wprowadzenie	1064
23.2.	Stany i cykl życia wątku	1066
23.2.1.	Stan nowy i stan działający	1066
23.2.2.	Stan oczekujący	1067
23.2.3.	Stan oczekujący czasowo	1067
23.2.4.	Stan zablokowany	1068
23.2.5.	Stan zakończony	1068
23.2.6.	Stan działający z punktu widzenia systemu operacyjnego	1068
23.2.7.	Priorytety i harmonogramowanie wątków	1068
23.2.8.	Odsuwanie wykonania w nieskończoność i blokady wzajemne	1069
23.3.	Tworzenie i wykonywanie wątków za pomocą frameworku Executor	1070
23.4.	Synchronizacja wątków	1074
23.4.1.	Dane niezmiennie	1074
23.4.2.	Monitory	1075
23.4.3.	Współdzielenie modyfikowalnych danych bez synchronizacji	1076
23.4.4.	Synchronizowany dostęp do współdzielonych i modyfikowanych danych — operacje niepodzielne	1080

23.5.	Związek typu producent – konsument bez synchronizacji	1083
23.6.	Związek typu producent – konsument — klasa ArrayBlockingQueue	1091
23.7.	Związek typu producent – konsument	1094
23.8.	Związek typu producent – konsument — bufor o określonym rozmiarze	1101
23.9.	Związek typu producent – konsument — interfejsy Lock i Condition	1109
23.10.	Kolekcje współbieżne	1116
23.11.	Wielowątkowość w JavaFX	1117
23.11.1.	Wykonywanie obliczeń w wątku roboczym — ciąg Fibonacciego	1119
23.11.2.	Przetwarzanie wyników pośrednich — sito Eratostenesa	1125
23.12.	Pomiar czasu działania metod sort i parallelSort	1130
23.13.	Java SE 8 — strumienie szeregowy i równoległe	1133
23.14.	(Dla zaawansowanych) Interfejsy Callable i Future	1135
23.15.	(Dla zaawansowanych) Framework Fork/Join	1140
23.16.	Podsumowanie	1140

## 24. Dostęp do baz danych poprzez JDBC 1155

24.1.	Wprowadzenie	1156
24.2.	Relacyjne bazy danych	1157
24.3.	Baza danych books	1158
24.4.	Język SQL	1162
24.4.1.	Podstawowe zapytanie SELECT	1162
24.4.2.	Klauzula WHERE	1163
24.4.3.	Klauzula ORDER BY	1165
24.4.4.	Łączenie danych z wielu tabel — operator INNER JOIN	1166
24.4.5.	Instrukcja INSERT	1168
24.4.6.	Instrukcja UPDATE	1169
24.4.7.	Instrukcja DELETE	1170
24.5.	Konfiguracja bazy danych Java DB	1171
24.5.1.	Tworzenie baz danych dla rozdziału w systemie Windows	1171
24.5.2.	Tworzenie baz danych dla rozdziału w systemie macOS	1172
24.5.3.	Tworzenie baz danych dla rozdziału w systemie Linux	1173
24.6.	Połączenie się z bazą danych i wykonanie zapytania	1173
24.6.1.	Automatyczne odnajdowanie sterowników	1175
24.6.2.	Połączenie z bazą danych	1175
24.6.3.	Utworzenie obiektu Statement do wykonywania poleceń SQL	1176

24.6.4. Wykonanie zapytania	1176
24.6.5. Przetwarzanie wyników zapytania	1176
24.7. Odpytywanie bazy danych books	1178
24.7.1. Klasa ResultSetTableModel	1178
24.7.2. Wyświetlenie interfejsu graficznego aplikacji	1185
24.7.3. Klasa DisplayQueryResultsController	1185
24.8. Interfejs RowSet	1190
24.9. Klasa PreparedStatement	1193
24.9.1. Aplikacja do przechowywania kontaktów używająca obiektów PreparedStatement	1194
24.9.2. Klasa Person	1194
24.9.3. Klasa PersonQueries	1196
24.9.4. Interfejs graficzny książki adresowej	1199
24.9.5. Klasa AddressBookController	1200
24.10. Procedury składowane	1204
24.11. Przetwarzanie transakcyjne	1205
24.12. Podsumowanie	1206

## 25. Wprowadzenie do JShell — interaktywnej konsoli Javy 9 1215

25.1. Wprowadzenie	1216
25.2. Instalacja JDK 9	1218
25.3. Wprowadzenie do JShell	1218
25.3.1. Uruchamianie sesji JShell	1219
25.3.2. Wykonywanie poleceń	1219
25.3.3. Jawne deklarowanie zmiennych	1221
25.3.4. Wylistowanie i wykonanie wcześniejszych fragmentów	1223
25.3.5. Wyliczanie wyrażeń i niejawne deklarowanie zmiennych	1224
25.3.6. Wykorzystywanie niejawnie zadeklarowanych zmiennych	1225
25.3.7. Sprawdzanie zawartości zmiennej	1225
25.3.8. Czyszczenie sesji JShell	1225
25.3.9. Instrukcje wielowierszowe	1226
25.3.10. Edycja fragmentów kodu	1226
25.3.11. Wyjście z JShell	1229
25.4. Wejście danych z konsoli w JShell	1229
25.5. Deklarowanie i stosowanie klas	1231
25.5.1. Tworzenie klasy w JShell	1231
25.5.2. Jawne deklarowanie zmiennych będących referencjami	1232
25.5.3. Tworzenie obiektów	1232
25.5.4. Modyfikowanie obiektów	1233
25.5.5. Tworzenie własnej nazwy zmiennej dla wyrażenia	1233
25.5.6. Zapisywanie i otwieranie plików z fragmentami kodu	1234

25.6.	Automatyczne uzupełnianie	1234
25.6.1.	Automatyczne uzupełnianie identyfikatorów	1235
25.6.2.	Automatyczne uzupełnianie poleceń JShell	1236
25.7.	Przeglądanie składowych klas i dokumentacji	1236
25.7.1.	Wyświetlanie składowych statycznych klasy Math	1237
25.7.2.	Przeglądanie parametrów metod	1238
25.7.3.	Przeglądanie dokumentacji metody	1238
25.7.4.	Przeglądanie dokumentacji pól publicznych	1238
25.7.5.	Przeglądanie dokumentacji klasy	1239
25.7.6.	Przeglądanie przeciążeń metod	1240
25.7.7.	Odkrywanie składowych konkretnego obiektu	1240
25.8.	Deklarowanie metod	1242
25.8.1.	Użycie w przód niezadeklarowanej metody — deklaracja metody displayCubes	1242
25.8.2.	Deklarowanie wcześniej niezadeklarowanej metody	1243
25.8.3.	Testowanie metody cube i zmiana jej deklaracji	1243
25.8.4.	Testowanie uaktualnionej metody cube i metody displayCubes	1244
25.9.	Wyjątki	1244
25.10.	Import klas i dodawanie pakietów do CLASSPATH	1245
25.11.	Korzystanie z zewnętrznego edytora	1247
25.12.	Podsumowanie poleceń JShell	1250
25.12.1.	Uzyskiwanie pomocy	1251
25.12.2.	Polecenie /edit — dodatkowe funkcjonalności pomocy	1251
25.12.3.	Polecenie /reload	1251
25.12.4.	Polecenie /drop	1252
25.12.5.	Tryby informacji zwrotnej	1253
25.12.6.	Inne opcje JShell konfigurowalne za pomocą /set	1254
25.13.	Skróty klawiaturowe w edycji fragmentów	1255
25.14.	W jaki sposób JShell modyfikuje Javę w celu jej interaktywnego użycia?	1256
25.15.	Obsługa JShell w IDE	1256
25.16.	Podsumowanie	1256

## 26. Sieć

1273 (online)

26.1.	Wprowadzenie	1274
26.2.	Odczyt pliku z serwera WWW	1275
26.3.	Wykonanie prostego serwera przy użyciu gniazd strumieniowych	1278
26.4.	Wykonanie prostego klienta przy użyciu gniazd strumieniowych	1280
26.5.	Interakcja klienta i serwera wykorzystująca gniazda strumieniowe	1281
26.6.	Datagramy — bezpołączeniowa interakcja między klientem i serwerem	1293

26.7.	Kółko i krzyżyk w wersji klient – serwer z serwerem wielowątkowym	1301
26.8.	Opcjonalne studium przypadku — aplikacja DeitelMessenger	1315
26.9.	Podsumowanie	1315

## 27. System modułów platformy Java 1321 (online)

27.1.	Wprowadzenie	1323
27.2.	Deklaracja modułu	1328
27.2.1.	Dyrektywa requires	1328
27.2.2.	Dyrektywa requires transitive — niejawne czytanie	1328
27.2.3.	Dyrektywy exports i exports...to	1329
27.2.4.	Dyrektywa uses	1329
27.2.5.	Dyrektywa provides...with	1329
27.2.6.	Modyfikator open oraz dyrektywy opens i opens...to	1329
27.2.7.	Ograniczone słowa kluczowe	1331
27.3.	Modułowa wersja aplikacji powitalnej	1331
27.3.1.	Struktura aplikacji	1331
27.3.2.	Klasa Welcome	1335
27.3.3.	Plik module-info.java	1335
27.3.4.	Graf zależności modułu	1335
27.3.5.	Kompilacja modułu	1337
27.3.6.	Uruchamianie aplikacji z poziomu rozbitego folderu aplikacji	1338
27.3.7.	Spakowanie modułu do pliku JAR	1338
27.3.8.	Uruchamianie aplikacji z modułowego pliku JAR	1339
27.3.9.	Dodatkowa uwaga — ścieżka klas a ścieżka modułów	1339
27.4.	Tworzenie i użycie własnych modułów	1340
27.4.1.	Eksport pakietu w celu użycia w innych modułach	1341
27.4.2.	Wykorzystanie klasy pakietu w innym module	1342
27.4.3.	Kompilacja i uruchomienie przykładu	1344
27.4.4.	Zapakowanie aplikacji do modułowych plików JAR	1345
27.4.5.	Silna enkapsulacja i dostępność	1345
27.5.	Graf zależności modułu — dokładniejsze spojrzenie	1346
27.5.1.	Moduł java.sql	1346
27.5.2.	Moduł java.se	1347
27.5.3.	Wyświetlenie grafu zależności modułów JDK	1347
27.5.4.	Błąd — graf modułu z cyklem	1347
27.6.	Migracja kodu do Javy 9	1350
27.6.1.	Moduł nienazwany	1351
27.6.2.	Moduły automatyczne	1351
27.6.3.	Narzędzie jdeps — analiza zależności	1352
27.7.	Zasoby w modułach — wykorzystanie modułu automatycznego	1354
27.7.1.	Moduły automatyczne	1355
27.7.2.	Wymaganie kilku modułów	1356
27.7.3.	Otwarcie modułu na potrzeby mechanizmu refleksji	1356
27.7.4.	Graf zależności modułu	1356

27.7.5. Kompilacja modułu	1357
27.7.6. Uruchomienie aplikacji po modularyzacji	1357
27.8. Tworzenie własnych systemów wykonawczych narzędziem <code>link</code>	1358
27.8.1. Wyświetlenie listy modułów JRE	1358
27.8.2. Własny system wykonawczy zawierający tylko moduł <code>java.base</code>	1359
27.8.3. Tworzenie własnego systemu wykonawczego dla aplikacji powitalnej	1361
27.8.4. Wykonywanie aplikacji powitalnej we własnym systemie wykonawczym	1362
27.8.5. Użycie mechanizmu rozwiązywania modułów z własnym systemem wykonawczym	1362
27.9. Usługi i klasa <code>ServiceLoader</code>	1363
27.9.1. Interfejs dostawcy usług	1365
27.9.2. Wczytywanie i użycie dostawców usług	1366
27.9.3. Dyrektywa <code>uses</code> modułu i konsumpcja usług	1368
27.9.4. Uruchomienie aplikacji bez dostawców usług	1369
27.9.5. Implementacja dostawcy usług	1369
27.9.6. Dyrektywa <code>provides...with</code> modułu i deklaracja dostawcy usług	1370
27.9.7. Uruchomienie aplikacji z jednym dostawcą usług	1370
27.9.8. Implementacja drugiego dostawcy usług	1371
27.9.9. Uruchomienie aplikacji z dwoma dostawcami usług	1372
27.10. Podsumowanie	1372

## 28. Pozostałe tematy związane z Javą 9 1375 (online)

28.1. Wprowadzenie	1376
28.2. Przypomnienie — funkcjonalności Javy 9 omówione w poprzednich rozdziałach	1377
28.3. Nowa wersja formatu tekstowego	1378
28.4. Wyrażenia regularne — nowe metody klasy <code>Matcher</code>	1378
28.4.1. Metody <code>appendReplacement</code> i <code>appendTail</code>	1380
28.4.2. Metody <code>replaceFirst</code> i <code>replaceAll</code>	1380
28.4.3. Metoda <code>results</code>	1380
28.5. Nowe metody interfejsu <code>Stream</code>	1381
28.5.1. Metody <code>takeWhile</code> i <code>dropWhile</code> strumienia	1382
28.5.2. Metoda <code>iterate</code> interfejsu <code>Stream</code>	1383
28.5.3. Metoda <code>ofNullable</code> interfejsu <code>Stream</code>	1383
28.6. Moduły w <code>JSHELL</code>	1384
28.7. API skórek dostępne w <code>JavaFX 9</code>	1385
28.8. Inne usprawnienia związane z interfejsem graficznym i grafiką	1386
28.8.1. Obrazy o wielu rozdzielczościach	1386
28.8.2. Obsługa obrazów <code>TIFF</code>	1386
28.8.3. Funkcjonalności pulpitu zależne od platformy	1386

28.9.	Tematy związane z bezpieczeństwem Javy 9	1387
28.9.1.	Filtrowanie nadchodzących danych serializowanych	1387
28.9.2.	Domyślne tworzenie magazynów kluczy PKCS12	1387
28.9.3.	Obsługa protokołu DTLS (Datagram Transport Layer Security)	1387
28.9.4.	Obsługa walidacji OCSP dla TLS	1388
28.9.5.	Rozszerzenie umożliwiające negocjację protokołu warstwy aplikacyjnej w TLS	1388
28.10.	Inne tematy związane z Javą 9	1388
28.10.1.	Usprawnienie łączenia tekstów	1388
28.10.2.	Obsługa usług i API logów na poziomie platformy	1388
28.10.3.	Aktualizacja API procesów	1389
28.10.4.	Podpowiedzi dotyczące oczekiwania	1389
28.10.5.	Obsługa paczek zasobów z kodowaniem UTF-8	1389
28.10.6.	Domyślne korzystanie z danych CLDR	1389
28.10.7.	Usunięcie ostrzeżeń o wycofaniu z instrukcji importu	1390
28.10.8.	Wielowydaniowe pliki JAR	1390
28.10.9.	Unicode 8	1390
28.10.10.	Rozbudowa obsługi współbieżności	1390
28.11.	Elementy usunięte z JDK i Javy 9	1391
28.12.	Elementy zaproponowane do usunięcia w przyszłych wersjach Javy	1392
28.12.1.	Ulepszone wycofywanie	1392
28.12.2.	Elementy, które prawdopodobnie zostaną usunięte w przyszłych wydaniach Javy	1392
28.12.3.	Znajdowanie wycofywanych funkcjonalności	1393
28.12.4.	Aplety Javy	1393
28.13.	Podsumowanie	1391

**A Tabela kolejności wykonywania operatorów 1395****B Zbiór znaków ASCII 1397****C Słowa kluczowe i słowa zarezerwowane 1399****D Typy podstawowe 1401**



<b>E Korzystanie z debuggera</b>	<b>1401</b>
E.1. Wprowadzenie	1402
E.2. Punkty wstrzymania oraz polecenia run, stop, cont i print	1402
E.3. Polecenia print i set	1406
E.4. Sterowanie wykonywaniem za pomocą poleceń step, step up i next	1408
E.5. Polecenie watch	1410
E.6. Polecenie clear	1412
E.7. Podsumowanie	1415
<b>Skorowidz</b>	<b>1417</b>



# Wprowadzenie do aplikacji Javy, wejścia – wyjścia i operatorów

# 2



## Cele

W tym rozdziale:

- Proste aplikacje Javy
- Instrukcje wejścia i wyjścia
- Typy podstawowe Javy
- Podstawowe koncepcje dotyczące pamięci
- Operatory arytmetyczne
- Kolejność wykonywania działań arytmetycznych
- Instrukcje podejmujące decyzje
- Operatory relacji i równości

- 2.1. Wprowadzenie
- 2.2. Twój pierwszy program
  - wyświetlenie wiersza tekstu
  - 2.2.1. Kompilacja aplikacji
  - 2.2.2. Wykonywanie aplikacji
- 2.3. Modyfikacja pierwszego programu
- 2.4. Wyświetlanie tekstu metodą `printf`
- 2.5. Inna aplikacja — dodawanie liczb całkowitych
  - 2.5.1. Deklaracja `import`
  - 2.5.2. Deklaracja i utworzenie obiektu `Scanner` w celu pobrania danych z klawiatury
  - 2.5.3. Prośba o wprowadzenie danych
  - 2.5.4. Deklaracja zmiennej dla liczby całkowitej i pobranie wartości z klawiatury
  - 2.5.5. Pobranie drugiej liczby całkowitej
  - 2.5.6. Użycie zmiennych w obliczeniach
  - 2.5.7. Wyświetlenie wyniku obliczeń
  - 2.5.8. Dokumentacja API Javy
  - 2.5.9. Deklaracja i inicjalizacja zmiennej jako osobne instrukcje
- 2.6. Zagadnienia dotyczące pamięci
- 2.7. Operacje arytmetyczne
- 2.8. Podejmowanie decyzji — operatory równości i relacji
- 2.9. Podsumowanie

Streszczenie | Ćwiczenia do samooceny | Odpowiedzi do samooceny | Ćwiczenia | Uczynić świat lepszym

## 2.1. Wprowadzenie

Ten rozdział stanowi wprowadzenie do programowania w języku Java. Zaczynamy od przykładów, które wyświetlą na ekranie prosty tekst. Następnie przejdziemy do programu, który pobiera od użytkownika dwie wartości, wylicza ich sumę i wyświetla wynik. Dowiesz się, jak poinstruować komputer, aby wykonywał operacje arytmetyczne i zapisywał wyniki w celu późniejszego ich użycia. Ostatni przykład pokaże, jak podejmować decyzje. Aplikacja porówna dwie liczby i wyświetli komunikat informujący o wyniku porównania. Do kompilacji i uruchamiania programów użyjemy narzędzi wiersza poleceń z JDK. Jeśli chcesz skorzystać z IDE, materiały wprowadzające znajdziesz pod adresem:

<http://www.deitel.com/books/jhpt11>

Zawarte są tam informacje o użyciu trzech najpopularniejszych IDE: Eclipse, NetBeans i IntelliJ IDEA.

## 2.2. Twój pierwszy program — wyświetlenie wiersza tekstu

Aplikacja Javy to program komputerowy, który wykona pewne działania, gdy użyjemy polecenia `java` do uruchomienia maszyny wirtualnej Javy. Punkty 2.2.1 i 2.2.2 zawierają informacje na temat kompilacji i uruchomienia aplikacji Javy. Najpierw zajmijmy się bardzo prostą aplikacją, która wyświetla jeden wiersz tekstu. Rysunek 2.1 przedstawia kod programu i wynik jego działania.

Będziemy wykorzystywać numery wierszy, aby ułatwić odniesienie się do konkretnego wiersza kodu — **nie** stanowią one części programu Javy. Ten prosty przykład ilustruje kilka istotnych elementów Javy. Wiersz 7. wykonuje całe zadanie — wyświetla na ekranie tekst `Witamy programistę Javy!`.

### *Umieszczanie komentarzy w programach*

Wstawiamy komentarze w kodzie programu, aby go udokumentować i poprawić czytelność. Kompilator Javy **ignoruje** komentarze, więc **nie** powodują one żadnej akcji ze strony komputera wykonującego program.

```

1 // Rysunek 2.1. Welcome1.java
2 // Program wyświetlający tekst
3
4 public class Welcome1 {
5     // Metoda main rozpoczyna wykonywanie aplikacji Javy
6     public static void main(String[] args) {
7         System.out.println("Witamy programistę Javy!");
8     } // Koniec metody main
9 } // Koniec klasy Welcome1

```

```
Witamy programistę Javy!
```

**Rysunek 2.1.** Program wyświetlający tekst

Stosujemy konwencję, w której każdy program zaczynamy od komentarza wskazującego numer rysunku i nazwę pliku z programem. Komentarz z pierwszego wiersza to:

```
// Rysunek 2.1. Welcome1.java
```

Zaczyna się od znaków `//`, co wskazuje na **komentarz jednowierszowy**, który kończy się wraz z końcem wiersza zawierającego znaki `//`. Komentarz jednowierszowy nie musi zaczynać się od początku wiersza, lecz może zaczynać się w jego środku, jak ma to miejsce w wierszach 5., 8. i 9. Wiersz 2.:

```
// Program wyświetlający tekst
```

to dalsza część stosowanej konwencji. Wyjaśniamy tutaj działanie programu.

Java wykorzystuje też **komentarze tradycyjne**, które swym zasięgiem mogą obejmować kilka wierszy kodu:

```
/* To komentarz tradycyjny,
   który jest podzielony na kilka wierszy */
```

Taki komentarz rozpoczyna się od znaków `/*` i kończy znakami `*/`. Kompilator ignoruje cały tekst znajdujący się między tymi znakami. Oba przedstawione rodzaje komentarzy Java odziedziczyła po językach C (komentarze tradycyjne) i C++ (komentarze jednowierszowe).

Java obsługuje również komentarze trzeciego typu — **komentarze Javadoc**. Są umieszczane między znakami `/**` i `*/`. Kompilator ignoruje cały tekst między nimi. Komentarze Javadoc umożliwiają osadzenie dokumentacji programu bezpośrednio w kodzie źródłowym. Stanowią one zalecany sposób dokumentowania kodu dla innych programistów. **Program narzędziowy javadoc** (część JDK) odczytuje komentarze Javadoc i na ich podstawie generuje dokumentację w formacie stron HTML5. Aby oszczędzić miejsce, stosujemy w przykładach komentarze ze znakami `//` zamiast komentarzy tradycyjnych lub Javadoc.

### Wykorzystanie pustych wierszy

Puste wiersze (takie jak wiersz 3.) oraz znaki spacji i tabulacji zwiększają czytelność programu. Wszystkie te elementy nazywa się zbiorczo **białymi spacjami**. Kompilator ignoruje białe spacje.



### 2.1. Typowy błąd programistyczny

Brak jednego z elementów definiujących komentarz tradycyjny lub Javadoc spowoduje zgłoszenie błędu składniowego. Błąd składniowy to błąd zgłaszany przez kompilator, gdy napotka on fragment kodu łamiący zasady języka Java (np. jego składnię). Zasady składni Javy przypominają zasady gramatyczne języka naturalnego, np. języka polskiego, francuskiego czy angielskiego. Błędy składniowe nazywane są również **błędami kompilacji** lub **błędami kompilatora**, ponieważ kompilator wykrywa je w trakcie kompilacji programu. Gdy zostanie odkryty błąd składniowy, kompilator wyświetla stosowny komunikat. Aby pomyślnie skompilować program, musisz usunąć wszystkie błędy składniowe.



### 2.1. Dobra praktyka programistyczna

Niektóre firmy i organizacje wymagają, aby każdy program rozpoczynał się od komentarza wskazującego cel powstania pliku, jego autora, a także datę i czas ostatniej modyfikacji.



### 2.2. Dobra praktyka programistyczna

Korzystaj z białych spacji, aby zwiększyć czytelność kodu.

## Deklaracja klasy

Wiersz 4.:

```
public class Welcome1 {
```

rozpoczyna się od **deklaracji klasy** o nazwie `Welcome1`. Każdy program Javy składa się z przynajmniej jednej klasy zdefiniowanej przez programistę. Słowo kluczowe **class** informuje o rozpoczęciu deklaracji klasy, a tuż za nim pojawia się **nazwa klasy** (`Welcome1`). Słowa kluczowe w Javie to słowa zarezerwowane przez język i są zawsze pisane małymi literami. Kompletną listę wszystkich słów kluczowych Javy znajdziesz w dodatku C.

W rozdziałach od 2. do 7. wszystkie definiowanie klasy będą rozpoczynały się słowem kluczowym **public**. Na razie po prostu go potrzebujemy. Więcej informacji na temat klas publicznych (`public`) i niepublicznych znajdziesz w rozdziale 8.

## Nazwa pliku dla klasy publicznej

Klasa publiczna **musi** być umieszczona w pliku o nazwie `NazwaKlasy.java`, więc klasa `Welcome1` znajduje się w pliku `Welcome1.java`.



### 2.2. Typowy błąd programistyczny

Jeśli nazwa pliku klasy publicznej nie jest równa nazwie klasy i rozszerzeniu `.java`, kompilator zgłosi błąd kompilacji. Pamiętaj, aby użyć właściwej wielkości liter.

## Nazwy klas i identyfikatory

Zgodnie z konwencją nazwy klas rozpoczyna się od wielkiej litery, a następnie stosuje wielką literę dla pierwszej litery każdego z następujących słów (np. `ToJestNazwaKlasy`). Nazwa klasy jest identyfikatorem — ciągiem znaków składającym się z liter, cyfr, podkreśleń (`_`) i znaków dolara (`$`), ale **nie** może zaczyna się od cyfry

ani zawierać znaków spacji. Oto kilka przykładów poprawnych identyfikatorów: `Witaj1`, `$wartosc`, `_wartosc`, `m_poleNumeru1` i `przycisk7`. Nazwa `7przycisk` **nie** jest poprawnym identyfikatorem, ponieważ rozpoczyna się od cyfry, a `pole wejsciowe` **nie** jest poprawnym identyfikatorem, bo zawiera znak spacji. Zazwyczaj identyfikator, który nie zaczyna się od wielkiej litery, nie jest nazwą klasy. W Javie istotna jest wielkość liter — małe i wielkie litery się różnią — więc `test` i `Test` to dwa różne (i całkowicie poprawne) identyfikatory.



### 2.3. Dobra praktyka programistyczna

*Stosuje się konwencję, w której każdy wyraz identyfikatora nazwy klasy pisany jest z wielkiej litery. Identyfikator klasy `DolarTestowy` zaczyna się od słowa `Dolar` z wielką literą `D` i stosuje drugie słowo `Testowy` z wielką literą `T`. Ten sposób nazywania nazywa się stylem wielbłądzim, ponieważ wielkie litery przypominają garby wielbłąda.*

### Podkreślenie (  ) w Javie 9

Od Javy 9 nie można stosować podkreślenia (  ) jako jedynego znaku definiującego identyfikator. 9

### Treść klasy

Lewy nawias klamrowy (na końcu wiersza 4.), `{`, rozpoczyna treść deklaracji klasy. Deklaracja klasy musi kończyć się odpowiadającym mu prawym nawiasem klamrowym (wiersz 9.), `}`. Wiersze od 5. do 8. mają wcięcie.



### 2.4. Dobra praktyka programistyczna

*Zastosuj jeden poziom wcięcia dla deklaracji każdej klasy dla kodu wewnątrz nawiasów klamrowych. W ten sposób podkreślisz strukturę deklaracji klasy i ułatwisz czytanie pliku. Wykorzystujemy trzy spacje jako formę wcięcia, choć wielu programistów preferuje dwie lub cztery spacje. Nieważne, co wybierzesz, ale swój wybór stosuj konsekwentnie.*



### 2.5. Dobra praktyka programistyczna

*IDE najczęściej automatycznie wykonują wcięcie kodu. Do stosowania wcięć służy też klawisz `Tab`. Można skonfigurować IDE, aby dodawało preferowaną liczbę spacji po każdym naciśnięciu klawisza `Tab`.*



### 2.3. Typowy błąd programistyczny

*Jeśli nawiasy nie są ułożone parami, zostanie zgłoszony błąd składniowy.*



### 2.1. Wskazówka zapobiegająca błędom

*Gdy użyjesz otwierającego nawiasu klamrowego, od razu wpisz też nawias zamykający, a dopiero później zacznij pisać treść wewnątrz nawiasów. Ta sztuczka pozwoli Ci uniknąć błędów związanych z niezamkniętymi nawiasami. Wiele IDE wykona ten krok za Ciebie.*

### Deklaracja metody

Wiersz 5.:

```
// Metoda main rozpoczyna wykonywanie aplikacji Javy
```

to komentarz wyjaśniający działanie wierszy od 6. do 8. Wiersz 6.:

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

to punkt początkowy każdej aplikacji Javy. Nawiasy po identyfikatorze `main` wskazują, że rozpoczyna się nowy blok, nazywany **metodą**. Deklaracja klasy Javy standardowo zawiera jedną lub więcej metod. W przypadku aplikacji Javy jedna z metod **musi** nosić nazwę `main` i musi być zdefiniowana w taki sam sposób jak w wierszu 6.

Metody wykonują zadania i mogą zwracać informacje, gdy je zakończą. Znaczenie słowa kluczowego `static` wyjaśnimy w punkcie 3.2.5. Słowo kluczowe **`void`** oznacza, że metoda **nie** zwróci żadnych informacji. W dalszej części pokażemy metodę, która zwraca informacje. Na razie po prostu stosuj przedstawiony wiersz i deklarację metody `main` w swoich programach. Fragment `String[] args` znajdujący się w nawiasach to wymagana część deklaracji `main` — zostanie opisany w rozdziale 7.

Lewy nawias klamrowy na końcu wiersza 6. rozpoczyna **treść deklaracji metody**. Kończy ją odpowiadający mu prawy nawias klamrowy (wiersz 8.). Wiersz 7. ma dodatkowe wcięcie.



#### 2.6. Dobra praktyka programistyczna

*Dokonaj dodatkowego wcięcia o jeden poziom względem poprzedniego wcięcia dla całej treści metody. W ten sposób podkreślisz strukturę metody i ułatwisz czytanie pliku.*

### Wyświetlanie informacji poleceniem `System.out.println`

Wiersz 7.:

```
System.out.println("Witamy programistę Javy!");
```

instruuje komputer, aby wykonał akcję, a dokładniej wyświetlił na ekranie tekst umieszczony między cudzysłowami. Cudzysłowy **nie** zostaną wyświetlone. Cudzysłowy i znaki między nimi tworzą tak zwany **ciąg znaków**, nazywany również **literalem tekstowym**. Białe spacje wewnątrz tekstu **nie** są przez kompilator ignorowane. Ciąg znaków **nie może** obejmować kilku wierszy kodu, ale w dalszej części książki pokażemy, jak obejść to ograniczenie.

Obiekt **`System.out`**, który jest predefiniowany, to tak zwany **standardowy obiekt wyjścia**. Umożliwia wyświetlanie informacji w **oknie poleceń**, w którym działa program. W systemie Windows okno poleceń nosi nazwę *Wiersz poleceń*, a w systemach Linux lub macOS najczęściej jest to **terminal** lub **powłoka**. Wielu programistów mówi po prostu o **wierszu poleceń**.

Metoda **`System.out.println`** wyświetla **wiersz** tekstu w wierszu poleceń. Gdy `System.out.println` zakończy swe zadanie, umieści kursor (miejsce rozpoczęcia wyświetlania następnych znaków) w pierwszej kolumnie nowego wiersza. Efekt jest taki sam jak w sytuacji, gdy naciśniesz klawisz *Enter* w edytorze tekstu — kursor przejdzie do pierwszego znaku nowego wiersza.



Cały wiersz 7., włączając w to `System.out.println`, argument "Witamy progra-  
 ↪mistę Javy!" w nawiasach i średnik (;), nosi nazwę **instrukcji** lub **polecenia**.  
 Metody najczęściej zawierają wiele instrukcji wykonujących odpowiednie zadanie. Większość instrukcji kończy się znakiem średnika.

### *Komentarze jednowierszowe na końcach nawiasów zamykających*

Jako pomoc dla początkujących programistów po nawiasach zamykających umieszczamy komentarze jednowierszowe informujące o zakończeniu deklaracji określonego elementu. Na przykład wiersz 8.:

```
} // Koniec metody main
```

wskazuje na koniec metody `main`, a wiersz 9.:

```
} // Koniec klasy Welcome1
```

wskazuje na koniec klasy `Welcome1`. Każdy z komentarzy wskazuje, że to nawias zamykający kończący określony fragment. Te dodatkowe komentarze będziemy stosowali tylko w kodzie z tego rozdziału.

## 2.2.1. Kompilacja aplikacji

Jesteśmy gotowi do skompilowania i wykonania programu. Zakładamy, że korzystasz z narzędzi wiersza poleceń JDK, a nie IDE. Poniższe instrukcje zakładają, że kod z przykładami znajduje się w folderze `c:\przyklady` w systemie Windows albo w folderze użytkownika `Documents/przyklady` w systemach Linux lub macOS.

Aby przygotować się do kompilacji programu, otwórz wiersz poleceń lub konsolę i przejdź do folderu z programem. Większość systemów operacyjnych oferuje polecenie `cd`, pozwalające przejść do wskazanego folderu. Oto przykład polecenia dla systemu Windows:

```
cd c:\przyklady\rozdzial02\02_01
```

który spowoduje przejście do folderu `02_01`. W systemach Linux i macOS polecenie ma postać:

```
cd ~/Documents/przyklady/rozdzial02/02_01
```

i powoduje przejście do folderu `02_01`. Aby skompilować program, wpisz:

```
javac Welcome1.java
```

Jeśli program nie zawiera żadnych błędów kompilacji, polecenie utworzy plik o nazwie `Welcome1.class` (nazywany też **plikiem klasy** `Welcome1`), zawierający niezależny od platformy kod bajtowy reprezentujący aplikację. Gdy użyjemy polecenia `java` w celu uruchomienia aplikacji na konkretnej platformie, JVM zamieni kod bajtowy na kod zrozumiały dla systemu operacyjnego i sprzętu.



### 2.4. Typowy błąd programistyczny

Błąd kompilacji typu `class Welcome1 is public, should be declared in a file named Welcome1.java` wskazuje, że nazwa pliku nie odpowiada nazwie klasy publicznej (oznaczonej `public`) zawartej w pliku lub podano złą nazwę klasy w trakcie kompilacji.

Czasem w trakcie nauki programowania warto celowo „popsuć” działający program, aby poznać różne błędy kompilacji. Komunikaty te nie zawsze informują o konkretnej przyczynie błędu. Po napotkaniu błędu komunikat to tylko ogólna sugestia. Spróbuj usunąć średnik lub nawias z kodu z rysunku 2.1 i skompiluj kod ponownie, aby zobaczyć błąd, który się pojawi.



### 2.2. Wskazówka zapobiegająca błędom

*Gdy kompilator zgłosi błąd składniowy, może on występować w innym wierszu niż wskazany. Najpierw sprawdź wiersz, o którym mówi komunikat. Jeśli nie znajdziesz tam błędu, szukaj kilka wierszy wyżej.*

Każdy komunikat o błędzie kompilacji zawiera nazwę pliku i numer wiersza, w którym błąd wystąpił. Na przykład `Welcome1.java:6` wskazuje, że błąd wystąpił w wierszu 6. pliku `Welcome1.java`. Pozostała część komunikatu zawiera opis błędu składniowego.

### 2.2.2. Wykonywanie aplikacji

Po skompilowaniu programu wpisz następujące polecenie i naciśnij klawisz *Enter*:

```
java Welcome1
```

aby uruchomić maszynę wirtualną Javy i wczytać plik `Welcome1.class`. Polecenie **pomija** rozszerzenie pliku `.class`. W przeciwnym razie JVM **nie** wykonałoby programu. Następnie wiersz 7. programu wyświetla tekst "Witamy programistę Javy!". Rysunek 2.2 przedstawia program wykonany w systemie Windows w programie *Wiersz polecenia*. (**Uwaga:** Większość środowisk stosuje czarne tło i biały tekst. Aby zwiększyć czytelność, dostosuj kolory).

```
C:\Windows\System32\cmd.exe
C:\przklady\rozdzial02\02_01>javac Welcome1.java
C:\przklady\rozdzial02\02_01>java Welcome1
Witamy programistę Javy!
C:\przklady\rozdzial02\02_01>
C:\przklady\rozdzial02\02_01>
```

Wpisz to polecenie, aby wykonać aplikację

Program wyświetla na ekranie tekst Witamy programistę Javy!

**Rysunek 2.2.** Wykonywanie klasy `Welcome1` w programie *Wiersz polecenia*



### 2.3. Wskazówka zapobiegająca błędom

*Jeśli w trakcie uruchamiania programu Javy otrzymasz komunikat typu `Exception in thread "main" java.lang.NoClassDefFoundError: Welcome1`, zmienna środowiskowa `CLASSPATH` nie jest ustawiona prawidłowo. Ponownie przeczytaj instrukcję instalacji JDK znajdującą się na początku książki. Niektóre systemy wymagają ponownego uruchomienia, aby zmiany dotyczące `CLASSPATH` były widoczne.*

## 2.3. Modyfikacja pierwszego programu

W tej części zmodyfikujemy przykład z rysunku 2.1, aby wyświetlić tekst w wierszu za pomocą wielu instrukcji, a także aby wyświetlić kilka wierszy tekstu za pomocą jednej instrukcji.

### *Wyświetlenie pojedynczego wiersza tekstu za pomocą kilku instrukcji*

Tekst Witamy programistę Javy! można wyświetlić na wiele sposobów. Klasa Welcome2 z rysunku 2.3 używa dwóch instrukcji (wiersz 7. i 8.), aby uzyskać efekt z rysunku 2.1. (**Uwaga:** Od tego momentu zmiany lub istotne fragmenty kodu będziemy w nowych listingach przedstawiać tak, jak wiersze 7. i 8.).

```

1 // Rysunek 2.3. Welcome2.java
2 // Wyświetlenie pojedynczego wiersza tekstu za pomocą kilku instrukcji
3
4 public class Welcome2 {
5     // Metoda main rozpoczyna wykonywanie aplikacji Javy
6     public static void main(String[] args) {
7         System.out.print("Witamy ");
8         System.out.println("programistę Javy!");
9     } // Koniec metody main
10 } // Koniec klasy Welcome2

```

Witamy programistę Javy!

**Rysunek 2.3.** Wyświetlenie pojedynczego wiersza tekstu za pomocą kilku instrukcji

Program jest bardzo podobny do tego z rysunku 2.1, więc omówimy jedynie różnice. Wiersz 2.:

```
// Wyświetlenie pojedynczego wiersza tekstu za pomocą kilku instrukcji
```

to komentarz jednowierszowy, który wyjaśnia działanie programu. Wiersz 4. rozpoczyna się od deklaracji klasy Welcome2. Wiersze 7. i 8. zawierające metodę main:

```
System.out.print("Witamy ");
System.out.println("programistę Javy!");
```

powodują wyświetlenie tylko **jednego** wiersza tekstu. Pierwsza instrukcja używa metody print z System.out do wyświetlania fragmentu tekstu. Każda instrukcja print lub println wznawia pisanie w miejscu, gdzie skończyła poprzednia. W odróżnieniu od println metoda print nie powoduje przeniesienia kursora do początku nowego wiersza — następny znak wyświetlany przez program znajduje się **dokładnie** za ostatnim znakiem wyświetlonym przez print. Oznacza to, że wiersz 8. umieści pierwszą literę ("p") bezpośrednio po ostatnim znaku z wiersza 7. (znak spacji tuż przed cudzysłowem).

### *Wyświetlenie kilku wierszy tekstu za pomocą jednej instrukcji*

Pojedyncza instrukcja może wyświetlić kilka wierszy kodu, wykorzystując **znak przejścia do nowego wiersza** (\n), który powoduje przejście przez metody print i println z System.out do początku nowego wiersza. Podobnie jak znaki spacji i tabulacji, także znaki nowego wiersza stanowią białe spacje. Program z rysunku 2.4 wyświetla trzy wiersze tekstu, używając znaku nowego wiersza do przejścia do każdej nowej linii. Większość programu jest taka sama jak na rysunkach 2.1 i 2.3.

```

1 // Rysunek 2.4. Welcome3.java
2 // Wyświetlenie kilku wierszy tekstu za pomocą jednej instrukcji
3
4 public class Welcome3 {
5     // Metoda main rozpoczyna wykonywanie aplikacji Javy
6     public static void main(String[] args) {
7         System.out.println("Witamy\nprogramistę\nJavy!");
8     } // Koniec metody main
9 } // Koniec klasy Welcome3

```

```

Witamy
programistę
Javy!

```

**Rysunek 2.4.** Wyświetlenie kilku wierszy tekstu za pomocą jednej instrukcji

Wiersz 7.:

```
System.out.println("Witamy\nprogramistę\nJavy!");
```

wyświetla w konsoli trzy wiersze tekstu. Standardowo znaki wewnątrz ciągu znaków są wyświetlane tak, jak je wpisano. Istnieją jednak pewne pary znaków (\ i n powtórzone w tekście dwukrotnie), które **nie** pojawiają się na ekranie. Lewy ukośnik (\) to znak ucieczki, który w metodach print i println z System.out ma znaczenie specjalne. Gdy lewy ukośnik pojawi się w tekście, Java łączy go ze znakiem następnym, aby utworzyć **sekwencję ucieczki** — w przypadku \n jest to znak nowego wiersza. Kiedy taki fragment pojawi się w wyświetlanym tekście, znak przejścia do nowego wiersza przeniesie kursor na początek nowego wiersza.

Rysunek 2.5 przedstawia listę kilku sekwencji ucieczki i opisuje sposób ich działania. Pełną listę wszystkich sekwencji ucieczki znajdziesz pod adresem:

<http://docs.oracle.com/javase/specs/jls/se8/html/jls-3.html#jls-3.10.6>

Sekwencja ucieczki	Opis
\n	Znak nowego wiersza. Umieszcza kursor na początku <b>nowego</b> wiersza.
\t	Znak tabulacji poziomej. Przenosi kursor do następnego miejsca tabulacji.
\r	Powrót karetki. Umieszcza kursor na początku <b>aktualnego</b> wiersza — <b>nie</b> przechodzi do nowego wiersza. Dowlone znaki po nim spowodują <b>nadpisanie</b> znaków, które już znajdują się w danym wierszu.
\\	Lewy ukośnik. Używany do wyświetlenia na ekranie znaku lewego ukośnika.
\"	Cudzysłów. Używany do wyświetlenia na ekranie znaku cudzysłowu. Na przykład instrukcja: <pre>System.out.println("\"w cudzysłowie\"");</pre> spowoduje wyświetlenie tekstu "w cudzysłowie".

**Rysunek 2.5.** Najczęściej używane sekwencje ucieczki

## 2.4. Wyświetlanie tekstu metodą printf

Metoda `System.out.printf` (f oznacza „formatowany”) wyświetla formatowane dane. Rysunek 2.6 przedstawia użycie jej do wyświetlenia dwóch osobnych tekstów: „Witamy” i „programistę Javy!”.

```

1 // Rysunek 2.6. Welcome4.java
2 // Wyświetlanie kilku wierszy metodą System.out.printf
3
4 public class Welcome4 {
5     // Metoda main rozpoczyna wykonywanie aplikacji Javy
6     public static void main(String[] args) {
7         System.out.printf("%s%n%s%n", "Witamy", "programistę Javy!");
8     } // Koniec metody main
9 } // Koniec klasy Welcome4

```

```

Witamy
programistę Javy!

```

Rysunek 2.6. Wyświetlanie kilku wierszy metodą `System.out.printf`

Wiersz 7.:

```
System.out.printf("%s%n%s%n", "Witamy", "programistę Javy!");
```

wywołuje metodę `System.out.printf` do wyświetlenia wyniku działania programu. Wywołanie metody korzysta z trzech argumentów. Gdy metoda wymaga kilku argumentów, umieszcza się je po kolei, **oddzielając przecinkami**. Wywołanie metody nazywa się czasem **zastosowaniem** metody.



### 2.7. Dobra praktyka programistyczna

Umieszczenie znaku spacji po każdym przecinku czyni listę argumentów czytelniejszą.

Pierwszym argumentem metody `printf` jest ciąg formatujący, który składa się ze **stałego tekstu** i **określników formatu**. Stały tekst jest wyświetlany przez `printf` w taki sam sposób jak przez metody `print` lub `println`. Każdy z określników formatu to **miejsce do zastąpienia** wartością, które wskazuje na **typ danych** wyjściowych. Określnik formatu może również zawierać dodatkowe informacje formatujące.

Określnik formatu rozpoczyna się znakiem procenta (%), po którym pojawia się znak reprezentujący **typ danych**. Przykładowo sekwencja `%s` oznacza miejsce do zastąpienia tekstem. W przedstawionym przykładzie ciąg formatujący wskazuje, że metoda `printf` powinna wyświetlić dwa teksty, a po każdym z nich przejść do nowego wiersza. Pierwszy określnik zostanie zastąpiony pierwszym argumentem po ciągu formatującym, a drugi następnym argumentem. W przykładzie tekst „Witamy” zastąpi pierwsze `%s`, a tekst „programistę Javy!” drugie `%s`. Każdy z tekstów pojawi się w swoim własnym wierszu.

Zwróć uwagę, że zamiast sekwencji ucieczki `\n` użyliśmy określnika formatu `%n`, który stosuje sekwencję przejścia do nowego wiersza **przenośną** między systemami. Nie można jej użyć jako argumentu w metodach `System.out.print` lub

System.out.println, ale przejście do nowego wiersza stosowane przez metodę System.out.println po wyświetleniu tekstu jest przenośne między systemami.

## 2.5. Inna aplikacja — dodawanie liczb całkowitych

Nasza następna aplikacja odczytuje dwie **liczby całkowite** (czyli liczby bez części ułamkowej, takie jak -22, 7, 0 lub 1024) wpisane przez użytkownika za pomocą klawiatury, dodaje je do siebie i wyświetla wynik. Program musi zapamiętać wartości podane przez użytkownika, aby móc je później zsumować. Programy pamiętają liczby i inne dane w pamięci komputera poprzez elementy nazywane **zmiennymi**. Program z rysunku 2.7 ilustruje to pojęcie. W przykładowym wyjściu użyliśmy kursywy za pogrubieniem do wskazania wartości wprowadzanych przez użytkownika (liczby 45 i 72). Podobnie jak w innych programach pierwszy i drugi wiersz zawiera odpowiednio nazwę pliku i opis sposobu działania.

```

1 // Rysunek 2.7. Addition.java
2 // Program dodający dwie wartości i wyświetlający wynik
3 import java.util.Scanner; // Program używa klasy Scanner
4
5 public class Addition {
6     // Metoda main rozpoczyna wykonywanie aplikacji Javy
7     public static void main(String[] args) {
8         // Tworzy obiekt Scanner, aby pobrać dane z konsoli
9         Scanner input = new Scanner(System.in);
10
11         System.out.print("Wpisz pierwszą liczbę: "); // Zapytanie
12         int number1 = input.nextInt(); // Odczytanie pierwszej liczby
13
14         System.out.print("Wpisz drugą liczbę: "); // Zapytanie
15         int number2 = input.nextInt(); // Odczytanie drugiej liczby
16
17         int sum = number1 + number2; // Dodanie liczb i zapamiętanie wyniku w sum
18
19         System.out.printf("Suma wynosi %d%n", sum); // Wyświetlenie sumy
20     } // Koniec metody main
21 } // Koniec klasy Addition

```

```

Wpisz pierwszą liczbę: 45
Wpisz drugą liczbę: 72
Suma wynosi 117

```

Rysunek 2.7. Program dodający dwie wartości i wyświetlający wynik

### 2.5.1. Deklaracja import

Ogromną zaletą Javy jest bogaty zbiór predefiniowanych klas, z których można korzystać wielokrotnie i nie trzeba wymyślać koła na nowo. Klasy są pogrupowane w pakiety — nazwane grupy powiązanych ze sobą klas — a ich zbiór nazywa się biblioteką klas Javy lub interfejsem programistycznym Javy (API Javy). Wiersz 3.:

```
import java.util.Scanner; // Program używa klasy Scanner
```

to deklaracja `import`, która pozwala określić lokalizację klasy wykorzystywanej przez program. Wskazuje, że program używa predefiniowanej klasy `Scanner` (o której za chwilę) z pakietu o nazwie `java.util`. Kompilator upewni się, czy klasa jest stosowana prawidłowo.



### 2.5. Typowy błąd programistyczny

*Wszystkie deklaracje `import` muszą pojawić się w pliku przed pierwszą deklaracją klasy. Umieszczenie deklaracji `import` w środku lub po deklaracji klasy jest błędem składniowym.*



### 2.6. Typowy błąd programistyczny

*Jeśli zapomnisz dodać deklarację `import`, spowoduje to błąd kompilacji z komunikatem typu `cannot find symbol`. Gdy tak się stanie, sprawdź, czy użyłeś deklaracji i czy wszystkie nazwy klas i pakietów są podane prawidłowo (włączając w to wielkość liter).*

## 2.5.2. Deklaracja i utworzenie obiektu `Scanner` w celu pobrania danych z klawiatury

Zmienna to miejsce w pamięci komputera, w którym to wartość może być przechowywana w celu późniejszego użycia. Wszystkie zmienne Javy muszą być zadeklarowane za pomocą **nazwy** i **typu** przed ich pierwszym użyciem. **Nazwa** zmiennej umożliwia programowi dostęp do **wartości** zmiennej znajdującej się w pamięci. Nazwa zmiennej to dowolny poprawny identyfikator, czyli dowolny ciąg liter, cyfr, podkreśleń (`_`) lub znaków dolara (`$`), który **nie** rozpoczyna się od cyfry i **nie** zawiera znaków spacji. Typ zmiennej określa rodzaj informacji przechowywanej w pamięci. Podobnie jak inne instrukcje, także deklaracja zmiennej kończy się znakiem średnika (`;`).

Wiersz 9. w metodzie `main`:

```
Scanner input = new Scanner(System.in);
```

to instrukcja deklaracji zmiennej, która wskazuje nazwę (`input`) i typ (`Scanner`) zmiennej do wykorzystania w programie. Klasa **`Scanner`** (z pakietu `java.util`) umożliwia programowi odczyt danych (np. liczb lub tekstu) w celu ich użycia. Dane mogą pochodzić z wielu różnych źródeł, w tym z klawiatury lub dysku twardego. Zanim użyjemy klasy `Scanner`, musimy utworzyć jej obiekt i wskazać **źródło** danych.

Znak `=` w wierszu 9. wskazuje, że zmienna `input` powinna zostać **zainicjalizowana** (przygotowana do użycia w programie) w momencie deklaracji wartością wyrażenia znajdującego się na prawo od znaku równości — `new Scanner(System.in)`. Wyrażenie używa słowa kluczowego **`new`** do utworzenia obiektu `Scanner`, który odczytuje znaki wpisywane na klawiaturze. **Standardowy obiekt wejścia, `System.in`**, umożliwia aplikacji odczyt bajtów danych wpisywanych przez użytkownika. Obiekt `Scanner` zamienia bajty na odpowiednie typy (np. `int`) w celu ich użycia przez program.



### 2.8. Dobra praktyka programistyczna

Wybór odpowiednich nazw zmiennych spowoduje, że program będzie się w zasadzie sam dokumentował (czyli zrozumienie zasady działania programu będzie możliwe przez samą analizę jego kodu bez stosowania dodatkowej dokumentacji zewnętrznej lub komentarzy).



### 2.9. Dobra praktyka programistyczna

Zgodnie z przyjętą konwencją w identyfikatorach nazw zmiennych stosuje się zapis wielbłądzi, ale z małą pierwszą literą, np. liczbaPierwsza.

## 2.5.3. Prośba o wprowadzenie danych

Wiersz 11.:

```
System.out.print("Wpisz pierwszą liczbę: "); //Zapytanie
```

używa metody `System.out.print` do wyświetlenia tekstu "Wpisz pierwszą liczbę: ". Ten tekst to **zapytanie**, ponieważ powoduje, że użytkownik będzie musiał podjąć pewną akcję. Używamy metody `print` zamiast metody `println`, aby użytkownik wprowadzał dane w tym samym wierszu co zapytanie. Przypomnij sobie z poprzedniego rozdziału informację, że identyfikatory zaczynające się od wielkiej litery zazwyczaj dotyczą nazw klas. Klasa `System` stanowi część pakietu `java.lang`.



### 2.1. Obserwacja z poziomu inżynierii oprogramowania

Pakiet `java.lang` jest domyślnie zaimportowany w każdym programie Javy. Klasy należące do tego pakietu to jedyne klasy, dla których nie trzeba stosować deklaracji `import`.

## 2.5.4. Deklaracja zmiennej dla liczby całkowitej i pobranie wartości z klawiatury

Deklaracja zmiennej z wiersza 12.:

```
int number1 = input.nextInt(); // Odczytanie pierwszej liczby
```

deklaruje zmienną `number1`, aby przechowywała daną typu `int`, czyli liczbę całkowitą w postaci 72, -1127 lub 0. Zakresem dopuszczalnych wartości dla typu `int` są liczby od -2 147 483 648 do +2 147 483 647. Wartości `int` oczywiście nie zawierają znaków spacji, ale dla zwiększenia czytelności dopuszcza się stosowanie w liczbach znaku podkreślenia, więc 60\_000\_000 reprezentuje wartość `int` równą 60 000 000.

Innymi rodzajami danych są `float` i `double`, które przechowują liczby zmiennoprzecinkowe, a także typ `char`, który przechowuje konkretny znak. Liczby zmiennoprzecinkowe to liczby mogące zawierać ułamek, np. 3.4, 0.0 lub -11.19 (w kodzie zamiast znaku przecinka używa się znaku kropki). Zmienna typu `char` reprezentuje pojedynczy znak, np. dużą literę (A), cyfrę (7), znak specjalny (np. \* lub %), a także sekwencję ucieczki (np. znak tabulacji \t). Typy `int`, `float`, `double` i `char` są **typami podstawowymi**. Nazwy tych typów są słowami kluczowymi i muszą być zapisywane małymi literami. Dodatek D podsumowuje charakterystykę wszystkich ośmiu typów podstawowych (`boolean`, `byte`, `char`, `short`, `int`, `long`, `float` i `double`).



Znak = w wierszu 12. wskazuje, że zmienna `number1` typu `int` powinna zostać w momencie deklaracji zainicjalizowana wynikiem `input.nextInt()`. Metoda `next` ↪ `Int()` obiektu `Scanner` pobiera z klawiatury liczbę całkowitą. Program czeka, aż użytkownik wpisze liczbę i naciśnie klawisz *Enter*, aby potwierdzić wartość.

Nasz program zakłada, że użytkownik wprowadzi wartość poprawną. Jeśli nie, pojawi się błąd logiczny i program przestanie działać. Rozdział 11. zawiera informacje na temat zabezpieczania programów przed tego rodzaju błędami. Mówi się, że program przygotowany na błędy jest odporny na porażki.

### 2.5.5. Pobranie drugiej liczby całkowitej

Wiersz 14.:

```
System.out.print("Wpisz drugą liczbę: "); // Zapytanie
```

odpytuje użytkownika o drugą liczbę całkowitą. Wiersz 15.:

```
int number2 = input.nextInt(); // Odczytanie drugiej liczby
```

deklaruje zmienną `number2` typu `int` i inicjalizuje ją drugą wartością całkowitą pobraną od użytkownika.

### 2.5.6. Użycie zmiennych w obliczeniach

Wiersz 17.:

```
int sum = number1 + number2; // Dodanie liczb i zapamiętanie wyniku w sum
```

deklaruje zmienną `sum` typu `int` i inicjalizuje ją wynikiem operacji `number1 + number2`. Gdy program napotka operację dodawania, wykonuje obliczenia na podstawie wartości znajdujących się w zmiennych `number1` i `number2`.

W przedstawionym kodzie operator dodawania jest **operatorem binarnym**, ponieważ wykorzystuje **dwa operandy** — `number1` i `number2`. Fragment dotyczący obliczenia nazywamy **wyrażeniem**. W zasadzie wyrażenie to dowolny fragment instrukcji posiadający **wartość**. Wartością wyrażenia `number1 + number2` jest **suma** liczb. Podobnie wartością wyrażenia `input.nextInt()` (wiersze 12. i 15.) jest liczba całkowita wpisana przez użytkownika.



#### 2.10. Dobra praktyka programistyczna

*Aby zwiększyć czytelność kodu, umieść znak spacji przed operatorem binarnym i po nim.*

### 2.5.7. Wyświetlenie wyniku obliczeń

Po wykonaniu obliczeń wiersz 19.:

```
System.out.printf("Suma wynosi %d%n", sum); // Wyświetlenie sumy
```

używa metody `System.out.printf` do wyświetlenia wartości zmiennej `sum`. Określnik formatu `%d` to **miejsce do zastąpienia** wartością typu `int` (w tym przypadku wartością z `sum`). Wszystkie pozostałe znaki tekstu formatującego to stała treść. Metoda `printf` wyświetli więc tekst "Suma wynosi ", po nim wartość `sum` (w miejscu wskazanym przez `%d`), a na końcu użyje znaku przejścia do nowego wiersza.

Obliczenia można również wykonać **wewnątrz** instrukcji wywołującej `printf`. Moglibyśmy połączyć wiersze 17. i 19. w jedną instrukcję:

```
System.out.printf("Suma wynosi %d%n", (number1 + number2));
```

Nawiasy, w których umieszczone jest wyrażenie `number1 + number2`, są opcjonalne — służą jedynie podkreśleniu, że wartość całego wyrażenia jest umieszczona w określniku formatu `%d`. Takie nawiasy nazywa się **redundantnymi**.

### 2.5.8. Dokumentacja API Javy

Dla każdej klasy z API Javy będziemy wskazywali pakiet, z którego pochodzi. W ten sposób będziesz w stanie szybko znaleźć opisy pakietów i klas w oficjalnej dokumentacji. Dokumentację w wersji WWW znajdziesz pod adresem:

```
http://docs.oracle.com/javase/8/docs/api/index.html
```

Wersję do pobrania znajdziesz w części *Additional Resources* na stronie:

```
http://www.oracle.com/technetwork/java/javase/downloads/index.html
```

### 2.5.9. Deklaracja i inicjalizacja zmiennej jako osobne instrukcje

Każda zmienna musi mieć wartość, **zanim** będzie można jej użyć w obliczeniach (lub innym wyrażeniu). Instrukcja z wiersza 12. dokonywała zarówno deklaracji zmiennej `number1`, **jak i** jej inicjalizacji wartością wpisaną przez użytkownika.

Zdarza się, że deklaracja zmiennej następuje w jednej instrukcji, a w drugiej jest inicjalizowana. Na przykład wiersz 12. można by rozbić na dwie instrukcje:

```
int number1; // Deklaracja zmiennej number1 typu int
number1 = input.nextInt(); // Przypisanie zmiennej number1 danych pobranych
// od użytkownika
```

Pierwsza instrukcja deklaruje zmienną `number1`, ale jej **nie** inicjalizuje. Druga instrukcja używa operatora przypisania (`=`) do **przypisania** (czyli nadania) zmiennej `number1` wartości wprowadzonej przez użytkownika. Instrukcję można przeczytać jako rozkaz „**nadaj** `number1` wartość pochodzącą z `input.nextInt()`”. Wszystko, co znajduje się po **prawej** stronie operatora przypisania, jest zawsze wyliczane **przed** wykonaniem samego przypisania.

## 2.6. Zagadnienia dotyczące pamięci

Nazwy zmiennych takie jak `number1`, `number2` i `sum` tak naprawdę odpowiadają **miejscom** w pamięci komputera. Każda zmienna ma **nazwę**, **typ**, **rozmiar** (w bajtach) i **wartość**.

Gdy wykona się następujący fragment kodu z rysunku 2.7 (wiersz 12.):

```
int number1 = input.nextInt(); // Odczytanie pierwszej liczby
```

liczba wpisana przez użytkownika trafi do miejsca w pamięci wskazywanego przez nazwę `number1`. Przypuśćmy, że użytkownik wpisze 45. Komputer umieści liczbę w położeniu `number1` (rysunek 2.8), zastępując nią poprzednią wartość w tym miejscu (jeśli istniała). Poprzednia wartość jest tracona, więc mówimy, że cały proces jest **destrukcyjny**.

Gdy wykona się instrukcja z wiersza 15.:

```
int number2 = input.nextInt(); // Odczytanie drugiej liczby
```

number1	45
---------	----

**Rysunek 2.8.** Miejsce w pamięci przedstawiające nazwę i wartość zmiennej number1

przypuśćmy, że jako wynik otrzymaliśmy 72. Komputer umieszcza liczbę całkowitą w miejscu wskazywanym przez zmienną number2. Pamięć wygląda obecnie tak, jak na rysunku 2.9.

number1	45
number2	72

**Rysunek 2.9.** Miejsce w pamięci dla zmiennych number1 i number2

Po tym, jak program z rysunku 2.7 otrzymał obie wartości i umieścił je w zmiennych number1 i number2, dodaje je do siebie i sumę umieszcza w zmiennej sum. Instrukcja (wiersz 17.):

```
int sum = number1 + number2; // Dodanie liczb i zapamiętanie wyniku w sum
```

wykonuje dodawanie i zastępuje wynikiem dowolną wcześniejszą wartość znajdującą się w sum. Po tej operacji pamięć wygląda tak, jak na rysunku 2.10. Wartości zmiennych number1 i number2 wyglądają dokładnie tak samo jak wcześniej pomimo użycia ich do wyliczenia sum. Wartości te były używane, ale **nie niszczone**, w trakcie operacji dodawania. Gdy odczytujemy wartość z pamięci, jest to operacja **nie-destrukcyjna**.

number1	45
number2	72
sum	117

**Rysunek 2.10.** Miejsce w pamięci po zsumowaniu wartości ze zmiennych number1 i number2

## 2.7. Operacje arytmetyczne

Większość programów wykonuje operacje arytmetyczne. Podsumowanie dostępnych operatorów arytmetycznych zawiera rysunek 2.11. Zwróć uwagę na pewne symbole specjalne niestosowane w algebrze. **Znak gwiazdki (\*)** oznacza mnożenie, a znak procenta (%) **resztę z dzielenia**, o którym więcej wkrótce. Operatory z rysunku 2.11 to operatory **binarne**, ponieważ wykorzystują **dwa** operandy. Na przykład wyrażenie  $f + 7$  zawiera operator binarny  $+$  i dwa operandy:  $f$  i  $7$ .

Dzielenie liczb całkowitych powoduje powstanie całkowitego wyniku. Na przykład wyrażenie  $7 / 4$  zwraca wynik 1, a  $17 / 5$  wynik 3. Część ułamkowa jest po prostu **ucinana (pomijana)** — nie dochodzi do **zaokrąglenia**. Java udostępnia operator reszty z dzielenia (%), który zwraca uciętą część. Wyrażenie  $x \% y$

Operacja	Operator	Wyrażenie algebraiczne	Wyrażenie w Javie
dodawanie	+	$f + 7$	<code>f + 7</code>
odejmowanie	-	$p - c$	<code>p - c</code>
mnożenie	*	$bm$	<code>b * m</code>
dzielenie	/	$x / y$ lub $\frac{x}{y}$ albo $x \div y$	<code>x / y</code>
reszta z dzielenia	%	$r \bmod s$	<code>r % s</code>

Rysunek 2.11. Operatory arytmetyczne

zwróci resztę z dzielenia  $x$  przez  $y$ . Wynikiem operacji  $7 \% 4$  będzie 3, a wynikiem operacji  $17 \% 5$  będzie 2. Operator jest używany głównie dla liczb całkowitych, ale można go użyć również w połączeniu z innymi typami. W ćwiczeniach dotyczących tego rozdziału i następnych rozważymy kilka interesujących zastosowań operatora reszty z dzielenia, na przykład sprawdzenie, czy pewna liczba jest wielokrotnością innej.

### Wyrażenia arytmetyczne zapisywane w linii prostej

Wyrażenia arytmetyczne w Javie trzeba zapisywać w linii prostej, czyli wersji przyjaznej dla komputerów. Oznacza to, że wyrażenie „ $a$  dzielone przez  $b$ ” zapisuje się w postaci `a / b`, więc wszystkie stałe, zmienne i operatory pojawiają się w linii prostej. Kompilatory nie akceptują następującego zapisu algebraicznego:

$$\frac{a}{b}$$

### Nawiasy do grupowania podwyrażeń

Nawiasy służą do grupowania wyrażeń w języku Java dokładnie w taki sam sposób jak w standardowych działaniach matematycznych. Aby pomnożyć  $a$  przez sumę  $b + c$ , napiszemy:

$$a * (b + c)$$

Jeśli wyrażenie zawiera zagnieżdżone nawiasy, jak tutaj:

$$((a + b) * c)$$

najpierw wyliczane są wartości **najbardziej wewnętrznych** nawiasów (w tym przypadku  $a + b$ ).

### Kolejność wykonywania działań

Java stosuje operatory arytmetyczne w ściśle określonej kolejności nazywanej **kolejnością wykonywania działań**, która w większości sytuacji jest taka sama jak w przypadku algebry.

1. Najpierw wykonuje się operacje mnożenia, dzielenia i reszty z dzielenia. Jeśli wyrażenie zawiera kilka tego rodzaju operacji, są one realizowane od lewej do prawej. Operacje mnożenia, dzielenia i reszty z dzielenia mają taki sam priorytet.
2. Następnie wykonuje się operacje dodawania i odejmowania. Jeśli wyrażenie zawiera kilka tego typu operacji, są one realizowane od lewej do prawej. Operacje dodawania i odejmowania mają taki sam priorytet.

Zasady te umożliwiają Javie stosowanie operatorów we właściwym **porządku**<sup>1</sup>. Gdy operatory są stosowane od lewej do prawej, mówimy o ich **łączności** (asocjacyjności). Są jednak operatory, które łączą się od prawej do lewej. Rysunek 2.12 zawiera podsumowanie wszystkich reguł dla poznanych do tej pory operatorów. Kompletną listę operatorów zawiera dodatek A.

Operatory	Operacje	Kolejność wykonywania
*	mnożenie	Wyliczone w pierwszej kolejności. Jeśli istnieje kilka operatorów tego typu, są wyliczone od <b>lewej do prawej</b> .
/	dzielenie	
%	reszta z dzielenia	
+	dodawanie	Wyliczone w drugiej kolejności. Jeśli istnieje kilka operatorów tego typu, są wyliczone od <b>lewej do prawej</b> .
-	odejmowanie	
=	przypisanie	Wykonywane na końcu.

**Rysunek 2.12.** Kolejność wykonywania działań

### Przykładowe wyrażenia algebraiczne i wyrażenia w języku Java

Przyjrzyjmy się kilku przykładom wyrażeń. Każde z nich przedstawione jest w wersji ze standardowym zapisem matematycznym i równoważnej wersji w języku Java. Oto przykład wyliczający średnią pięciu zmiennych:

$$\text{Algebra: } m = \frac{a + b + c + d + e}{5}$$

$$\text{Java: } m = (a + b + c + d + e) / 5;$$

Nawiasy są niezbędne, ponieważ dzielenie ma wyższy priorytet niż dodawanie. Cały element  $(a + b + c + d + e)$  musi zostać podzielony przez 5. Gdybyśmy pominęli nawiasy, otrzymalibyśmy zapis  $a + b + c + d + e / 5$ , który byłby równoważny wersji

$$a + b + c + d + \frac{e}{5}$$

Oto przykład wyrażenia zapisywanego w jednym wierszu:

$$\text{Algebra: } y = mx + b$$

$$\text{Java: } y = m * x + b;$$

Tutaj nawiasy nie są wymagane. Operacja mnożenia jest realizowana jako pierwsza, bo ma wyższy priorytet niż dodawanie. Przypisanie jest realizowane na końcu, bo ma niższy priorytet niż mnożenie lub dodawanie.

Poniższy przykład wykorzystuje operatory reszty z dzielenia (%), mnożenia, dzielenia, dodawania i odejmowania:

<sup>1</sup> Używamy prostych przykładów do przedstawienia kolejności wykonywania działań. W bardziej złożonych wyrażeniach mogą pojawić się subtelne problemy z kolejnością. Więcej informacji na ten temat znajdziesz w rozdziale 15. specyfikacji języka Java (<https://docs.oracle.com/javase/specs/jls/se8/html/jls-15.html>).

Algebra:  $z = pr \% q + w / x - y$

Java: `= p * r % q + w / x - y;`



Liczby w kółkach pod instrukcją wskazują **kolejność**, w jakiej Java zastosuje operatory. Operacje `*`, `%` i `/` są wykonywane jako pierwsze w kolejności **od lewej do prawej** (łączą się od lewej do prawej), ponieważ mają wyższy priorytet niż operacje `+` i `-`. Następnie są wykonywane operacje `+` i `-`. One też są realizowane **od lewej do prawej**. Operator przypisania (`=`) wchodzi do gry jako ostatni.

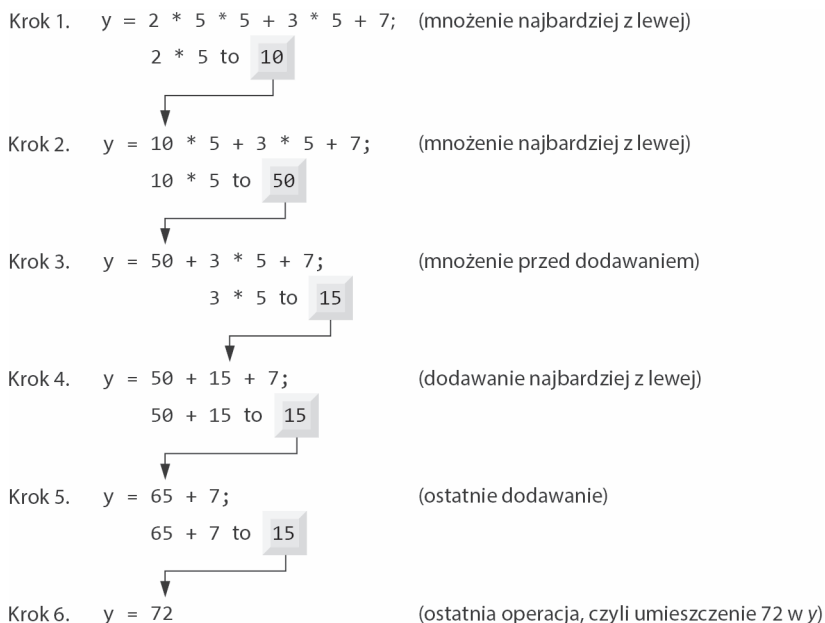
### Wyliczanie wielomianu stopnia drugiego

Aby lepiej zrozumieć kolejność wykonywania działań, przeanalizujmy obliczanie wartości wyrażenia, które jest wielomianem stopnia drugiego w postaci  $ax^2 + bx + c$ :

`y = a * x * x + b * x + c;`



Operacje mnożenia są wykonywane w pierwszej kolejności i od lewej do prawej, bo mają wyższy priorytet niż dodawanie. (Ponieważ Java nie ma operatora arytmetycznego dla podnoszenia do potęgi,  $x^2$  zapisaliśmy jako `x * x`. Alternatywę dla tej operacji zawiera podrozdział 5.4). Operacje dodawania są wykonywane **od lewej do prawej**. Przypuśćmy, że zmienne `a`, `b`, `c` i `x` zainicjalizowano (nadano im wartości) w następujący sposób: `a = 2`, `b = 3`, `c = 7` i `x = 5`. Rysunek 2.13 przedstawia kolejność, w której będą stosowane operatory.



**Rysunek 2.13.** Kolejność wylizczania wartości w wielomianie stopnia drugiego

Można zastosować nieobowiązkowe nawiasy, aby uczynić wyrażenie bardziej czytelnym. Analizowane wyrażenie można zapisać następująco:

$$y = (a * x * x) + (b * x) + c;$$

## 2.8. Podejmowanie decyzji — operatory równości i relacji

**Warunek** to wyrażenie, które może być **prawdziwe** lub **falsywe**. W tym podrozdziale omówimy **instrukcję warunkową if**, która umożliwi programowi podejmowanie **decyzji** na podstawie zawartości zmiennej. Na przykład warunek „suma punktów jest większa lub równa 60” określa, czy student zdał test. Jeśli warunek z instrukcji **if** jest **prawdziwy**, wykona się treść zawarta w instrukcji. Jeśli jest **falsywy**, zostanie pominięta.

Warunki w instrukcjach **if** mogą powstawać z użyciem **operatorów równości** (**==** i **!=**) lub **operatorów relacji** (**>**, **<**, **>=** i **<=**) podsumowanych na rysunku 2.14. Oba operatory równości mają taki sam priorytet, który jest **niższy** od priorytetu operatorów relacji. Operatory równości łączą się od **lewej do prawej**. Operatory relacji mają taki sam priorytet i również łączą się od **lewej do prawej**.

Operator matematyczny	Operator równości lub relacji w Javie	Przykładowy warunek w Javie	Znaczenie warunku w Javie
<i>Operatory równości</i>			
=	==	x == y	x jest równe y
≠	!=	x != y	x jest różne od y
<i>Operatory relacji</i>			
>	>	x > y	x jest większe od y
<	<	x < y	x jest mniejsze od y
≥	>=	x >= y	x jest większe lub równe y
≤	<=	x <= y	x jest mniejsze lub równe y

**Rysunek 2.14.** Operatory równości i relacji

Na rysunku 2.15 zostało użytych sześć instrukcji **if** do porównywania dwóch liczb całkowitych wpisanych przez użytkownika. Jeśli warunek w którejkolwiek instrukcji będzie **prawdziwy**, instrukcja powiązana z instrukcją **if** wykona się; w przeciwnym razie zostanie pominięta. Używamy klasy **Scanner** do pobrania liczb całkowitych od użytkownika i umieszczenia ich w zmiennych **number1** i **number2**. Program **porównuje** liczby i wyświetla informacje, jeżeli wynik porównywania jest prawdziwy. Przedstawiamy również trzy wyniki dla różnych wartości wpisanych przez użytkownika.

```

1 // Rysunek 2.15. Comparison.java
2 // Porównywanie liczb za pomocą instrukcji if, operatorów równości i operatorów relacji
3 import java.util.Scanner; // Program używa klasy Scanner
4
5 public class Comparison {
6     // Metoda main rozpoczyna wykonywanie aplikacji Java

```

**Rysunek 2.15.** Porównywanie liczb za pomocą instrukcji **if**, operatorów równości i operatorów relacji

```

7 public static void main(String[] args) {
8     // Tworzy obiekt Scanner, aby pobrać dane z konsoli
9     Scanner input = new Scanner(System.in);
10
11     System.out.print("Wpisz pierwszą liczbę: "); // Zapytanie
12     int number1 = input.nextInt(); // Odczytanie pierwszej liczby
13
14     System.out.print("Wpisz drugą liczbę: "); // Zapytanie
15     int number2 = input.nextInt(); // Odczytanie pierwszej liczby
16
17     if (number1 == number2) {
18         System.out.printf("%d == %d%n", number1, number2);
19     }
20
21     if (number1 != number2) {
22         System.out.printf("%d != %d%n", number1, number2);
23     }
24
25     if (number1 < number2) {
26         System.out.printf("%d < %d%n", number1, number2);
27     }
28
29     if (number1 > number2) {
30         System.out.printf("%d > %d%n", number1, number2);
31     }
32
33     if (number1 <= number2) {
34         System.out.printf("%d <= %d%n", number1, number2);
35     }
36
37     if (number1 >= number2) {
38         System.out.printf("%d >= %d%n", number1, number2);
39     }
40 } // Koniec metody main
41 } // Koniec klasy Comparison

```

```

Wpisz pierwszą liczbę: 777
Wpisz drugą liczbę: 777
777 == 777
777 <= 777
777 >= 777

```

```

Wpisz pierwszą liczbę: 1000
Wpisz drugą liczbę: 2000
1000 != 2000
1000 < 2000
1000 <= 2000

```

```

Wpisz pierwszą liczbę: 2000
Wpisz drugą liczbę: 1000
2000 != 1000
2000 > 1000
2000 >= 1000

```

Rysunek 2.15. Porównywanie liczb za pomocą instrukcji if, operatorów równości i operatorów relacji — ciąg dalszy



Metoda main (wiersze od 7. do 40.) klasy Comparison rozpoczyna wykonywanie programu. Wiersz 9.:

```
Scanner input = new Scanner(System.in);
```

deklaruje zmienną input typu Scanner i przypisuje jej obiekt Scanner powiązany ze standardowym wejściem (np. klawiaturą).

Wiersze 11. i 12.:

```
System.out.print("Wpisz pierwszą liczbę: "); //Zapytanie
int number1 = input.nextInt(); //Odczytanie pierwszej liczby
```

pytają użytkownika o pierwszą liczbę całkowitą i zapisują ją w zmiennej number1 typu int.

Wiersze 14. i 15.:

```
System.out.print("Wpisz drugą liczbę: "); //Zapytanie.
int number2 = input.nextInt(); //Odczytanie drugiej liczby.
```

pytają użytkownika o drugą liczbę całkowitą i zapisują ją w zmiennej number2 typu int.

Wiersze od 17. do 19.:

```
if (number1 == number2) {
    System.out.printf("%d == %d%n", number1, number2);
}
```

porównują zawartość zmiennych number1 i number2 pod kątem równości. Jeśli wartości są sobie równe, instrukcja z wiersza 18. wyświetla tekst wskazujący, że wartości są równe. Instrukcje if z wierszy 21., 25., 29., 33. i 37. porównują number1 i number2, wykorzystując operatory !=, <, >, <= i >=. Jeśli warunki są prawdziwe w jednej z tych instrukcji, wówczas wyświetli się tekst umieszczony poniżej warunku.

Każda instrukcja if z rysunku 2.15 zawiera jako swoją treść tylko jedno wcięte polecenie. Zauważ, że treść znajduje się w nawiasach klamrowych ({ i }). Tworzy to tak zwaną **instrukcję złożoną** lub **blok instrukcji**.



### 2.11. Dobra praktyka programistyczna

Zastosuj wcięcie dla instrukcji znajdujących się wewnątrz instrukcji if, aby poprawić czytelność kodu. IDE najczęściej wykonują to zadanie automatycznie po określeniu wielkości wcięcia.



### 2.4. Wskazówka zapobiegająca błędom

Jeśli treścią wykonywaną dla warunku jest tylko jedno polecenie, nie trzeba używać nawiasów klamrowych. Są one niezbędne tylko w sytuacji, gdy poleceń jest więcej. Wkrótce się przekonasz, że zapominanie o stosowaniu nawiasów klamrowych w przypadku wielu instrukcji wewnątrz treści prowadzi do subtelnych błędów. Aby ich uniknąć, zawsze stosuj nawiasy klamrowe dla instrukcji if.

### Białe spacje

Zwróć uwagę na zastosowanie białych spacji na rysunku 2.15. Przypomnijmy, że kompilator najczęściej ignoruje białe spacje. Jeśli instrukcje zajmują kilka wierszy, warto rozdzielić je w sposób logiczny, co poprawia czytelność, ale nie wpływa



### 2.7. Typowy błąd programistyczny

Umieszczanie średnika tuż po nawiasie zamykającym warunek w instrukcji `if` jest często błędem logicznym (ale nie błędem składniowym). Średnik powoduje, że treść instrukcji `if` jest tak naprawdę pusta, więc spełnienie warunku nie spowoduje żadnej akcji. Co gorsza, oryginalna treść warunku wykona się zawsze (niezależnie od warunku), co spowoduje zapewne błędne działanie programu.



### 2.5. Wskazówka zapobiegająca błędom

Długie instrukcje mogą swym zasięgiem obejmować kilka wierszy. Jeśli taki podział jest konieczny, użyj naturalnych miejsc złamania wiersza, na przykład po przecinku w przypadku listy elementów lub po operatorze w przypadku długiego wyrażenia. Jeżeli instrukcja jest podzielona na kilka wierszy, stosuj wcięcia aż do jej końca.

na działanie aplikacji. W idealnej sytuacji liczba instrukcji w metodzie powinna być stosunkowo niewielka, ale nie zawsze jest to możliwe.

### Opisane dotychczas operatory

Rysunek 2.16 przedstawia omówione dotychczas operatory od najwyższego do najniższego priorytetu. Wszystkie poza operatorem przypisania (=) dotyczą łączenia od lewej do prawej. Operator przypisania wykonuje się od prawej do lewej. To, co trafia do zmiennej wskazanej po lewej stronie jako efekt działania wyrażenia znajdującego się po prawej stronie operatora przypisania, jest też wynikiem całego operatora — na przykład wartością wyrażenia  $x = 7$  jest 7. Wyrażenie takie jak  $x = y = 0$  wylicza się tak, jakby było zapisane jako  $x = (y = 0)$ , czyli najpierw następuje przypisanie wartości 0 do zmiennej  $y$ , a dopiero potem przypisanie wyniku tego przypisania (0) do zmiennej  $x$ .

Operatory	Asocjacyjność	Typ
* / %	od lewej do prawej	mnożące
+ -	od lewej do prawej	dodające
< <= > >=	od lewej do prawej	relacyjne
== !=	od lewej do prawej	równości
=	od prawej do lewej	przypisania

Rysunek 2.16. Priorytet i kierunek działania przedstawionych dotychczas operatorów



### 2.12. Dobra praktyka programistyczna

Pisząc wyrażenie zawierające wiele operatorów, skorzystaj z tabeli kolejności operatorów dostępnej w dodatku A. Sprawdź, czy operacje w wyrażeniu są wykonywane w kolejności, której oczekujesz. Jeśli w przypadku złożonego wyrażenia nie jesteś pewien kolejności, użyj z nawiasów, aby wymusić pożądaną kolejność wykonywania operacji.

## 2.9. Podsumowanie

W tym rozdziale poznałeś naprawdę dużo rozmaitych funkcjonalności Javy, w tym wyświetlanie danych na ekranie, wprowadzanie danych z klawiatury, wykonywanie obliczeń i podejmowanie decyzji. Przedstawione tu aplikacje wprowadzają wiele podstawowych pojęć programistycznych. Przekonasz się podczas lektury następnego rozdziału, że aplikacje Javy zawierają w metodzie `main` jedynie kilka wierszy kodu, ponieważ większość prac aplikacja wykonuje przez tworzenie obiektów i wywoływanie metod. Z rozdziału 3. dowiesz się, jak tworzyć własne klasy i używać utworzonych na ich podstawie obiektów.

### Streszczenie

#### *Podrozdział 2.2. Twój pierwszy program — wyświetlenie wiersza tekstu*

- Aplikację Javy uruchamia się, wykonując polecenie `java` w celu uruchomienia JVM.
- Komentarze dokumentują program i zwiększają jego czytelność. Kompilator je ignoruje.
- Komentarz jednowierszowy zaczyna się znakami `//` i kończy wraz z końcem wiersza, w którym się znajduje.
- Komentarz tradycyjny może obejmować swym zasięgiem wiele wierszy. Zaczyna się znakami `/*` i kończy znakami `*/`.
- Komentarz Javadoc, zaczynający się znakami `/**` i kończący znakami `*/`, pozwala osadzać dokumentację bezpośrednio w kodzie. Program `javadoc` umożliwia generowanie na podstawie tych komentarzy dokumentacji projektowej.
- Błąd składniowy występuje, gdy kompilator natknie się na kod, który łamie reguły języka Java. Przypomina to błąd gramatyczny w języku naturalnym.
- Puste wiersze, znaki spacji i znaki tabulacji to tak zwane białe spacje. Dzięki nim program jest czytelniejszy, ale nie wpływa to na jego działanie (kompilator takie znaki ignoruje).
- Słowa kluczowe to nazwy zarezerwowane przez Javę. Zawsze są zapisywane małymi literami.
- Słowo kluczowe `class` rozpoczyna deklarację klasy.
- Zgodnie z przyjętą konwencją wszystkie nazwy klas Javy rozpoczynają się od wielkiej litery, a także zawierają wielką literę na początku każdego następnego wyrazu (np. `NazwaPewnejKlasy`).
- Nazwa klasy Javy to identyfikator — ciąg znaków składający się z liter, cyfr, znaku podkreślenia (`_`) i znaku dolara (`$`), który jednak nie zaczyna się od cyfry i nie zawiera spacji.
- Deklaracja klasy publicznej (`public`) musi być zapisana w pliku o takiej samej nazwie mającym rozszerzenie `.java`.
- Java to język, w którym istotna jest wielkość liter, więc „widzi” różnicę między małymi i dużymi literami.
- Treść deklaracji każdej klasy znajduje się w nawiasach klamrowych (`{ }`).

- Metoda `main` to punkt początkowy każdej aplikacji. Musi się zaczynać w następujący sposób:

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

W przeciwnym razie JVM nie uruchomi aplikacji.

- Metody wykonują zadania i zwracają informacje, gdy je zakończą. Słowo kluczowe `void` wskazuje, że metoda wykonuje zadanie, ale nie zwraca informacji.
- Instrukcje powodują wykonywanie pewnych akcji przez komputer.
- Ciąg znaków umieszcza się w cudzysłowie i nazywa czasem literałem znakowym.
- Standardowy obiekt wyjścia (`System.out`) wyświetla znaki w oknie terminala.
- Metoda `System.out.println` wyświetla w terminalu tekst przekazany jako argument, a następnie przenosi kursor do początku nowego wiersza.

### *Punkt 2.2.1. Kompilacja aplikacji*

- Kompilację programu przeprowadza się poleceniem `javac`. Jeśli program nie zawiera żadnych błędów składniowych, powstanie plik klasy zawierający kod bajtowy reprezentujący aplikację. Kod bajtowy jest interpretowany przez JVM w trakcie wykonywania programu.

### *Punkt 2.2.2. Wykonywanie aplikacji*

- Aby uruchomić aplikację, użyj polecenia `java` i podaj po nim nazwę klasy zawierającej metodę `main`.

### *Podrozdział 2.3. Modyfikacja pierwszego programu*

- Metoda `System.out.print` wyświetla przekazany argument i umieszcza kursor od razu za ostatnim wyświetlonym znakiem.
- Lewy ukośnik (`\`) to znak ucieczki. Java łączy go z następnym znakiem w sekwencję ucieczki — np. kombinacja `\n` reprezentuje znak przejścia do nowego wiersza.

### *Podrozdział 2.4. Wyświetlanie tekstu metodą `printf`*

- Metoda `System.out.printf` (f oznacza „formatowane”) wyświetla dane w postaci sformatowanej.
- Pierwszym argumentem metody `printf` jest ciąg formatujący, który zawiera stały tekst lub określniki formatu. Każdy określnik formatu wskazuje typ danych wyjściowych i jest elementem tymczasowym informującym o konkretnym miejscu wstawienia danych.
- Określnik formatu rozpoczyna się znakiem procentu (`%`), po którym następuje znak definiujący typ danych, np. określnik formatu `%s` oznacza tekst.
- Określnik formatu `%n` to przenośne przejście do nowego wiersza. Nie można go użyć jako argumentu w metodach `System.out.print` lub `System.out.println`, ale metoda `System.out.println` na końcu tekstu również wstawia przejście do nowego wiersza w sposób przenośny między systemami.

### *Punkt 2.5.1. Deklaracja `import`*

- Deklaracja `import` pomaga kompilatorowi odnaleźć klasę używaną przez program.
- Bogaty zestaw predefiniowanych klas Javy jest pogrupowany w pakiety — nazwane grupy klas. Najczęściej mówi się o bibliotece klas Javy lub API Javy.

### ***Punkt 2.5.2. Deklaracja i utworzenie obiektu Scanner w celu pobrania danych z klawiatury***

- Zmienna to miejsce w pamięci komputera, w którym znajduje się wartość używana później przez program. Wszystkie zmienne muszą być deklarowane przed pierwszym użyciem za pomocą nazwy i typu.
- Nazwa zmiennej umożliwia programowi odwołanie się do konkretnego miejsca w pamięci.
- Klasa Scanner (z pakietu `java.util`) umożliwia programowi odczyt danych z zewnątrz. Tworzony obiekt Scanner musi otrzymać jako argument źródło danych.
- Zmienne należy inicjalizować przed pierwszym użyciem przez program.
- Wyrażenie `new Scanner(System.in)` tworzy obiekt Scanner, który odczytuje dane ze standardowego obiektu wejścia (`System.in`).

### ***Punkt 2.5.3. Prośba o wprowadzenie danych***

- Zapytanie prosi użytkownika o podjęcie konkretnej akcji.

### ***Punkt 2.5.4. Deklaracja zmiennej dla liczby całkowitej i pobranie wartości z klawiatury***

- Typ danych `int` służy do deklarowania zmiennych mogących przechowywać wartości całkowite. Typ `int` dopuszcza wartości w zakresie od `-2 147 483 648` do `+2 147 483 647`.
- Wartości typu `int` nie mogą w kodzie źródłowym zawierać spacji, ale ze względu na poprawę czytelności dopuszcza się stosowanie znaków podkreślenia (np. `60_000_000`).
- Typy `float` i `double` dotyczą obsługi liczb zmiennoprzecinkowych, takich jak `-11.19` lub `3.4`.
- Zmienne typu `char` reprezentują pojedyncze znaki, np. dużą literę (`A`), cyfrę (`7`), znak specjalny (np. `*` lub `%`), a także sekwencję ucieczki (np. znak tabulacji `\t`).
- Typy takie jak `int`, `float`, `double` i `char` to typy podstawowe. Typy podstawowe są słowami kluczowymi, więc zapisuje się je małymi literami.
- Metoda `nextInt` klasy `Scanner` pobiera od użytkownika liczbę całkowitą.

### ***Punkt 2.5.6. Użycie zmiennych w obliczeniach***

- Fragmenty instrukcji mające wartości nazywamy wyrażeniami.

### ***Punkt 2.5.7. Wyświetlenie wyniku obliczeń***

- Określnik formatu `%d` stanowi miejsce dla wartości typu `int`.

### ***Punkt 2.5.9. Deklaracja i inicjalizacja zmiennej jako osobne instrukcje***

- Zmienna musi mieć przypisaną wartość, aby mogła być używana przez program.
- Operator przypisania (`=`) umożliwia nadanie wartości zmiennej.

## ***Podrozdział 2.6. Zagadnienia dotyczące pamięci***

- Nazwy zmiennych odpowiadają pewnym obszarom w pamięci komputera. Każda zmienna ma nazwę, typ, rozmiar i wartość.
- Wartość wstawiana do zmiennej zastępuje wartość, która znajdowała się w danym miejscu pamięci wcześniej. Innymi słowy, stara wartość jest tracona.

### Podrozdział 2.7. Operacje arytmetyczne

- Operatorami arytmetycznymi są: + (dodawanie), - (odejmowanie), \* (mnożenie), / (dzielenie) i % (reszta z dzielenia).
- Dzielenie liczb całkowitych zwraca liczbę całkowitą bez części ułamkowej.
- Operator reszty z dzielenia (%) zwraca resztę pozostałą po dzieleniu liczb całkowitych.
- Wyrażenia arytmetyczne w kodzie trzeba pisać w wersji liniowej.
- Jeśli wyrażenie zawiera zagnieżdżone nawiasy, w pierwszej kolejności wyliczane są wartości z najbardziej wewnętrznych nawiasów.
- Java stosuje operatory arytmetyczne w wyrażeniach w ściśle określonej kolejności wskazywanej przez kolejność wykonywania działań.
- Gdy mówimy, że operatory są stosowane od lewej do prawej, mamy na myśli ich łączenie (asocjacyjność). Niektóre operatory są stosowane od prawej do lewej.
- Dodatkowe (zbędne) nawiasy czynią wyrażenia czytelniejszymi.

### Podrozdział 2.8. Podejmowanie decyzji — operatory równości i relacji

- Instrukcja `if` podejmuje decyzję na podstawie spełnienia (prawda lub fałsz) wskazanego warunku.
- Warunki w instrukcji `if` wykorzystują operatory równości (`==` i `!=`) oraz operatory relacji (`>`, `<`, `<=` i `>=`).
- Instrukcja `if` rozpoczyna się od słowa kluczowego `if`, następnie jest umieszczony w nawiasach warunek, a potem treść w postaci dodatkowych instrukcji. Dodatkowe instrukcje, jeśli jest ich więcej niż jedna, muszą znajdować się w nawiasach klamrowych.

## Ćwiczenia do samooceny

- 1.1. Wypełnij puste miejsca w każdym z następujących zdań:
  - a) \_\_\_\_\_ i \_\_\_\_\_ zaczynają i kończą treść każdej metody.
  - b) Instrukcja \_\_\_\_\_ służy do podejmowania decyzji.
  - c) Znaki \_\_\_\_\_ rozpoczynają komentarz jednowierszowy.
  - d) \_\_\_\_\_, \_\_\_\_\_ i \_\_\_\_\_ są nazywane białymi spacjami.
  - e) \_\_\_\_\_ są zarezerwowane przez język Java.
  - f) Aplikacje Javy rozpoczynają działanie od metody \_\_\_\_\_.
  - g) Metody \_\_\_\_\_, \_\_\_\_\_ i \_\_\_\_\_ wyświetlają informacje w oknie terminala.
- 1.2. Określ, które z poniższych zdań są **prawdziwe**, a które **fałszywe**. Jeśli są fałszywe, wyjaśnij dlaczego.
  - a) Komentarze powodują, że w trakcie wykonywania programu komputer wyświetla tekst umieszczony po znakach `//`.
  - b) Wszystkie zmienne muszą mieć wskazany typ w momencie ich deklarowania.
  - c) Java traktuje zmienne `liczba` i `LiczBa` jako identyczne.
  - d) Operator reszty z dzielenia (%) może być stosowany tylko dla operandów będących liczbami całkowitymi.
  - e) Operatory arytmetyczne `*`, `/`, `%`, `+` i `-` mają taki sam priorytet.
  - f) Identyfikator `_` (podkreślenie) użyty samodzielnie jest poprawny w Javie 9.
- 1.3. Napisz instrukcje realizujące każde z wymienionych niżej zadań:
  - a) Zadeklaruj zmienne `c`, `thisIsAVariable`, `q76354` i `number` o typie `int` i zainicjalizuj każdą z nich wartością 0.
  - b) Poproś użytkownika o wpisanie liczby całkowitej.

- c) Pobierz liczbę całkowitą i umieść ją w zmiennej `value` typu `int`. Załóż, że istnieje obiekt `Scanner` o nazwie `input` obsługujący odczyt wartości z klawiatury.
  - d) Wyświetl w oknie konsoli w jednym wierszu tekst "To jest program Javy". Użyj metody `System.out.println`.
  - e) Wyświetl tekst "To jest program Javy" w dwóch wierszach w konsoli. Pierwszy wiersz powinien kończyć się wyrazem `program`. Użyj metody `System.out.printf` i dwóch określników formatu `%s`.
  - f) Jeśli zmienna `number` nie jest równa 7, wyświetl tekst "Zmienna number nie jest równa 7".
- 1.4. Znajdź i popraw błędy w każdym z następujących fragmentów kodu:
- a) 

```
if (c < 7); {
    System.out.println("c jest mniejsze od 7");
}
```
  - b) 

```
if (c => 7) {
    System.out.println("c jest większe lub równe 7");
}
```
- 1.5. Napisz deklaracje, instrukcje lub komentarze, które realizują wszystkie wskazane zadania:
- a) Wskaż, że program będzie wyliczał iloczyn trzech liczb całkowitych.
  - b) Utwórz obiekt `Scanner` w zmiennej `input`, który odczytuje wartości ze standardowego wejścia.
  - c) Poproś użytkownika o wpisanie pierwszej liczby całkowitej.
  - d) Odczytaj pierwszą liczbę całkowitą podaną przez użytkownika i zapamiętaj ją w zmiennej `x` typu `int`.
  - e) Poproś użytkownika o wpisanie drugiej liczby całkowitej.
  - f) Odczytaj drugą liczbę całkowitą podaną przez użytkownika i zapamiętaj ją w zmiennej `y` typu `int`.
  - g) Poproś użytkownika o wpisanie trzeciej liczby całkowitej.
  - h) Odczytaj trzecią liczbę całkowitą podaną przez użytkownika i zapamiętaj ją w zmiennej `z` typu `int`.
  - i) Oblicz iloczyn trzech liczb całkowitych znajdujących się w zmiennych `x`, `y` i `z`, a następnie umieść wynik w zmiennej `result` typu `int`.
  - j) Użyj metody `System.out.printf` do wyświetlenia tekstu "Iloczyn wynosi", po którym pojawi się zawartość zmiennej `result`.
- 1.6. Wykorzystując polecenia z ćwiczenia 2.5, napisz kompletny program, który wylicza i wyświetla iloczyn trzech liczb całkowitych.

## Odpowiedzi do samooceny

- 2.1. a) lewy nawias (`{`), prawy nawias (`}`); b) `if`; c) `//`; d) znaki spacji, nowych wierszy i tabulacji; e) słowa kluczowe; f) `main`; g) `System.out.print`, `System.out.println` i `System.out.printf`.
- 2.2. Odpowiedzi:
- a) Fałsz. Komentarze nie powodują podejmowania przez program jakichkolwiek akcji. Dotyczą dokumentowania kodu i poprawy jego czytelności.
  - b) Prawda.
  - c) Fałsz. Java jest językiem, w którym ważna jest wielkość liter, więc zmienne są różne.
  - d) Fałsz. Operator reszty może być stosowany także dla operandów niebędących liczbami całkowitymi.

- e) Falsz. Operatory \*, / i % mają wyższy priorytet niż operatory + i -.
- f) Falsz. Od Javy 9 \_ (podkreślenie) nie jest już poprawnym identyfikatorem.

2.3. Odpowiedzi:

- a) 

```
int c = 0;
int thisIsAVariable = 0;
int q76354 = 0;
int number = 0;
```
- b) 

```
System.out.print("Wpisz liczbę całkowitą: ");
```
- c) 

```
int value = input.nextInt();
```
- d) 

```
System.out.println("To jest program Javy ");
```
- e) 

```
System.out.printf("%s%n%s%n", "To jest program", "Javy");
```
- f) 

```
if (number != 7) {
    System.out.println("Zmienna number nie jest równa 7");
}
```

2.4. Odpowiedzi:

- a) Błąd polega na użyciu średnika tuż za nawiasem kończącym warunek (c < 7) w if. W efekcie kod poniżej wykona się niezależnie od tego, czy warunek okazuje się prawdziwy.  
Sposób poprawy: usuń średnik po nawiasie.
- b) Błąd polega na użyciu niepoprawnego operatora relacji =>.  
Sposób poprawy: zamień => na >=.

2.5. Odpowiedzi:

- a) *// Oblicz iloczyn trzech liczb całkowitych.*
- b) 

```
Scanner input = new Scanner(System.in);
```
- c) 

```
System.out.print("Wpisz pierwszą liczbę całkowitą: ");
```
- d) 

```
int x = input.nextInt();
```
- e) 

```
System.out.print("Wpisz drugą liczbę całkowitą: ");
```
- f) 

```
int y = input.nextInt();
```
- g) 

```
System.out.print("Wpisz trzecią liczbę całkowitą: ");
```
- h) 

```
int z = input.nextInt();
```
- i) 

```
int result = x * y * z;
```
- j) 

```
System.out.printf("Iloczyn wynosi %d%n", result);
```

2.6. Odpowiedź:

```

1 // Ćwiczenie 2.6. Product.java
2 // Oblicz iloczyn trzech liczb całkowitych
3 import java.util.Scanner; // Program używa klasy Scanner
4
5 public class Product {
6     public static void main(String[] args) {
7         // Utwórz obiekt Scanner, aby pobrać dane wejściowe z klawiatury
8         Scanner input = new Scanner(System.in);
9
10        System.out.print("Wpisz pierwszą liczbę całkowitą: "); // Zapytanie
11        int x = input.nextInt(); // Odczyt pierwszej liczby
12
13        System.out.print("Wpisz drugą liczbę całkowitą: "); // Zapytanie
14        int y = input.nextInt(); // Odczyt drugiej liczby
15
16        System.out.print("Wpisz trzecią liczbę całkowitą: "); // Zapytanie
17        int z = input.nextInt(); // Odczyt trzeciej liczby

```



```

18
19     int result = x * y * z; // Wyliczenie iloczynu
20
21     System.out.printf("Iloczyn wynosi %d%n", result);
22 } // Koniec metody main
23 } // Koniec klasy Product

```

**Wpisz pierwszą liczbę całkowitą: 10**

**Wpisz drugą liczbę całkowitą: 20**

**Wpisz trzecią liczbę całkowitą: 30**

## Ćwiczenia

- 2.7. Wypełnij puste miejsca w następujących zdaniach:
- \_\_\_\_\_ służy do dokumentowania programu i poprawy jego czytelności.
  - Decyzję w języku Java podejmuje się za pomocą \_\_\_\_\_.
  - Operatorami arytmetycznymi o takim samym priorytecie są mnożenie i \_\_\_\_\_.
  - Gdy nawiasy w wyrażeniu są zagnieżdżone, najpierw wykonywany jest \_\_\_\_\_ zestaw nawiasów.
  - Miejsce w pamięci, które w trakcie trwania programu może przechowywać różne wartości, nosi nazwę \_\_\_\_\_.
- 2.8. Napisz instrukcje Javy, które realizują opisane poniżej zadania:
- Wyświetlenie tekstu "Wpisz liczbę całkowitą: " i pozostawienie kursora w tym samym wierszu.
  - Przypisanie wyniku mnożenia zmiennych b i c do zmiennej a typu int.
  - Użycie komentarza do wskazania, że program wylicza wynagrodzenie.
- 2.9. Określ, które z poniższych zdań są **prawdziwe**, a które **falsywe**. Jeśli są falszywe, wyjaśnij dlaczego.
- Operatory Javy są wyliczane od lewej do prawej.
  - Wszystkie wymienione nazwy zmiennych są prawidłowe: `_pod_barem_`, `m928134`, `t5`, `j7`, `jej_sprzedaz$`, `jego_$bilans_laczny`, `a`, `b$`, `c`, `z` i `z2`.
  - Poprawne wyrażenie arytmetyczne w Javie bez nawiasów jest wyliczane od lewej do prawej.
  - Te wszystkie identyfikatory są uznawane za nieprawidłowe: `3g`, `87`, `67h2`, `h22` i `2h`.
- 2.10. Co wyświetli każda z poniższych instrukcji przy założeniu, że  $x = 2$  i  $y = 3$ ?
- `System.out.printf("x = %d%n", x);`
  - `System.out.printf("Wynikiem %d + %d jest %d%n", x, x, (x + x));`
  - `System.out.printf("x =");`
  - `System.out.printf("%d = %d%n", (x + y), (y + x));`
- 2.11. Które z poniższych instrukcji Javy zawierają zmienne o modyfikowanych wartościach?
- `int p = i + j + k + 7;`
  - `System.out.println("zmienne, których wartości są modyfikowane");`
  - `System.out.println("a = 5");`
  - `int value = input.nextInt();`
- 2.12. Przy założeniu, że  $y = ax^3 + 7$ , które z poniższych instrukcji Javy odpowiadają temu równaniu?

- a) `int y = a * x * x * x + 7;`
- b) `int y = a * x * x * (x + 7);`
- c) `int y = (a * x) * x * (x + 7);`
- d) `int y = (a * x) * x * x + 7;`
- e) `int y = a * (x * x * x) + 7;`
- f) `int y = a * x * (x * x + 7);`

2.13. Określ kolejność wykonywania operacji w każdej z następujących instrukcji i wylicz wartość x po każdej operacji:

- a) `int x = 7 + 3 * 6 / 2 - 1;`
- b) `int x = 2 % 2 + 2 * 2 - 2 / 2;`
- c) `int x = (3 * 9 * (3 + (9 * 3 / (3))));`

2.14. Napisz aplikację, która wyświetla w jednym wierszu liczby od 1 do 4, a każda liczba jest oddzielona od poprzedniej spacją. Użyj następujących technik:

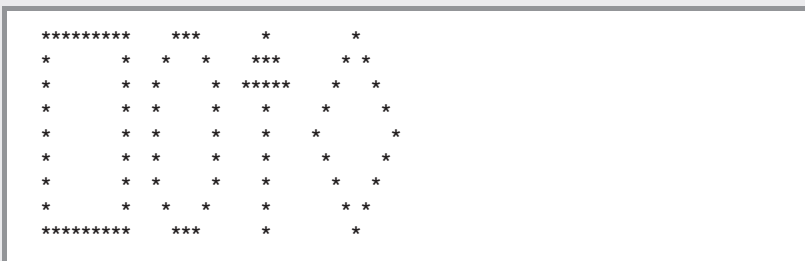
- a) pojedyncza instrukcja `System.out.println;`
- b) cztery instrukcje `System.out.print;`
- c) pojedyncza instrukcja `System.out.printf.`

2.15. (Arytmetyka) Napisz aplikację, która prosi użytkownika o wpisanie dwóch liczb całkowitych, pobiera je i wyświetla ich sumę, iloczyn, różnicę i iloraz. Użyj technik przedstawionych na rysunku 2.7.

2.16. (Porównywanie liczb całkowitych) Napisz aplikację, która prosi o dwie liczby całkowite, pobiera je od użytkownika i wyświetla większą spośród liczb z tekstem "jest większa", natomiast jeśli liczby są sobie równe, wyświetla tekst "Liczby są sobie równe". Użyj technik z rysunku 2.15.

2.17. (Arytmetyka — najmniejszy i największy) Napisz aplikację, która pobiera od użytkownika trzy liczby całkowite i wyświetla ich sumę, średnią, iloczyn, liczbę najmniejszą i największą. Użyj technik z rysunku 2.15. (Uwaga: Wyliczenie średniej w tym zadaniu powinno skutkować uzyskaniem średniej w postaci liczby całkowitej, czyli dla sumy wynoszącej 7 wynikiem powinno być 2, a nie 2,3333...).

2.18. (Wyświetlanie kształtów z gwiazdek) Napisz aplikację, która wyświetla prostokąt, elipsę, strzałkę i romb za pomocą znaków gwiazdki (\*).



2.19. Co wyświetli poniższy kod?  
`System.out.printf("%*n**%*n***%*n****%*n*****%*n");`

2.20. Co wyświetli poniższy kod?  
`System.out.println("");`  
`System.out.println("****");`  
`System.out.println("*****");`  
`System.out.println("*****");`  
`System.out.println("****");`

2.21. Co wyświetli poniższy kod?  
`System.out.print("");`  
`System.out.print("****");`

```
System.out.print("*****");
System.out.print("*****");
System.out.println("***");
```

- 2.22. Co wyświetli poniższy kod?

```
System.out.print("**");
System.out.println("****");
System.out.println("*****");
System.out.print("*****");
System.out.println("***");
```

- 2.23. Co wyświetli poniższy kod?

```
System.out.printf("%s%n%s%n%s%n", "**", "****", "*****");
```

- 2.24. (Najmniejsza i największa liczba całkowita) Napisz aplikację, która odczytuje pięć liczb całkowitych i wyświetla najmniejszą i największą z tej grupy. Wykorzystaj tylko techniki programistyczne poznane w tym rozdziale.

- 2.25. (Liczby parzyste i nieparzyste) Napisz aplikację, która odczytuje liczbę całkowitą i określa, czy jest to liczba parzysta czy nieparzysta. (Wskazówka: Użyj operatora reszty z dzielenia. Liczba całkowita to wielokrotność 2. Każda liczba podzielna przez 2 zwraca resztę równą 0).

- 2.26. (Wielokrotność) Napisz aplikację, która odczytuje dwie liczby całkowite i określa, czy pierwsza liczba jest wielokrotnością drugiej. (Wskazówka: Użyj operatora reszty z dzielenia).

- 2.27. (Wzór szachownicy z gwiazdek) Napisz aplikację, która wyświetli poniższy wzór:

```
* * * * *
 * * * * *
* * * * *
 * * * * *
* * * * *
 * * * * *
* * * * *
 * * * * *
* * * * *
```

- 2.28. (Średnica, obwód i pole koła) To ćwiczenie wybiega nieco w przód. W tym rozdziale omawialiśmy liczby całkowite i typ `int`. Java potrafi również reprezentować liczby zmiennoprzecinkowe definiujące część ułamkową, np. 3,14159. Napisz aplikację, która pobiera od użytkownika promień koła jako liczbę całkowitą, a następnie wylicza średnicę, obwód i pole koła, wykorzystując liczbę zmiennoprzecinkową 3.14159 jako  $\pi$ . Użyj technik z rysunku 2.7. (Uwaga: Możesz też użyć predefiniowanej stałej `Math.PI` dla wartości  $\pi$ . Stała ta jest dokładniejsza niż 3,14159. Klasa `Math` jest zdefiniowana w pakiecie `java.lang`, więc jest zaimportowana automatycznie i nie trzeba importować jej samodzielnie). Użyj następujących wzorów ( $r$  to promień):

$$\begin{aligned} \text{średnica} &= 2r \\ \text{obwód} &= 2\pi r \\ \text{pole} &= \pi r^2 \end{aligned}$$

Nie przechowuj wyników obliczeń w zmiennej. Użyj ich bezpośrednio jako argumentów instrukcji używającej metody `System.out.printf`. Wartościami obwodu i pola będą wartości zmiennoprzecinkowe. Użyj dla nich określnika formatu `%f`. Więcej informacji na ten temat znajdziesz w rozdziale 4.

- 2.29. (Wartość całkowita znaku) Oto kolejny wybieg w przyszłość. W tym rozdziale omawialiśmy liczby całkowite i typ `int`. Java potrafi również reprezentować duże i małe litery, a także znaki specjalne. Każdemu znakowi odpowiada liczba całkowita. To, jakiemu znakowi odpowiada konkretna liczba, zależy od tak zwanego zestawu znaków używanego przez komputer. Aby wskazać pojedynczy znak, wystarczy umieścić go między apostrofami, np. `'A'`.

Aby pobrać liczbę całkowitą odpowiadającą znakowi, można poprzedzić go konstrukcją (int), czyli użyć zapisu:

```
(int) 'A'
```

Taki operator nazywamy operatorem rzutowania. (Więcej informacji na temat operatora rzutowania znajdziesz w rozdziale 3.). Poniższa instrukcja spowoduje wyświetlenie znaku i jego liczbowego odpowiednika:

```
System.out.printf("Znak %c ma wartość liczbową %d\n", 'A',
↳ ((int) 'A'));
```

Wykonanie kodu spowoduje wyświetlenie znaku A i wartości 65 (z zestawu znaków Unicode). Określnik formatu %c służy do wstawiania pojedynczych znaków, np. znaku 'A'.

Korzystając z instrukcji podobnych do tych, które pojawiły się w rozdziale, napisz aplikację, która wyświetli liczbowe odpowiedniki niektórych liter, cyfr i symboli specjalnych. Wyświetl liczbowe wartości znaków A B C a b c 0 1 2 \$ \* + / oraz spacji.

- 2.30. (Oddzielanie cyfr w liczbie całkowitej)** Napisz aplikację, która przyjmuje liczbę składającą się z pięciu cyfr, dzieli ją na poszczególne cyfry i wyświetla je oddzielone od siebie trzema spacjami. Jeśli użytkownik wpisze liczbę 42339, program powinien wyświetlić tekst:

```
4 2 3 3 9
```

Załóż, że użytkownik wpisze liczbę o odpowiedniej liczbie cyfr. Co się stanie, jeśli liczba będzie miała więcej cyfr? Co się stanie, jeśli będzie ich miała mniej? (Wskazówka: Wykonanie tego zadania jest możliwe na podstawie wiedzy zdobytej w tym rozdziale. Trzeba użyć dzielenia i reszty z dzielenia do „wyluskania” odpowiednich cyfr).

- 2.31. (Tabela z kwadratami i sześciانami liczb)** Wykorzystując tylko i wyłącznie techniki poznane w tym rozdziale, napisz aplikację obliczającą kwadraty i sześciانy liczb od 0 do 10 i wyświetlającą je w formie tabelarycznej jak poniżej:

liczba	kwadrat	sześciان
0	0	0
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000

- 2.32. (Wartości dodatnie, ujemne i zero)** Napisz program przyjmujący pięć liczb, sprawdzający je i wyświetlający liczbę wartości dodatnich, ujemnych i wynoszących 0.

## Uczyć świat lepszym

- 2.33. (Kalkulator indeksu BMI)** Indeks masy ciała (BMI) wprowadziliśmy w ćwiczeniu 1.10. Wzór ma postać:

$$BMI = \frac{\text{wagaWKilogramach}}{\text{wzrostWMetrach} \times \text{wzrostWMetrach}}$$

Utwórz kalkulator BMI, który odczytuje od użytkownika wagę w kilogramach i wzrost w metrach, a następnie wylicza i wyświetla wartość BMI. Dodatkowo wyświetl informację o kategorii, do której został przypisany użytkownik na podstawie zakresów ze strony:

<http://bmi-online.pl/>

aby możliwe było łatwe określenie nadwagi lub otyłości.

(Uwaga: W tym rozdziale korzystaliśmy z typu `int` do reprezentacji liczb całkowitych. Ponieważ zastosowanie wagi w metrach powodowałoby bardzo niepoprawne wyniki bez części ułamkowej, warto poprosić użytkownika o wpisanie wagi w centymetrach i podzielić ją przez 100. W rozdziale 3. poznasz typ `double`, który umożliwi reprezentację liczb z uławkami. Gdyby obliczenia wykonywać za pomocą tego typu, wynik również byłby liczbą zmiennoprzecinkową, czyli znacznie dokładniejszą).

- 2.34. (Kalkulator przyrostu populacji na świecie)** Znajdź w internecie informację na temat aktualnej liczby ludzi na świecie i jej rocznego przyrostu. Napisz aplikację, która przyjmie te dwie wartości, a następnie wyświetli przewidywaną liczbę ludzi na świecie po jednym, dwóch, trzech, czterech i pięciu latach.
- 2.35. (Kalkulator oszczędności dzięki carpoolingowi)** Odwiedź kilka stron WWW dotyczących tematu carpoolingu. Napisz aplikację, która oblicza koszt codziennych dojazdów, abyś mógł ustalić, ile pieniędzy zaoszczędziłbyś dzięki zmianie sposobu dojazdu na carpooling, który dodatkowo redukuje emisję gazów cieplarnianych i zmniejsza korki. Aplikacja powinna pobrać od użytkownika następujące informacje i wyświetlić dzienny koszt dojazdu do pracy:
- łączna liczba kilometrów przejeżdżanych każdego dnia;
  - koszt litra paliwa;
  - średnie spalanie;
  - opłaty parkingowe w przeliczeniu na jeden dzień;
  - inne opłaty w przeliczeniu na jeden dzień.





## A

- abstrakcyjne
  - implementacje, 779
  - klasy, 516, 527
  - klasy nadrzędne, 490, 494
  - metody, 527
- adnotacja
  - @FunctionalInterface, 832
  - @FXML, 208, 589–593, 601, 629, 1331
  - @Override, 456
  - @XMLElement, 731
- adres IP, 83
- aktualizacja
  - JavaFX, 643
  - plików sekwencyjnych, 713
- algorytm wyszukiwania liniowego, 887
- algorytmy, 166
  - $O(n)$ , 890
  - $O(n^2)$ , 891
  - równoległe, 908
  - sortowania, 887, 896, 909
  - wyszukiwania, 886, 909
- alokacja pamięci, 946
- analiza zależności, 1352
- AnchorPane, 607
- Android, 70
- animacja, 997, 1028
  - klatka po klatce, 1034, 1051
  - Timeline, 1031, 1051
  - Transition, 1026, 1051
- anonimowe klasy wewnętrzne, 595, 601
- Apache Software Foundation, 69
- API
  - Date/Time, 407, 1132
  - Java Database Connectivity, 1157
  - Javy, 102, 280
  - logów, 1388
  - procesów, 1389
  - przejścia przez stos, 556
  - skórek, 1385
  - współbieżności, 1066, 1149
- aplety Javy, 1393
- aplikacja
  - DeitelMessenger, 1315
  - do przechowywania kontaktów, 1194
  - do przeglądania okładek, 625, 630
  - do wyliczania napiwków, 578
  - kalkulator procentu składanego, 237, 268
  - MathTutor, 1363
  - Painter, 79, 608
  - rzut kostką sześcienną, 295
  - sprawdzanie salda płatności, 709
  - StudentPoll, 340
  - suma liczb parzystych, 236
  - TicTacToeServer, 1301
  - wyświetlenie tekstu i obrazka, 150, 573, 598
  - wyświetlenie wiersza tekstu, 92
- aplikacje
  - testowanie, 78
  - współbieżne, 1066
- architektura
  - MVC, 582
  - wtyczek, 1364
- argument, 132
  - sweepFlag, 1049
- artefakty modułu, 1337
- asembler, 64
- asercje, 536, 560, 566
  - warunki końcowe, 560
  - warunki wstępne, 560
- asynchroniczne
  - obliczanie elementów ciągu, 1139
  - wykonywanie obiektów Runnable, 1136
- atrybut, 67
  - encoding, 1004
  - version, 1004
- AudioClip, 1019

- automatyczne
  - inkrementowanie, 1159
  - odgadywanie typu, 746
  - odnajdowanie sterowników, 1175
  - opakowywanie, 675, 808, 927, 934, 952, 962
  - pakowanie, 742, 777
  - rozpakowywanie, 742, 777, 931
  - uzupełnianie, 1217, 1234
    - identyfikatorów, 1235
    - poleczeń JShell, 1236
  - wypakowanie, 927
  - zwalnianie zasobów, 566
- B**
- baza danych, 63, 1156
  - books, 1158, 1207
  - Java DB, 1171, 1208
- bazy danych
  - konfiguracja, 1171, 1208
  - odpytywanie, 1178, 1209
  - połączenie, 1173, 1175, 1209
  - protokół komunikacji, 1175
  - relacyjne, 1206
  - sterownik, 1175
  - tabele, 1158
  - tworzenie, 1172
- bezpieczeństwo
  - typów, 745, 920, 927
  - wątkowe, 1091
- bezpieczne programowanie, 43
- bezpółłączeniowa transmisja, 1293
- bezpośrednie wywołanie metody, 506
- białe spacje, 93, 113
- biblioteka
  - ControlsFX, 1019
  - klas Javy, 74, 280
- biblioteki JavaFX, 635
- big data, 63
- BigDecimal, 432, 433, 441
- binarne drzewo
  - sortowania, 971
  - wyszukiwania, 966, 973, 975
- bity, 61
- blok instrukcji, 113
  - catch, 341–543, 549, 564
  - finally, 542, 549, 565
  - try, 542, 564
- blokada
  - monitora, 1075, 1094
  - na wyłączność, 1098
  - ponownego wejścia, 1099
- blokady wzajemne, 1069
- bloki, 173, 215
- błędy
  - kompilacji, 94, 507, 1345
  - logiczne, 75, 1084, 1243, 1404
    - fatalne, 174
    - niefatalne, 174
  - programistyczne, 76, 97, 103, 114, 136, 174, 178, 181, 182, 190, 191, 236, 288, 289, 292, 330, 331, 335, 416, 425, 429, 456, 472, 491, 506, 508, 521, 543, 548, 549, 553, 656, 684, 704, 746, 775, 854, 922, 938, 958, 959, 1095, 1167, 1169, 1177, 1178, 1183, 1286, 1294
  - składniowe, 94, 115, 174, 1404
  - synchroniczne, 545
- BorderPane, 202, 204, 607, 609, 869
  - konfiguracja, 611
  - korzeń grafu, 611
- brak synchronizacji, 1084
  - dostępu, 1080
- bufor, 728, 963
  - niesynchronizowany, 1090
  - o określonym rozmiarze, 1103
- buforowane strumienie znakowe, 729
- buforowanie, 727, 733
- bufory
  - o określonym rozmiarze, 1101, 1145
  - wieloelementowe, 1091
- Button, 581, 597, 599
- C**
- Canvas, 202
- CERT, 43
- Character, 671, 687
- chmura, 86
- ciąg Fibonacciego, 856, 1119, 1147
  - obliczenia asynchroniczne, 1139
  - obliczenia synchroniczne, 1139
- cieniowanie
  - kształtów, 1053
  - Phonga, 1053
- Circle, 621
- CLDR, Common Locale Data Repository, 1389
- ControlsFX, 1019



CSS, Cascading Style Sheets, 571, 973,  
 998  
 programowe wczytywanie, 1006,  
 1047  
 sterowanie czcionkami, 1046  
 styl GUI, 1046  
 stylowanie  
 dwuwymiarowych kształtów,  
 1009, 1048  
 obiektu Arc, 1011  
 obiektu Circle, 1011  
 obiektu Ellipse, 1011  
 obiektu Line, 1010  
 obiektu Rectangle, 1011  
 cykl życia wątku, 1066, 1142  
 cykliczność bufora, 1106  
 czas działania metod, 1130, 1148  
 czyszczenie sesji JShell, 1225  
 czytnik ekranowy, 634

## D

dane  
 modyfikowalne, 1074  
 niezmiennie, 1074  
 trwałe, 698  
 współdzielone i modyfikowane,  
 1135  
 datagramy, 1293, 1317  
 DB2, 1176  
 DBMS, Database Management System,  
 1156, 1206  
 debugger, 1403  
 polecenie  
 clear, 1414  
 cont, 1404  
 next, 1410  
 print, 1404, 1408  
 run, 1404  
 set, 1408  
 step, 1410  
 step up, 1410  
 stop, 1404  
 watch, 1412  
 punkty wstrzymania, 1404  
 sterowanie wykonywaniem, 1410  
 tryb wstrzymania, 1406  
 debugowanie, 66  
 scenek, 636  
 definicja GUI VideoPlayer, 1021  
 definiowanie dwuwymiarowych  
 kształtów, 1007, 1048  
 deklaracja  
 dostawcy usług, 1370  
 implementacji, 524  
 importu, 102, 116, 138  
 importu FXML, 1004  
 interfejsu, 510  
 interfejsu Payable, 513  
 klasy, 94, 115, 130, 157, 402, 715  
 klasy abstrakcyjnej, 491  
 konstruktora, 140  
 metody, 96, 1242  
 abstrakcyjnej, 491  
 przeciążonej, 306  
 modułu, 1328, 1325, 1341–1343  
 nowych rodzajów wyjątków, 559,  
 566  
 obiektu Scanner, 103  
 package, 957  
 tablicy, 330  
 typu enum, 421  
 XML, 1004  
 zmiennej, 104, 117, 492  
 jawna, 1221, 1232  
 niejawna, 1224, 1225  
 dekrementacja, 196, 216  
 deserializacja, 698, 1387  
 danych, 719, 732  
 deskryptor modułu, 1323  
 diagram  
 aktywności, 168, 170, 213  
 dla instrukcji do...while, 241  
 dla instrukcji for, 235  
 dla instrukcji if, 171  
 dla instrukcji if...else, 172  
 dla instrukcji switch, 247  
 dla instrukcji while, 178, 179  
 instrukcji switch, 246  
 klas UML, 136, 147, 159  
 dla hierarchii Payable, 513  
 dla hierarchii Employee, 493  
 Account, 136  
 stanu  
 dla cyklu życia wątku, 1067  
 zawierający interfejs, 512  
 związków encji, 1160  
 dodanie  
 etykiety, 153, 575  
 ImageView, 576  
 kontroltek, 621, 626  
 liczb całkowitych, 102  
 obrazu do folderu, 574  
 przycisków, 613

- dokument
    - JEP, 1325
    - JSR, 1325
  - dokumentacja, 1236
    - metody, 1238
    - pól publicznych, 1238
  - domknięcie przejściowe, 1362
  - dopasowanie
    - do wzorca, 1163
    - do dowolnej liczby wystąpień, 683
  - dostawcy usług, 1363
    - deklaracja, 1370
    - implementacja, 1369
    - interfejs, 1368
    - wczytywanie, 1366
  - dostęp
    - do baz danych, 1155
    - do pakietów modułu, 1330
    - do pakietu, 1330
    - do składowych, 407, 439
    - do tablicy, 329
    - do współdzielonego bufora, 1105
    - do współdzielonych
      - i modyfikowanych danych,  
1080, 1096
    - do zmiennych, 425
    - na poziomie pakietu, 430, 441, 451
    - sekwencyjny, 772
  - dostrajanie wydajności, 66
  - dowiązanie
    - właściwości, 619, 641
    - właściwość do właściwości, 624
    - dynamiczne, 490, 505
    - późne, 505
  - drzewa, 764, 975
    - binarne, 975
    - wyszukiwania, 966, 973, 975
  - duże
    - O, 891, 908, 910
    - zrównoleglenie, 908
  - dynamiczna alokacja pamięci, 946, 973
  - dynamiczne struktury danych, 944, 973
  - dyrektywa
    - exports, 1329
    - exports...to, 1329, 1345
    - opens, 1329
    - opens...to, 1329
    - provides...with, 1329,, 1370
    - requires, 1328
    - requires transitive, 1328
    - uses, 1329, 1368
  - dziedziczenie, 67, 447, 474, 477, 522
    - implementacji, 522, 530
    - interfejsu, 522, 530
    - jednobazowe, 448, 477
    - wielobazowe, 477
  - dziel i rządź, 280
  - dzielenie
    - całkowite, 182
    - przez zero, 539, 564
- E**
- Eclipse, 51
  - Eclipse Foundation, 69
  - edycja fragmentów, 1226, 1248, 1255
  - edytor zewnętrzny, 1247, 1249
  - edytowanie pól, 587
  - efektywnie finalne zmienne lokalne, 562
  - eksport pakietu, 1341
  - elastyczność, 524
  - element, 1024
    - Box, 1043
    - children, 1004
    - Circle, 1048
    - Cylinder, 1043
    - enum, 601
    - Line, 1048
    - Path, 1049
    - Polygon, 1049
    - Polyline, 1049
    - Rectangle, 1048
    - zerowy, 329
  - eliminacja duplikatów, 970
  - enkapsulacja, 67, 1324, 1330, 1345
    - zasobów, 1354
  - etykiety
    - dodanie, 575
    - konfiguracja, 575
    - ewaluacja leniwa, 792
- F**
- fabryka komórek, 631, 642
  - FIFO, 963
  - filtrowanie, 793, 835
    - danych serializowanych, 1387
    - strumienia, 809, 838
    - tekstów, 812
    - wyników zapytania, 1188
  - filtry, 727

finalne  
   klasy, 528  
   metody, 528  
 FlowPane, 607  
 folder  
   aplikacji, 78  
   instalacyjny JDK, 50, 1171  
 format  
   tekstowy, 1378  
   TIF, 1386  
   XML, 718  
 formatowanie liczb, 581  
   zmiennoprzecinkowych, 146, 239  
 formaty multimediów, 1019  
 fragmenty kodu, 1217  
 fraktale, 864, 878  
 framework  
   Executor, 1070  
   Fork/Join, 1140, 1149  
   GStreamer, 638  
   Image, 1386  
   kolekcji, 740, 772, 777  
 funkcja  
   linear-gradient, 1011  
   radial-gradient, 1049  
   rgba, 1010, 1048  
 funkcje  
   czyste, 804  
   mieszające, 767, 768  
 funkcjonalności  
   Javy 8, 32  
   Javy 9, 32, 1377  
   obiektu GraphicsContext, 1041  
   pulpitu, 1386  
 FXML, FX Markup Language, 150,  
 572, 597, 1002  
   definiowanie dwuwymiarowych  
   kształtów, 1007, 1048  
   GUI, 1047  
   referencja do pliku CSS, 1047

## G

generowanie  
   klasy kontrolera, 588, 614, 622,  
   627  
   liczb losowych, 294, 315  
   tablicy liczb losowych, 890  
 GitHub, 69  
 głębia, 1053

gniazda  
   datagramowe, 1274, 1275, 1316  
   sietciowe, 1274  
   strumieniowe, 1274, 1278, 1281,  
   1316  
 gra  
   działo, 1061, 1062  
   SpotOn, 1060  
   w kółko i krzyżyk, 1301, 1307,  
   1311–1315  
   wyścigi konne, 1061  
 gradient, 1048  
   liniowy, 1049  
 graf, 598  
   modułu z cyklem, 1347  
   układanie węzłów, 606, 639  
   zależności modułu, 1325, 1335,  
   1343, 1346, 1348, 1357  
 graficzny interfejs użytkownika, GUI,  
 150, 569, 605  
 grafika, 997, 1386  
 GridPane, 580, 583, 599, 607, 621  
   dodawanie kontrolerek, 584  
   dodawanie wierszy, 583  
   dostosowanie rozmiaru, 585  
   dostosowanie rozmiaru kolumn, 586  
   edytowanie pól, 587  
   konfiguracja, 621  
   konfiguracja odstępów, 586  
   zmiana rozmiaru przycisku, 586  
 grupowanie, 821, 826, 840  
   podwyrażeń, 108  
 GStreamer, 638  
 GUI, Graphical User Interface, 150,  
 570, 605  
 GUI VideoPlayer, 1020

## H

harmonogramowanie wątków, 1068  
 HBox, 608  
 hierarchia  
   danych, 61, 62  
   dziedziczenia klas, 450, 462, 485,  
   546  
   klas, 492  
   kształtów, 450  
   MyShape, 525, 526  
   wyjątków, 546, 565  
 HTML, HyperText Markup Language,  
 83

- I
- IDE, Integrated Development Environment, 75, 92, 571
    - obsługa JShell, 1256
  - identyfikatory, 131
  - ImageView, 597
    - dodanie, 576
    - konfiguracja, 576
    - zmiana rozmiaru elementu, 577
  - implementacja
    - algorytmu sortowania przez scalanie, 904
    - bufora o określonym rozmiarze, 1103
    - ChangeListener, 596
    - dostawcy usług, 1369
    - dziedziczenia, 522
    - FactorialCalculator, 855
    - interfejsu Collection, 772
    - interfejsu Payable, 515
    - interfejsu Runnable, 1070
    - iteracji sterowanej licznikiem, 181
    - iteracji sterowanej znacznikiem, 186
    - klasy List, 947
    - Map, 767
    - sortowania przez wstawianie, 901
    - sortowania przez wybieranie, 897
    - synchronizacji, 1111
    - uogólnionej klasy, 927
    - wyszukiwania binarnego, 893
    - wyszukiwania liniowego, 888
  - implementacje abstrakcyjne, 772, 779
  - import
    - klas, 1245, 1246
    - statyczny, 428, 440
      - na żądanie, 428
      - pojedynczy, 428
    - typów, 958
    - typu na żądanie, 958
  - indeks, 329
  - informacje
    - debugowania, 539
    - o plikach, 701, 724, 730
    - o wersji, 1325
  - inicjalizacja
    - obiektów, 139, 141, 159, 629
    - tablicy, 332
    - zmiennej, 106, 117, 185
  - inkrementacja, 196, 216
  - instalacja JDK 9, 1218
  - instrukcja, 97, 116
    - assert, 560
    - break, 245, 251, 269
    - case, 245
    - catch, 564
    - continue, 251, 270
    - default, 245, 246
    - DELETE, 1170, 1208
    - do...while, 169, 240, 269
    - for, 169, 231, 233, 267
      - rozszerzona, 263, 346
    - goto, 167
    - if, 111, 118, 170, 171, 214, 1229
    - if...else, 169–172, 177, 195, 215
    - INSERT, 1168, 1208
    - ON, 1167
    - ORDER BY, 1165, 1166, 1207
    - return, 133, 158
    - switch, 169, 245, 269
    - throw, 552
    - throws, 544, 564
    - try, 341, 544, 562, 566
    - UPDATE, 1169, 1208
    - WHERE, 1163, 1166, 1207
    - while, 169, 177, 189, 215, 233
  - instrukcje
    - importu, 591, 1390
    - iteracji, 169, 214, 260
    - sekwencji, 260
    - sterujące, 167
    - wielowierszowe, 1226
    - wyboru, 169, 214, 260
    - złożone, 113
  - integralność, 1324
  - IntelliJ IDEA Community Edition, 51
  - interakcja klienta i serwera, 1281
    - bezpółłączeniowa, 1293, 1317
  - interfejs, 486
    - AutoCloseable, 519, 566
    - BinaryOperator<T>, 805
    - Buffer, 1084
    - CachedRowSet, 1191
    - Callable, 1135, 1148
    - CallableStatement, 1204
    - CarbonFootprint, 533
    - ChangeListener, 520, 582, 591, 599
    - Collection, 741–744, 777
    - Comparable<T>, 519, 923
    - Comparator, 520, 754
    - Condition, 1109, 1110, 1145
    - Consumer<T>, 805
    - DataInput, 733

- DataOutput, 733
  - DirectoryStream, 700, 730
  - Executor, 1143
  - ExecutorService, 1070, 1143
  - Function<T, R>, 805
  - Future, 1135, 1148
  - IntStream, 788, 801
  - Iterator, 746
  - List, 741–745, 772, 777
  - Lock, 1109, 1145
  - Map, 741, 767, 778
  - Path, 700
  - Payable, 512–517
  - Predicate<T>, 805, 840
  - PreparedStatement, 1211
  - ProblemProvider, 1367
  - Queue, 741, 763, 772, 778
  - RowSet, 1190, 1210
  - RowSetFactory, 1192
  - Runnable, 519, 1070, 1143
  - Serializable, 519
  - Set, 741, 772, 778
  - SortedMap, 767, 778
  - SortedSet, 766, 778
  - Stream, 807, 838, 1383
  - Supplier<T>, 805, 1149
  - TableModel, 1178–1181
  - UnaryOperator<T>, 805
  - Worker, 1119, 1147
  - interfejsy
    - API Javy, 518
    - aplikacji, 578, 611
    - dodawanie metod, 520
    - domyślne metody, 519
    - dostawcy usług, 1363, 1365, 1368
    - dziedziczenie, 522
    - funkcyjne, 520, 804, 837
    - graficzne aplikacje, 149, 626, 1020, 1050, 1121, 1386
    - implementacja, 511, 515
    - klasy abstrakcyjne, 511, 520
    - metody prywatne, 521
    - metody statyczne, 520
    - nasłuchiwanie zdarzeń, 519, 581
    - NIO, 730
    - polimorficzne, 494
    - prywatne metody, 530
    - rozszerzenia, 529, 530
    - stosowanie, 509, 529
    - tworzenie, 509, 529
    - znacznikowe, 511
    - internet rzeczy, IoT, 74, 84
    - interpreter, 65, 996
    - iOS, 70
    - IoT, Internet of Things, 74, 84
    - IP, Internet Protocol, 83
    - iteracja, 177, 860
      - nieokreślona, 184
      - sterowana licznikiem, 179, 215, 230, 267
      - sterowana znacznikiem, 183, 186, 216
      - wewnętrzna, 789, 834
      - zewnętrzna, 786, 833
- J**
- Java, 74
  - Java DB, 1171, 1176
  - Java Micro Edition, 58
  - Java Standard Edition, 57
  - JavaFX, 570, 571, 605
    - aktualizacja, 636, 643
    - animacje, 997
    - API skórek, 1385
    - audio, 1019
    - biblioteki, 635
    - dotychczasowe możliwości, 634, 643
    - funkcjonalności, 596
    - grafika, 997
    - komponenty układu, 607, 608
    - narzędzie Scene Builder, 571, 597
    - struktura okna aplikacji, 572, 598
    - tworzenie animacji, 1031, 1034
    - tworzenie interfejsu graficznego, 720
    - ułatwienia dostępu, 634
    - ułożenie komponentów, 580
    - wideo, 997, 1019
    - wielowątkowość, 1117, 1146
    - własne kontrolki, 635
  - JavaFXPorts, 635
  - jawne
    - deklarowanie zmiennych, 1221, 1232
    - rzutowanie, 488, 507, 926
  - jawnie nazwane moduły, 1345
  - JAXB, 714, 731
  - jądro systemu Linux, 70
  - JCP, Java Community Process, 43
  - JDBC, 1155, 1157, 1211
  - JDK, 1171

- jednostka, 280, 312
  - arytmetyczno-logiczna, 60
  - logiczna komputera, 59
  - pamięci, 60
  - procesora, 60
  - przechowywania danych, 61, 698
  - wejściowa, 59
  - wyjściowa, 59
- JEP, JDK Enhancement Proposal, 42
- język
  - Ada, 71
  - Basic, 71
  - C, 72
  - C#, 72
  - C++, 72
  - COBOL, 72
  - Fortran, 72
  - FXML, 597
  - HTML, 83
  - Java, 74
  - JavaScript, 72
  - Objective-C, 72
  - Pascal, 72
  - PHP, 73
  - Python, 73
  - Ruby on Rails, 73
  - Scala, 73
  - Simple, 990
  - SMS, 696
  - SQL, 1156, 1162
  - Swift, 73
  - UML, 68
  - Visual Basic, 73
  - XML, 597
- języki
  - assemblerowe, 64
  - maszynowe, 64, 77
  - wysokiego poziomu, 64
- JIT, Just-In-Time, 77
- JPMS, Java Platform Module System, 1323
- JShell, 1216
  - automatyczne uzupełnianie poleceń, 1234, 1236
  - błędy kompilacji, 1221
  - czyszczenie sesji, 1225
  - deklarowanie metod, 1242
  - edycja fragmentów kodu, 1226
  - import klas, 1245
  - instrukcje wielowierszowe, 1226
  - kończenie sesji, 1229
  - moduły, 1384
  - modyfikacja procesu uruchamiania, 1254
  - modyfikowanie obiektów, 1233
  - odkrywanie składowych obiektu, 1240
  - opcje, 1254
  - polecenia, 1250
  - przeglądanie
    - dokumentacji metody, 1238
    - dokumentacji pól publicznych, 1238
    - parametrów metod, 1238
    - przeciążeń metod, 1240
    - składowych klas, 1236
  - skrótów klawiaturowe, 1255
  - testowanie uaktualnionej metody, 1244
  - tryb informacji zwrotnych, 1226, 1253
  - tworzenie
    - klasy, 1231
    - nazwy zmiennej, 1233
    - obiektów, 1232
    - obiektu Scanner, 1230
  - uruchamianie sesji, 1219
  - uzyskiwanie pomocy, 1251
  - wejście danych, 1229
  - wyjątki, 1244
  - wykonywanie poleceń, 1219
  - wylistowanie fragmentów, 1223
  - zachowanie ustawień edytora, 1249
- JShell Edit Pad, 1249
- JSR, Java Specification Request, 42
- JVM, Java Virtual Machine, 75

## K

- kalkulator procentu składanego, 237, 268
- kanwa, 1037
- kaskadowe arkusze stylów, CSS, 571, 973, 998
- katalog bin, 50
- klasa, 65, 94, 401
  - AbstractTableModel, 1178
  - Account, 129, 131, 139, 714
  - Accounts, 715
  - AccountTest, 133, 141, 158
  - AdditionProblemProvider, 1369
  - AddressBookController, 1200
  - Analysis, 193
  - AnimationTimer, 1034, 1036, 1051

- Application, 580, 599
- ArcTo, 1016, 1049
- ArrayBlockingQueue, 1091, 1103, 1144
- ArrayList, 382, 515, 744
- Arrays, 369, 381
- ArrayWriter, 1077
- AudioClip, 1019
- AutoPolicy, 248, 269
- AutoPolicyTest, 250
- BankAccount, 449
- BasePlusCommissionCompensationModel, 523
- BasePlusCommissionEmployee, 458, 466, 470, 487, 489, 502
- BigDecimal, 431, 441, 591, 854
- BigIntegers, 854
- Bindings, 1025
- BlockingBuffer, 1091
- BlockingBufferTest, 1092
- Boolean, 742
- BufferedInputStream, 728, 733
- BufferedOutputStream, 727, 733
- BufferedReader, 733
- BufferedWriter, 732
- Byte, 742
- ByteArrayInputStream, 728, 733
- Canvas, 869, 1037
- Card, 342, 757
- Character, 671, 687, 742
- CharArrayReader, 729, 733
- CharArrayWriter, 729
- CircularBuffer, 1105
- Client, 1287, 1297
- ClosePath, 1016, 1049
- Collections, 743, 751, 752, 777
- Collectors, 840
- ColorChooserController, 622, 641
- CommissionCompensationModel, 523
- CommissionEmployee, 454, 469, 500
- CommissionEmployeeTest, 456
- Comparator, 754
- CompletableFuture, 1136, 1139, 1149, 1391
- Consumer, 1086
- CoverViewerController, 628, 633, 642
- CreateTextFile, 705
- CreditInquiry, 709
- DataInputStream, 727
- DataOutputStream, 727
- Date, 418
- DeckOfCards, 343–345
- DirectoryChooser, 720, 732
- DisplayQueryResultsController, 1185
- Double, 742
- DoubleProperty, 641
- DrawLines, 209
- DrawLinesController, 207, 263
- DrawRainbowController, 375
- DrawRandomLinesController, 435
- DrawSmileyController, 309
- DriverManager, 1175
- Duration, 1132
- Employee, 419, 449, 493, 813
- EmployeeTest, 420, 427
- Error, 565
- Exception, 546, 565
- Executors, 1070
- FadeTransition, 1030, 1051
- FibonacciNumbersController, 1121
- File, 732
- FileChooser, 720, 732
- FileChooserTest, 721
- FileChooserTestController, 722
- FileReader, 733
- Files, 700, 729, 730
- FillTransition, 1030, 1051
- FilterInputStream, 727
- FilterOutputStream, 727
- FindPrimesController, 1125, 1127
- Float, 742
- Formatter, 730
- FractalController, 870, 873
- FXCollections, 626, 642
- FXMLLoader, 582, 597, 600, 601
- GradeBook, 337, 351, 360
- GradeBookTest, 354, 364
- GraphicsContext, 308, 1037
- HashMap, 767, 771
- HourlyCompensationModel, 523
- HourlyEmployee, 498, 499
- ImagePattern, 1052
- ImageTextCell, 631, 642
- InputStreamReader, 729
- Instant, 1132
- Integer, 742
- IntStream, 833
- Invoice, 513, 514
- JEditorPane, 1275
- kontenera, 573

- klasa
- kontrolera, 573
  - LineNumberReader, 729, 733
  - LinkedList, 747, 751
  - List, 947, 950, 956
  - List<E>, 962, 964
  - ListCell, 630
  - ListNode, 950
  - ListTest, 950
  - ListView, 630
  - Loan, 449
  - Long, 742
  - MashMap, 778
  - Matcher, 676, 683–685, 689, 1378
  - Math, 282, 313, 1237
  - MathTutor, 1366
  - Media, 1019, 1049
  - MediaPlayer, 1019, 1049
  - MediaView, 1019
  - MediaViewer, 1049
  - MoveTo, 1016, 1049
  - MultipleSelectionModel, 630
  - MultiplicationProblemProvider, 1371
  - MyLine, 434
  - Node, 640
  - NumberFormat, 432, 441, 581, 591, 599, 601
  - Object, 473, 478
  - OutputStreamWriter, 729
  - Painter, 614
  - PainterController, 614, 640
  - ParallelTransition, 1030, 1051
  - Path, 1012, 1049
  - PathElement, 1016, 1049
  - Paths, 700, 729, 730
  - PathTransition, 1031, 1051
  - Pattern, 676, 685, 688
  - Person, 1194
  - PersonQueries, 1196, 1199
  - PhongMaterial, 1053
  - PipedInputStream, 727
  - PipedOutputStream, 727
  - PipedReader, 729, 733
  - PipedWriter, 729, 733
  - Platform, 1119
  - Polygon, 1012, 1049
  - Polyline, 1012, 1049
  - PolyShapesController, 1014, 1049
  - PreparedStatement, 1193, 1211
  - PrintStream, 727, 733
  - PrintTask, 1071
  - PrintWriter, 707
  - PriorityQueue, 763, 764, 778
  - Producer, 1085
  - Queue<E>, 964
  - RadioButton, 640
  - RandomAccessFile, 727
  - Reader, 733
  - ReadServerFile, 1277
  - ReentrantLock, 1109
  - ResultSetTableModel, 1178
  - RotateTransition, 1030, 1051
  - RowSetProvider, 1192, 1210
  - SalariedCompensationModel, 523
  - ScaleTransition, 1031, 1051
  - Scanner, 103, 117, 158
  - ScheduledService, 1119, 1147
  - SecureRandom, 294, 298, 797, 828
  - SequenceInputStream, 733
  - SequentialTransition, 1031, 1051
  - Server, 1281, 1294
  - ServiceLoader, 1363, 1367
  - Shape, 449
  - Shape3D, 1042
  - SharedArrayTest, 1078
  - SharedBufferTest, 1088
  - SharedBufferTest2, 1115
  - Short, 742
  - SimpleArray, 1076, 1080
  - Slider, 641
  - Stack, 927
  - Stack<E>, 932, 961
  - StackTest, 962
  - StreamTokenizer, 987
  - String, 651, 657, 686
  - StringBuilder, 664, 668, 687
  - StringJoiner, 809
  - StringProperty, 641
  - StringReader, 729, 734
  - StringWriter, 729
  - StrokeTransition, 1030, 1051
  - Student, 175, 449
  - StudentTest, 177
  - SwingNode, 1210
  - SynchronizedBuffer, 1095–1099, 1112
  - System, 757
  - TableRowSorter, 1210
  - Task, 1119, 1147
  - Thread, 1071
  - ThreeDimensionalShapesController, 1043
  - Throwable, 546, 558, 565
  - TicTacToeClient, 1308



- TicTacToeServer, 1301
- Time, 405, 410, 438
- TimeComparator, 754
- TimeData, 1139
- TipCalculator, 589, 601
- TipCalculatorController, 591–594, 601
- ToggleGroup, 640
- Transform, 1017
- Transition, 1051
- TransitionAnimationsController, 1028, 1051
- Tree, 969
- Tree<E>, 970
- TreeMap, 767
- TreeNode, 968
- TreeNode<E>, 970
- TreeSet, 765
- UnsynchronizedBuffer, 1088
- ValidateInput, 679
- VideoPlayerController, 1022, 1050
- Welcome, 1335
- Writer, 733
- klasy
  - abstrakcyjne, 490, 491, 511, 516, 520, 527
  - anonimowe wewnętrzne, 595, 601
  - do wielokrotnego stosowania, 956
  - finalne, 508, 528
  - java.io, 726, 733
  - konkretne, 490, 496, 500
  - nadrzędne, 448–452, 477, 487, 490, 506, 509
    - bezpośrednie, 448
    - pośrednie, 448
  - odnoszące się do samych siebie, 945, 973
  - otoczkowe, 671, 742, 777
  - pochodne, 448
  - składowe statyczne, 424
  - sterujące, 133
  - uogólnione, 917, 939, 950
  - właściwości, 619
  - znaków, 677
- klauzula, *Patrz* instrukcja
- klient, 1316
- klucz, 767
  - główny, 1157, 1159
  - obcy, 1160
  - rekordu, 705
- klucze unikatowe, 767
- kod
  - bajtowy, 75
  - zwalniający zasoby, 562
- kodowanie UTF-8, 1389
- kolejka, 763, 944, 973
  - działania, 963
  - zastosowania, 963
- kolejność
  - wykonywania działań, 108
  - wykonywania operatorów, 198, 258, 1395, 1396
  - wyliczania wartości, 110
- kolekcja, 739, 740, 777
  - ArrayBlockingQueue, 1118
  - ConcurrentHashMap, 1118
  - ConcurrentLinkedDeque, 1118
  - ConcurrentLinkedQueue, 1118
  - ConcurrentSkipListMap, 1118
  - ConcurrentSkipListSet, 1118
  - CopyOnWriteArraySet, 1118
  - CopyOnWriteArraySet, 1118
  - DelayQueue, 1118
  - LinkedBlockingDeque, 1118
  - LinkedBlockingQueue, 1118
  - LinkedTransferQueue, 1118
  - PriorityBlockingQueue, 1118
  - SynchronousQueue, 1118
- kolekcje
  - niemodyfikowalne, 743
  - niezmiennicze, 772, 779
  - synchronizowane, 743, 771, 779
  - uogólnione, 741
  - współbieżne, 1116, 1146
- kolizja, 767, 1053
- kolor, 641
  - tła węzła, 640
- kolory RGBA, 618
- kolumny, 621
- komentarze, 92, 115
  - Javadoc, 93, 115
  - jednowierszowe, 93, 97, 115
  - tradycyjne, 93, 115
- kompilacja
  - aplikacji, 79, 97, 116, 158
  - modułu, 1337, 1344
  - programu, 75
  - typu JIT, 77
  - wymazywanie, 923
- kompilator, 64, 548, 920, 974, 989
  - Simple, 994
- kompilowanie typów pakietów, 957

- komponent
  - BorderPane, 609, 869
  - Button, 581
  - GridPane, 580, 599
  - JEditorPane, 1275
  - JScrollPane, 1210
  - JTable, 1210
  - Pane, 609
  - Slider, 581
  - TextField, 581
  - TitledPane, 609, 639
  - ToolBar, 202, 869
- komponenty
  - GUI, 570
  - układu, 607, 608
  - wielokrotnego stosowania, 449
- kompozycja, 402, 418, 439, 474, 961
  - wyrażeń lambda, 819
- komunikacja oparta na strumieniach, 1274
- komunikaty, 67
- konfiguracja
  - bazy danych, 1171, 1208
  - BorderPane, 611
  - Circle, 621
  - etykiety, 153, 575
  - GridPane, 621
  - ImageView, 154, 576
  - Javy DB, 1171
  - kolumn, 621
  - komponentu układu GridPane, 621
  - kontenera układu, 151
  - kontenera układu VBox, 575
  - kontrolerek, 626
  - kontrolki Canvas, 202
  - pól tekstowych, 621
  - Rectangle, 621
  - suwaków, 621
  - tekstu przycisku, 202
  - układu BorderPane, 204
  - wierszy, 621
- konkatenacja, 287
- konkretna
  - klasa, 497
  - podklasa, 500
- konsola
  - Javy 9, 1215
  - REPL, 1216
- konstruktor, 134, 139, 158, 632
  - bezargumentowy, 416, 439, 525, 1317
  - CommissionEmployee, 455
  - DatagramPacket, 1317
  - DeckOfCards, 344
  - delegujący, 413
  - domyślny, 142, 160, 403, 413, 416, 439
  - dwuparametrowy, 144
  - enum, 421
  - klasy nadrzędnej, 464
  - podklasy, 464, 473, 478
  - prywatny, 521, 530
  - przeciążony, 414
  - String, 651
  - StringBuilder, 665
  - ResultSetTableModel, 1182
- konsument usługi, 1368
- kontener, 573
  - układu, 151, 573
  - układu VBox, 574, 575
- kontroler, 573, 582, 615
  - książki adresowej, 1200–1202
- kontrolka, 150, 570
  - Button, 597, 599, 1020
  - Canvas, 202, 204
  - ColorPicker, 869
  - ExceptionDialog, 1019
  - ImageView, 597
  - Label, 597
  - ListView, 641
  - MediaView, 1020, 1050
  - RadioButton, 608, 610
  - Slider, 597, 599
  - TableView, 634
  - TextArea, 732
  - TextField, 597, 599
  - ToggleGroup, 608
  - WebView, 644
- kontrolki
  - ControlsFX, 1020
  - modyfikowane programowo, 583
- konwencje nazewnictwa
  - modułów, 1332
  - pakietów, 957
- konwersja
  - IntStream, 829
  - jawna, 189
  - między typami, 189
  - niejawna, 190
- kopiowanie
  - list, 757
  - przeglądarki okładek, 631
- korzeń, 598
- krok rekurencyjny, 914

krzywa Kocha, 865  
 kształty, 608, 1006  
   dwuwymiarowe, 1047  
   JavaFX, 609, 640  
   trójwymiarowe, 1042, 1045, 1053  
 kwantyfikatory, 681, 688  
   leniwe, 681  
   zachłanne, 681, 688

**L**

Label, 597  
 lambda, 596, 783, 789, 833, 837  
   obliczanie silni, 855  
   obsługa zdarzeń, 831, 841  
   odgadywanie typów, 806  
   parametr typu, 791  
   przechwytyjąca, 806  
   pusta listą parametrów, 792  
   składnia, 834  
   uproszczenie listy parametrów, 792  
   uproszczenie treści, 792  
   zasięg leksykalny, 806  
 LAMP, 85  
 lewy ukośnik, 100, 116, 704  
 liczba argumentów, 158  
 liczby  
   całkowite, 102, 105  
   losowe, 294, 890  
   pierwsze, 1125  
   zmiennoprzecinkowe, 129, 142,  
     160  
     formatowanie, 239  
 licznik, 179  
   instrukcji, 989  
 LIFO, Last In, First Out, 289, 960, 974  
 Linux, 69  
 List<Employee>, 839  
 lista  
   argumentów o zmiennej długości,  
     365, 381  
   modułów JDK, 1325  
   parametrów, 132, 158, 791  
 ListView, 630, 641  
 listy, 743, 777, 944  
   jednokierunkowe, 946, 973  
   tworzenie, 815  
 literały  
   klasy, 1367  
   liczb zmiennoprzecinkowych, 160  
   tekstowe, 651  
   znakowe, 116, 650, 686

logiczna reprezentacja tablic, 329  
 logiczne operacje wyjściowe, 728  
 luźne powiązanie, 1368

**Ł**

łańcuchy wyjątków, 536  
 łączenie  
   danych, 1208  
   tekstów, 287, 661, 799, 836, 1388  
 łączność, 198  
   operatorów, 258

**M**

macOS, 70  
 mapowanie, 789  
   tekstów, 811, 838  
 mapy, 767  
 marshaling, 714  
 mashupy, 83  
 maszyna wirtualna, VM, 75  
   Javy, JVM, 75  
 mechanizm  
   dowiązania właściwości, 641  
   odśmieciania pamięci, 423  
   refleksji, 1326, 1330, 1356, 1368  
   rozszerzeń, 959  
   rozwiązywania modułów, 1362  
   skórek, 637  
   wczytywania klas, 958, 960  
 MediaPlayer, 1019, 1024  
 MediaView, 1019, 1021  
 metadane, 1176  
 metoda, 66, 96, 116, 279–316  
   absolute, 1210  
   add, 441  
   addAll, 752, 761  
   addEntryButtonPressed, 1204  
   addLast, 751  
   addPerson, 1199  
   addTableModelListener, 1178  
   allMatch, 817  
   append, 668, 687  
   anyMatch, 817  
   appendReplacement, 1380  
   appendTail, 1380  
   applyFilterButtonPressed, 1190  
   asString, 641  
   average, 801  
   awaitTermination, 1079, 1144  
   binarySearch, 752, 759, 760

- metoda
  - bind, 641
  - blockingGet, 1098, 1105, 1114
  - blockingPut, 1097, 1105, 1114
  - browseAllButtonPressed, 1204
  - call, 1147
  - cancelButtonPressed, 1127, 1130
  - capacity, 666
  - charAt, 667, 686, 687
  - charValue, 675, 688
  - clear, 763
  - clearButtonPressed, 618, 1017
  - clearRect, 878
  - closeConnection, 1287, 1293
  - collect, 799, 836
  - colorRadioButtonSelected, 617
  - compareTo, 654, 657, 686
  - compile, 684
  - concat, 686
  - connectToServer, 1292
  - containsKey, 770
  - convertToUppercaseStrings, 749
  - copy, 752, 758
  - count, 801
  - createJdbcRowSet, 1192, 1210
  - createStatement, 1209
  - cube, 1244
  - dealCard, 345
  - decreaseLevelButtonPressed, 874
  - deposit, 144
  - digit, 673, 687
  - disconnectFromDatabase, 1184
  - disjoint, 752, 761
  - displayAlert, 1190
  - displayContact, 1204
  - displayCubes, 1242, 1244
  - displayMessage, 1297, 1301
  - displayShape, 1017
  - displayState, 1099, 1106
  - draw, 486
  - drawFractal, 874, 875
  - drawingAreaMouseClicked, 1016
  - dropWhile, 1382
  - earnings, 455, 465, 472, 485, 506
  - endsWith, 657
  - ensureCapacity, 666, 687
  - equals, 654, 656, 686
  - equalsIgnoreCase, 654, 656, 686
  - execute, 1143, 1211
  - executeQuery, 1209
  - exists, 730
  - fabryczna of, 775
    - interfejsu Map, 776
    - interfejsu Set, 775
  - factorial, 853
  - fibonacci, 858
  - fill, 752, 757
  - fillArc, 377, 1052
  - fillOval, 1052
  - fillRect, 1052
  - fillRoundRect, 1039, 1052
  - filter, 835
  - finalize, 423
  - find, 684, 689
  - findAny, 817
  - findButtonPressed, 1204
  - findFirst, 817
  - fireTableStructureChanged, 1210
  - first, 766
  - flatMap, 825
  - forDigit, 673
  - forEach, 797, 821, 829, 835, 840
  - format, 707, 1148
  - frequency, 752, 761
  - generate, 830
  - get, 730, 745, 770, 1149
  - getAddress, 1317
  - getAllEntries, 1204
  - getAllPeople, 1199
  - getAsDouble, 801
  - getAverage, 364
  - getBalance, 144, 146
  - getCause, 558
  - getChars, 652, 667
  - getClass, 1006
  - getColumnClass, 1178, 1183, 1209
  - getColumnClassName, 1210
  - getColumnCount, 1183, 1209, 1210
  - getColumnName, 1210
  - getColumnType, 1177
  - getConnection, 1209
  - getData, 1317
  - getFileName, 703, 730
  - getInputStream, 1279, 1281
  - getLastModifiedTime, 703, 731
  - getMessage, 556
  - getMinimum, 363
  - getName, 132
  - getOutputStream, 1279, 1281
  - getPeopleByLastName, 1199
  - getPrimesButtonPressed, 1127, 1130
  - getResource, 1006

- getRowCount, 1184, 1210
- getSelectedToggle, 640
- getSelectionModel, 642
- getStreams, 1286
- getUserData, 641
- getValue, 878
- getValueAt, 1184, 1210
- getWidth, 209
- goButtonPressed, 1121, 1123
- handle, 1034, 1036, 1051
- headSet, 766, 778
- indexOf, 686
- initialize, 594, 617, 624, 641, 874, 1015, 1189
- inorderTraversal, 971
- insert, 687, 970
- insertAtBack, 952
- insertAtFront, 952
- insertionSort, 902
- insertNode, 970
- ints, 797, 828
- isAbsolute, 703, 731
- isCancelled, 1148
- isDefined, 673, 687
- isDigit, 687
- isDirectory, 703, 731
- isEmpty, 771, 961
- isJavaIdentifierPart, 673
- isJavaIdentifierStart, 673, 687
- isLetter, 673
- isLetterOrDigit, 673
- isLowerCase, 673, 687
- isNaN, 539
- isUpperCase, 673
- iterate, 830, 1383
- iterator, 746
- keySet, 771, 778
- lastIndexOf, 660
- length, 652, 666, 686, 687
- linearSearch, 889
- lines, 825
- lineSeparator, 757
- lookingAt, 684, 689
- main, 116, 137, 181, 302, 599, 889, 965
- map, 811, 834
- mapToDouble, 822
- mapToObj, 798, 836
- marshal, 732
- matcher, 684, 689
- matches, 677, 688
- Math.max, 287
- max, 752, 758, 801
- maximum, 286, 924, 925
- merge, 908
- mergeSort, 907
- min, 752, 758, 801
- multiply, 441
- negate, 839
- newBufferedReader, 719
- newBufferedWriter, 732
- newCachedThreadPool, 1072, 1143
- newCondition, 1110
- newDirectoryStream, 731
- next, 158, 746, 1177, 1209
- nextDouble, 160
- nextInt, 105
- nextLine, 158
- nextNumberButtonPressed, 1121, 1124
- notify, 1094, 1145
- notifyAll, 1094, 1145
- now, 1148
- observableArrayList, 642
- of, 779, 836
- offer, 763, 778
- ofNullable, 1383
- orElse, 801
- outputBarChart, 363
- outputGrades, 363
- parallelPrefix, 1133, 1148
- parallelSetAll, 1132, 1148
- parallelSort, 371, 1132
- peek, 763
- playPauseButtonPressed, 1025
- poll, 763
- pop, 961
- postorderHelper, 972
- postorderTraversal, 971
- pow, 441
- preorderHelper, 971
- preorderTraversal, 971
- print, 956, 961
- printArray, 922
- printCards, 757
- printf, 101, 116
- printPass, 900, 903
- printReversedList, 749
- printSet, 766
- printStackTrace, 556
- processConnection, 1287, 1293
- push, 928, 961
- put, 770, 778

- metoda
  - range, 423, 833
  - rangeClosed, 833
  - readRecords, 713
  - reduce, 802, 803
  - regionMatches, 654, 657
  - remove, 746
  - removeColors, 746
  - removeFromBack, 954
  - removeFromFront, 954
  - removeItems, 749
  - removeTableModelListener, 1178
  - replace, 662, 686
  - replaceAll, 684, 688, 1380
  - replaceFirst, 684, 689, 1380
  - results, 1380
  - reverse, 667, 752, 757
  - reverseOrder, 753
  - rollback, 1206
  - rollDice, 301
  - runClient, 1292
  - runLater, 1119, 1147
  - runServer, 1285
  - sceneProperty, 1050
  - selectDirectoryButtonPressed, 726
  - selectDouble, 1025, 1050
  - selectedItemProperty, 630, 642
  - selectFileButtonPressed, 726
  - selectFirstEntry, 1204
  - selectionSort, 899
  - send, 1317
  - sendPacketToClient, 1297
  - setCellFactory, 642
  - setCharAt, 667
  - setCommand, 1211
  - setContent, 1210
  - setDiffuseColor, 1053
  - setDiffuseMap, 1053
  - setErr, 699
  - setFill, 1039, 1052
  - setGlobalAlpha, 1039, 1052
  - setIn, 699
  - setInitialDirectory, 732
  - setItems, 642
  - setLength, 666
  - setLineCap, 1039, 1052
  - setName, 131, 132
  - setOnError, 1050
  - setOnReady, 1050
  - setOnRunning, 1147
  - setOnSucceeded, 1147
  - setOut, 699
  - setPage, 1277
  - setPassword, 1211
  - setQuery, 1184
  - setRowFilter, 1210
  - setSpecularColor, 1053
  - setString, 1211
  - setStroke, 878, 1039, 1052
  - setTime, 403, 406, 414
  - setUrl, 1210
  - setUserData, 610, 640
  - setUsername, 1210
  - shapeRadioButtonSelected, 1017
  - showDialog, 732
  - showOpenDialog, 732
  - shuffle, 344, 752, 755
  - shutdown, 1073, 1143
  - signal, 1110
  - size, 703, 731, 745
  - Size, 771
  - sizeRadioButtonSelected, 618
  - sleep, 1086
  - sort, 752, 971, 1132
  - sortArray, 907
  - split, 676, 689
  - splitAsStream, 825
  - startFibonacci, 1139
  - startsWith, 657
  - stream, 808, 838
  - stroke, 1052
  - strokeArc, 377, 1052
  - strokeLine, 209, 211, 1039, 1052
  - strokeRect, 1052
  - strokeRoundRect, 1040, 1052
  - submit, 1148
  - submitQueryButtonPressed, 1189
  - substring, 686, 1241
  - sum, 801, 833, 935
  - supplyAsync, 1149
  - swap, 899
  - System.out.print, 116
  - System.out.println, 96, 116
  - tailSet, 766
  - takeWhile, 1382
  - testPopDouble, 931
  - testPopInteger, 932
  - testPushDouble, 931
  - testPushInteger, 932
  - toAbsolutePath, 703, 731
  - toArray, 750
  - toCharArray, 687
  - toExternalForm, 1006
  - toLowerCase, 663

- toMillis, 1148
- toPath, 726, 732
- toString, 341, 404, 455, 465, 472, 703, 731
- toUniversalString, 404
- toUpperCase, 662, 673, 687, 1241
- trim, 663, 687
- undoButtonPressed, 618
- unlock, 1109
- updateItem, 633, 642
- updateMessage, 1120, 1147
- updateProgress, 1125
- updateValue, 1125
- valueOf, 441, 663, 687
- valueProperty, 641
- wait, 1094, 1145
- waitForConnection, 1286
- waitForPackets, 1297, 1301
- metody
  - abstrakcyjne, 490, 491, 527
  - charAt, 652
  - domyślne interfejsu, 519
  - dostępowe, 417, 439
  - fabryczne, 521, 773, 779
  - finalne, 508, 528
  - instancyjne, 675, 798, 811
  - interfejsu Stream, 1381
  - klasy
    - Character, 675
    - Collections, 743, 751, 759, 762, 777
    - CompletableFuture, 1391
    - Matcher, 685, 1378
    - Math, 282
    - Object, 473
    - Stack<E>, 961
    - String, 652, 661
    - SynchronizedBuffer, 1097
    - Tree, 971
  - obiektu dayName, 1240
  - obsługi zdarzeń, 1121, 1127
  - otoczek synchronizacji, 771
  - otoczkowe, 743
  - pobierające, 129, 138, 157, 1209
  - powrót z metody, 288
  - predykatowe, 248, 417, 950
  - prywatne, 530
  - przeciążone, 280, 308, 927, 939, 1127, 1240
  - publiczne, 158
  - rekurencyjne, 850
  - rozdzielanie metod przeciążonych, 307
  - styczne, 282, 313, 425, 600
  - typy zwracane, 308
  - uogólnione, 918, 920, 938
    - implementacja, 920
    - przekształcenia, 920
  - ustawiające, 129, 138, 157
  - warunki końcowe, 560
  - warunki wstępne, 560
  - wyszukujące, 658, 659
  - wywoływanie, 288
    - z wieloma parametrami, 284, 313
  - miary bajtów, 63
  - Microsoft SQL Server, 1176
  - migracja, 1327
    - kodu do Javy 9, 1350
  - model, 582
    - przerywania kodu, 543
  - modularyzacja platformy, 1324
  - moduł, 1323
    - com.deitel.videoplayer, 1356
    - java.base, 1333, 1359
    - java.se, 1347, 1348
    - java.sql, 1346
  - modułowa wersja aplikacji, 1331
  - moduły, 1321
    - aplikacji MathTutor, 1363
    - artefakty, 1337
    - automatyczne, 1351, 1354, 1355
    - deklaracja, 1328
    - jawnie nazwane, 1345
    - JDK, 1325
    - JRE, 1358
    - kompilacja, 1344
    - mechanizm rozwiązywania, 1362
    - nienazwane, 1351
    - pliki zasobów, 1357
    - standardowe, 1325
    - tworzenie, 1340
    - uruchomienie aplikacji, 1345
    - w JShell, 1384
    - zasoby, 1354
  - modyfikacja
    - hierarchii, 523
    - klasy Employee, 515
  - modyfikator
    - final, 421, 508
    - open, 1329
    - private, 430
    - protected, 430
    - public, 430
    - static, 421

modyfikatory dostępu, 130  
   prywatne, 131  
   publiczne, 131  
 modyfikowanie obiektów, 130, 1233  
 monitory, 1075  
 Mozilla Foundation, 69  
 MVC, Model-View-Controller, 582, 599  
 MySQL, 1176  
 mysza  
   obsługa zdarzeń, 610

## N

nagłówek metody, 131  
 narzędzie  
   jdeps, 1352, 1353  
   jlink, 1325, 1358  
   Scene Builder, 150, 571, 597, 642  
   Scenic View, 636, 643  
 nasłuchiwanie zmian  
   właściwości, 620, 625, 641  
   zaznaczenia, 630  
 nawias  
   klamrowy, 158  
   lewy, 95  
   prawy, 95  
   kwadratowy, 329, 331  
   ostry, 1004  
   redundantny, 106  
 nazwa  
   klasy, 94, 115, 157  
   klasy kontrolera, 588, 626  
   metody, 588, 613  
   modułu 1332  
   pakietu, 957  
   pliku, 94  
   zmiennej, 1233  
   kontrolera, 613  
   kwalifikowana, 1167  
   zmiennej, 117  
 NetBeans, 51  
 niefatalny błąd logiczny, 174  
 niejawne  
   czytanie, 1328  
   deklarowanie zmiennych, 1224, 1225  
 nieprzewidywalność  
   harmonogramowania wątków, 1080  
 niesynchronizowany bufor, 1090  
 nietypowa inicjalizacja, 140  
 NIO, 697, 700, 730

notacja  
   dużego O, 885, 910  
   rombowa, 374, 746  
 numery wersji JDK, 52

## O

obiekt, 157, 401  
   ActionEvent, 592  
   Arc, 1009  
   ArrayList, 327, 740  
   AutoCloseable, 562  
   BigDecimal, 432  
   Box, 1042  
   BufferedReader, 719  
   CachedRowSet, 1191, 1210  
   CallableStatement, 1211  
   canRead, 1112  
   Canvas, 1006  
   canWrite, 1112  
   CharSequence, 683  
   Circle, 1008  
   Condition, 1111  
   Connection, 1209  
   Cylinder, 1042  
   DatagramPacket, 1294, 1317  
   DatagramSocket, 1294  
   Ellipse, 1009  
   EventHandler, 1034  
   Executor, 1070  
   ExecutorService, 1072  
   GraphicsContext, 1006, 1038, 1041  
   HyperlinkListener, 1278  
   InputStream, 729, 1279, 1281  
   Interpolator.EASE\_IN, 1051  
   Interpolator.EASE\_OUT, 1051  
   IntStream, 800  
   Iterator, 743  
   JdbcRowSet, 1210  
   JScrollPane, 1189  
   JTable, 1179, 1180, 1181  
   KeyFrame, 1051  
   Line, 1008  
   LinearGradient, 1052  
   Lock, 1111  
   Map, 770  
   Map.Entry, 779  
   MathContext, 441  
   Media, 1022, 1024, 1050  
   MediaPlayer, 1022, 1024, 1050  
   MediaView, 1022, 1050



- MyLine, 436
- Node, 1045
- OutputStream, 1279, 1281
- Path, 700, 730, 1013
- PhongMaterial
  - dla Box, 1045
  - dla Cylinder, 1045
  - dla Sphere, 1045
- Polyline, 1013
- PreparedStatement, 1194, 1198, 1211
- RadialGradient, 1052
- Rectangle, 1008
- ResultSetMetaData, 1209
- RowSet
  - połączony, 1190
  - rozłączony, 1190
- Runnable, 1070
- Scanner, 103, 117, 133, 1230
- SecureRandom, 294
- ServerSocket, 1278
- ServiceLoader, 1367
- Socket, 1280
- Sphere, 1042
- Statement, 1209
- Stop, 1052
- String, 686
- SynchronizedBuffer, 1098
- System.err, 552
- System.in, 103
- System.out, 96, 116, 542, 552
- Throwable, 565
- Timeline, 1034
- objektowa analiza, 68
- obiekty, 65
  - klasy nadrzędnej, 449, 487
  - niepolimorficznie, 503
  - obsługi błędów, 542
  - otoczkowe, 759
  - po serializacji, 714
  - podklasy, 449
  - programowe, 56, 510
  - strumieni, 1279
  - tekstowa reprezentacja, 663
  - warunków, 1110
  - wejścia, 103
  - wyjścia, 116
  - zapobieganie tworzeniu, 521
  - zwykle, 715
- obliczenie
  - iloczynu wartości, 803
  - procentu składanego, 433
  - silni, 855
  - sumy kwadratów, 803
  - wartości ciągu Fibonacciego, 1123
- obrazy, 1053
- obsługa
  - CLDR, 1389
  - dla zdarzeń MediaPlayer, 1025
  - ekranów, 637
  - GTK+ 3, 637
  - JShell w IDE, 1256
  - obrazów TIFF, 1386
  - paczek zasobów, 1389
  - protokołu DTLS, 1387
  - Stream<Employee>, 812
  - Stream<Integer>, 807, 838
  - Stream<String>, 810
  - usług, 1388
  - walidacji, 1388
  - współbieżności, 1390
  - wyjątków, 340, 380, 535, 541, 545, 552, 563
  - zdarzeń, 200, 206, 581, 599, 831, 841, 869
  - zdarzeń myszy, 593, 610, 640
- odbijanie koła od krawędzi, 1032
- odczyt
  - obiektów serializowanych, 720
  - z pliku, 1275–1277, 1316
  - sekwencyjnego, 719, 732
  - tekstowego, 707
- odgadywanie typów, 746, 806
- odkrywanie
  - metody substring, 1241
  - składowych obiektu, 1240
- odpytywanie bazy danych, 1178, 1209
- odśmiecanie pamięci, 423, 438, 440, 946
- odtworzenie wideo, 1022, 1024, 1049, 1050
- odwzorowanie, 767, 778, 789, 834, 840
  - liczb, 836
  - obiektów Employee, 820
- offset, 652
- ograniczenie
  - wieloznaczności, 937
  - liczby elementów strumienia, 830
- okno
  - DirectoryChooser, 720, 732
  - FileChooser, 720, 732
  - JavaFX, 572, 598
  - JShell Edit Pad, 1227

- określanie
    - typu obiektu, 503
    - źródła danych, 788
  - określnik formatu %f, 160
  - okręg z gwiazdek, 1018
  - opcje JShell, 1254
  - operacja
    - average, 793
    - count, 793
    - distinct, 793
    - filter, 793
    - forEach, 793
    - limit, 793
    - logiczna AND, 819
    - logiczna OR, 819
    - map, 793
    - max, 793
    - min, 793
    - reduce, 793, 802, 837
    - sorted, 793
  - operacje
    - arytmetyczne, 107, 118
    - gorliwe, 792
    - IntStream, 799, 836
    - kończące, 792, 835, 836
    - leniwe, 792
    - na stosie, 960
    - niedestrukcyjne, 107
    - niepodzielne, 1094
    - pośrednie, 792, 835
    - redukcji, 793
    - równoległe, 1133
    - rzutowania, 926
    - szeregowe, 1133
    - zbiorcze, 743
  - operandy, 105
  - operator, 1395, 1396
    - =, 656
    - AND, 255, 270
    - dekrementacji, 196, 216
    - inkrementacji, 196, 216
    - INNER JOIN, 1166, 1208
    - instanceof, 503, 506
    - negacji logicznej, 256, 270
    - OR, 253, 255, 270
    - przedrostkowy
      - dekrementacji, 196
      - inkrementacji, 196
    - reszty, 118
    - rzutowania, 189, 291
    - trójargumentowy, 174
    - warunku, 174, 215
    - XOR, 255, 270
  - operatory
    - binarne, 105, 107
    - logiczne, 252, 255, 270
    - przypisania, 106, 114, 216
    - przypisania z dodawaniem, 195
    - równości i relacji, 111, 118
  - oprogramowanie, 58
    - jako usługa, SaaS, 86
  - ORACLE, 1176
  - organizacja komputera, 59
  - otoczki synchronizacji, 771
- ## P
- PaaS, Platform as a Service, 86
  - pakiet
    - com.deitel.mathtutor, 1364
    - com.deitel.mathtutor.spi, 1364
    - java.io, 700, 726, 733
    - java.lang, 104, 683
    - java.math, 431
    - java.net, 1274
    - java.nio, 700
    - java.sql, 1209
    - java.text, 599
    - java.util, 103, 809
    - java.util.concurrent, 1143, 1149
    - java.util.concurrent.locks, 1109
    - java.util.function, 805
    - java.util.stream, 808
    - javafx.application, 599
    - javafx.beans.property, 620
    - javafx.beans.value, 599
    - javafx.collections, 642
    - javafx.concurrent, 1147
    - javafx.event, 592
    - javafx.scene, 640
    - javafx.scene.canvas, 1052
    - javafx.scene.control, 592, 599, 601
    - javafx.scene.control.skin, 644
    - javafx.scene.layout, 574
    - javafx.scene.media, 1019
    - javafx.scene.shape, 609, 639, 1006
    - javafx.swing.table, 1178
    - javafx.xml.bind, 716
  - pakiety API Javy, 292, 293
  - pakowanie, 742
    - automatyczne, 777
  - pamięciowe strumienie znakowe, 729

- pamięć, 106
  - dynamiczna alokacja, 946
  - Simpletronu, 987
- Pane, 607, 609, 612
- para klucz–wartość, 779
- parametry, 132, 137, 157
  - typu, 791, 921
  - metod, 1238
- pewność konfiguracji, 1364
- pętla for, 231, 234
  - sumowanie liczb, 237
  - wyrażenia arytmetyczne, 234
  - wyrażenia opcjonalne, 234
- pętla nieskończona, 189
- phishing, 737
- pierwszy program, 92
- pióro Lo, 866, 872
- platforma jako usługa, PaaS, 86
- plik
  - clients.xml, 718
  - module-info.java, 1335
  - Painter.fxml, 611, 640
  - PolyShapes.css, 1013
  - TipCalculator.fxml, 583
  - TransitionAnimations.fxml, 1026
  - Welcome.fxml, 151
- pliki, 63, 697, 730
  - .class, 1352
  - .css, 999
  - .xml, 732
  - binarne, 698
  - DOT, 1353
  - JAR, 1338, 1345
    - wielowydaniowe, 1390
  - sekwencyjne
    - aktualizacja, 713
    - deserializacja danych, 719, 732
    - odczyt, 719, 732
  - tekstowe, 698
    - sekwencyjne, 704, 731
    - z fragmentami kodu, 1234
- pobieranie
  - danych, 183
  - danych z obiektu wyjątku, 556
  - informacji o plikach, 701
  - obiektów Path, 701
  - obiektu GraphicsContext, 1038
  - wartości z klawiatury, 104
- podgląd
  - GUI, 586, 587
  - interfejsu aplikacji, 155, 578
- podklasy, 449, 452, 477, 478, 487, 509
- podkreślenie, 95
- podpowiedzi, 1389
- podprotokół derby, 1175
- pola, 62
  - edytowanie, 587
  - uaktywnienie, 587
  - klasy SynchronizedBuffer, 1097
  - statyczne, 282, 313, 1188
  - tekstowe, 600, 641
    - konfiguracja, 621
- pole
  - tipTextField, 587
  - totalTextField, 587
- polecenia JShell, 1250
- polecenie, 97 *Patrz także* instrukcja
  - /drop, 1252
  - /edit, 1251
  - /methods, 1243
  - /open, 1234
  - /reload, 1251
  - /set editor, 1247
  - clear, 1414
  - cont, 1404
  - jdeps, 1352
  - jlink, 1358
  - next, 1410
  - print, 1404, 1408
  - run, 1404
  - set, 1408
  - step, 1410
  - step up, 1410
  - stop, 1404
  - System.out.println, 96
  - watch, 1412
- polimorficzne przetwarzanie klas, 505, 517
- polimorfizm, 247, 484–486, 492, 527
- połączenia strumieniowe, 1316
- połączenie z bazą danych, 1173, 1175, 1209
- pomiar czasu działania metod, 1130, 1148
- ponowne zgłaszanie wyjątków, 553
- porównanie
  - leksykograficzne, 656
  - tekstów, 653, 656
- postdekrementacja, 196
- postinkrementacja, 196
- potoki, 727
  - strumienia, 788, 795, 808, 833

- powiązanie
  - luźne, 1327
  - ściśle, 1327
- powielenie kodu, 147, 462
- pozycjonowanie względne, 607
- prawy ukośnik, 704
- precyzja liczb zmiennoprzecinkowych, 190
- predefiniowane klasy znaków, 677
- predekrementacja, 196
- predykat, 793
- preinkrementacja, 196
- priorytet
  - operatorów, 114
  - wątków, 1068
- procedura, 166
  - nasłuchiwania właściwości, 620, 641
  - obsługi zdarzenia, 206, 593, 599, 640
  - składowana, 1204, 1211
- proces
  - destrukcyjny, 106
  - obiektywnej analizy i projektowania, 68
- programowanie
  - deklaratywne, 572, 789
  - do interfejsu, 522, 530
  - funkcyjne, 789
  - generyczne, 784
  - imperatywne, 789
  - obiektywne, 56, 447, 483, 784
  - proceduralne, 784
  - skonkretyzowane, 485
  - strukturalne, 168
  - współbieżne, 1065
- programowe
  - tworzenie komponentów, 630
  - wczytywanie CSS, 1006, 1047
- programy strukturalne, 259
- projektowanie, 68
  - bazujące na dziedziczeniu, 475
  - kompozycji, 475
- promocja argumentu, 291, 314
- protokół
  - DTLS, 1387
  - IP, 83
  - TCP, 82
  - UDP, 1275
- prototypowanie, 1218
- prywatne
  - konstruktorzy, 530
  - metody, 530
  - metody interfejsów, 521, 530
- przeciążanie
  - konstruktorów, 410, 439
  - metod, 280, 306, 316
  - metod uogólnionych, 927, 939
  - niezamierzone, 456
- przeciążone konstruktory, 410, 414, 439
- przeglądanie
  - dokumentacji metody, 1238
  - dokumentacji pól publicznych, 1238
  - parametrów metod, 1238
  - przeciążeń metod, 1240
  - składowych klas, 1236
  - tablic, 750
- przekazywanie
  - przez referencję, 350, 381
  - przez wartość, 350, 381
- przekształcenie, 1017, 1049
  - Rotate, 1049
  - Scale, 1049
  - Translate, 1049
- przepelnienie
  - arytmetyczne, 183
  - stosu, 290
- przepływ sterowania programem, 558
- przesłanie
  - metod, 451, 495, 590, 1036
  - zmiennej, 304
- przestrzeń nazw XML, 1005
- przetwarzanie
  - interakcji użytkownika, 1287, 1293
  - polimorficzne, 503, 505, 517
  - transakcyjne, 1205, 1211
  - wyników pośrednich, 1125, 1148
  - wyników zapytania, 1176
- przezroczystość, 618
- przycisk, 613
  - zmiana rozmiaru, 586
- przyciski opcji, 608, 639
- przysłonięcie zmiennej, 132
- pseudokod, 167, 213
- publiczne
  - klasy abstrakcyjne, 529
  - metody domyślne, 519
- pula wątków, 1070
- punkty wstrzymania, 1404

puste wiersze, 115  
 pusty obiekt, 669

## R

RadioButton, 608, 612, 639, 640  
 ramki stosu, 290, 859

Rectangle, 621

redukcja, 786, 833

refaktoryzacja, 85

referencja, 148

do konstruktora, 826

do metody, 792, 796, 835

do metody instancyjnej, 798, 811

do obiektu, 160, 451

do pliku CSS, 1003, 1005, 1047

do samego siebie, 408

statyczna do metody, 798

this, 439

w przód, 1243

refleksja, 1326, 1330, 1356, 1368

reguła, 999

CSS #label1, 1001

CSS .vbox, 1000

uczciwości, 1109, 1110

zagnieżdżania, 259

rejestracja procedury obsługi zdarzenia,  
 581, 593

rekordy, 62

aktywacyjne, 289

rekurencja, 849, 859, 876

kontra iteracja, 877

pośrednia, 852

z nawrotami, 875

rekurencyjna

metoda sortArray, 907

struktura folderów, 852

rekurencyjne

nawracanie, 878

wyszukiwanie

binarne, 913

liniowe, 913

relacyjne

bazy danych, 1157, 1206

systemy bazodanowe, 1156

REPL, Read Evaluate Print Loop, 1216

reprezentacja

drzewa binarnego, 966

listy jednokierunkowej, 947

operacji

insertAtBack, 953

insertAtFront, 953

removeFromBack, 955

removeFromFront, 954

reszta z dzielenia, 107

RGB, 308

RGBA, 618, 641

rodzaje

fragmentów, 1217

polimorfizmu, 485

sterowania, 261

rozbity folder aplikacji, 1338

rozgałęzienie, 992

bezw warunkowe, 993

rozgłaszanie, 1275

rozmiar

MediaViewer, 1025

węzłów, 606

rozpakowywanie, 742, 777

rozszerzanie klasy abstrakcyjnej, 1036

rozszerzenia interfejsów, 519, 529

rozszerzona instrukcja for, 380

rozwiązania chmurowe, 86

rozwiązywanie modułów, 1325

rozwijanie stosu, 555

rysowanie

deszczu, 79, 82

koła, 1040

koncentrycznych okręgów, 266

kształtów, 1006

linii, 200, 209, 1039

losowych linii, 436

łodygi, 79

łuku, 375, 1041

na obiekcie Canvas, 1037, 1038,  
 1052

obiektów

Path, 1015

Polygon, 1014

Polyline, 1013

owalu, 1040

pióra Lo, 872

płatków kwiatka, 79

trawy, 81

z użyciem polimorfizmu, 524

zaokrąglonego prostokąta, 1039

rzecz, 84

rzeczywisty argument typu, 921

rzut kostką sześcienną, 295

rzutowanie typu, 189, 291, 314

jawne, 507

w dół, 488, 503, 507

## S

- SaaS, Software as a Service, 86
- SAM, Single Abstract Method, 520, 804, 837
- scena, 572
- Scene Builder, 571, 597
  - dostosowywanie GUI, 581
  - generowanie klasy kontrolera, 588
  - uruchomienie narzędzia, 574
- Scenic View, 636, 643
- scenka, 573, 591
- SDK, Software Development Kit, 86
- sekwencja ucieczki, 100, 677
- sekwencyjne
  - pliki tekstowe, 704, 731
  - struktury danych, 966
- selektor
  - klasy, 1000
  - typu, 1009
- serializacja, 719
  - obiektów, 511, 1387
  - obiektu List, 731
  - XML, 697, 713, 731
- serializowalny obiekt CachedRowSet, 1191
- serwer, 1316
  - wielowątkowy, 1301
  - WWW, 1275
- siatka, 580
- sieć, 1273
  - WWW, 83
- silna enkapsulacja, 1345
- silnia, 852
- silnik WebKit, 638
- Simple, 985
- Simpletron, 987
- sito Eratostenesa, 1125, 1126, 1148
- skalowalność, 1324
- skalowanie
  - prostokątu, 1028
  - wartości BigDecimal, 434
- składanie strumieni wejściowych, 728
- składnia lambda, 791, 834
- składowe
  - chronione, 451, 477
  - klasy, 1236
  - statyczne, 289, 424, 440
- skórka, 637, 644, 1385
- Slider, 581, 597, 599
- słowa
  - kluczowe ograniczone, 1331
  - zarezerwowane, 1399
- słowo kluczowe, 115, 1399
  - class, 94
  - final, 283, 1075
  - IDENTITY, 1159
  - implements, 529
  - new, 103, 134, 158, 332
  - private, 131
  - public, 130, 286
  - static, 286
  - synchronized, 1094, 1111, 1145
  - this, 408, 439
  - throw, 553
  - void, 96
- słowo kluczowe SQL
  - DELETE, 1162
  - FROM, 1162
  - GROUP BY, 1162
  - INNER JOIN, 1162
  - INSERT, 1162
  - ORDER BY, 1162
  - SELECT, 1162
  - UPDATE, 1162
  - VALUES, 1168
  - WHERE, 1162
- SMS, Short Message Service, 696
- sortowanie, 885
  - bąbelkowe, 912
  - kubelkowe, 913
  - po wielu kolumnach, 1166
  - przez scalanie, 896, 903, 907, 911
  - przez wstawianie, 896, 900, 911
  - przez wybieranie, 896, 897, 899, 911
  - strumienia, 808, 838
  - tekstów, 812
  - w kolejności malejącej, 753, 1165
  - w kolejności rosnącej, 752
  - wartości IntStream, 804, 837
  - za pomocą własnego komparatora, 754
- sprawdzenie konfiguracji, 78
- sprzęt, 58
- SQL, 1156, 1162, 1206
  - łączenie klauzul, 1166
- StackPane, 607
- stałe
  - ResultSet, 1182, 1183
  - statyczne, 283

- standardowy
  - obiekt wyjścia, 96
  - strumień błędów, 552
  - strumień wyjścia, 552
- standaryzacja interakcji, 510
- stany wątku, 1066
- statyczna referencja do metody, 798
- statyczne
  - metody, 425
  - metody interfejsu, 520
- sterowanie
  - czcionkami, 999, 1046
  - programem, 167
  - wykonywaniem programu, 1410
- sterownik
  - JDBC, 1206
  - urządzenia, 492
- stos, 314, 928, 944, 960, 973
  - operacje, 960
  - przepelnienie, 290
  - wywołań metod, 289, 859
  - zastosowania, 960
- stosowanie
  - interfejsów, 509, 529
  - klas abstrakcyjnych, 490
- Stream<Integer>, 838
- Stream<String>, 840
- String, 651, 657, 686
- StringBuilder, 664, 668, 687
- struktury
  - danych, 943
    - dynamiczne, 944, 973
    - liniowe, 966
    - typu FIFO, 963
    - typu LIFO, 289, 960
    - własne, 943
  - folderów, 1355
  - katalogów, 1337
  - okna aplikacji JavaFX, 598
  - sekwencyjne, 168, 170, 213
  - sterujące, 167, 213
    - zagnieżdżone, 191
  - wyboru, 170
- strumienie, 730, 783, 786, 833
  - bajtowe, 698, 699
  - błędów, 699
  - buforowane, 727
  - danych, 727
  - filtrowanie, 809, 838
  - filtrów, 727
  - jako tablice elementów, 728
  - nieskończone, 829, 841
  - ograniczenie liczby elementów, 830
  - potoków, 727
  - przetwarzanie potoku, 788
  - równoległe, 1135
  - skracanie przetwarzania potoku, 816
  - sortowanie, 808, 838
  - sumowanie, 787
  - szeregowane, 1133, 1135, 1148
  - wartości losowych, 841
  - upraszczanie zadań, 799
  - wartości losowych, 827
  - wejścia, 697, 699
  - wyjścia, 697, 699
  - znakowe, 699
    - buforowane, 729
    - pamięciowe, 729
    - plików, 729
    - potoków, 729
    - tekstów, 729
  - zrównoleglone, 1133, 1148
- strumień IntStream, 797, 835
- styl wielbłądzi, 95
- stylowanie
  - dwuwymiarowych kształtów, 1048
  - obiektu
    - Arc, 1011
    - Circle, 1011
    - Ellipse, 1011
    - Line, 1010
    - Rectangle, 1011
- suwak, 582
  - konfiguracja, 621
  - ustawienie właściwości, 587
- Sybase, 1176
- symbol
  - @, 1232
  - złączenia, 178
- synchroniczne obliczanie elementów
  - ciągu, 1139
- synchronizacja, 1084
  - dostępu, 1080, 1096, 1105
  - kolekcji, 779
  - wątków, 1074, 1143
- system
  - bazodanowy, 1176
  - modułów, 1321
  - operacyjny, 68
    - Android, 70
    - iOS, 70
    - Linux, 69
    - macOS, 70
    - Windows, 69

system  
 warstwowy, 492  
 wykonawczy, 1358  
 zarządzania bazą danych, 1206  
 szkielet kontrolera, 206  
 szybkie prototypowanie, 1218

**Ś**

ścieżka CLASSPATH, 1246  
 ścieżki  
 pełne, 701, 730  
 względne, 701, 730  
 ścisłe powiązanie, 1327  
 średnik, 97  
 środek  
 elementu Ellipse, 1048  
 łuku, 1048  
 środowisko programistyczne, 74, 75

**T**

tabela  
 AuthorISBN, 1159  
 Authors, 1158  
 Titles, 1158  
 tablica  
 frequency, 340  
 liczb losowych, 890  
 typu Employee, 505  
 tablice, 327, 329, 379  
 analiza wyników, 338  
 deklaracja, 330  
 dwuwymiarowe, 358  
 elementy jako liczniki, 337  
 graficzne przedstawienie danych,  
 336  
 inicjalizacja, 332  
 jako obiekty, 751  
 logiczna reprezentacja, 329  
 mieszające, 764  
 operacje, 359  
 przeglądanie, 750  
 przekazywanie do metod, 348  
 sumowanie elementów, 335  
 tablic jednowymiarowych, 357  
 tworzenie, 330, 332, 357, 379,  
 1132  
 wielowymiarowe, 356, 381

TCP, Transmission Control Protocol,  
 82  
 technika dziel i rządź, 280  
 technologie, 85  
 tekst, 600, 649  
 filtrowanie, 812  
 łączenie, 661, 799, 836, 1388  
 mapowanie, 811, 838  
 porównywanie, 653  
 sortowanie, 812  
 tokenizacja, 675  
 wydobywanie fragmentu, 660  
 znajdowanie położenia znaków, 658  
 tekstowa reprezentacja obiektu, 663  
 test  
 aplikacji Client, 1292  
 aplikacji Server, 1285  
 testowanie, 78  
 drzewa binarnego, 969  
 kalkulatora napiwków, 579  
 klasy, 460, 467  
 CircularBuffer, 1106  
 Queue<E>, 965  
 Stack<E>, 962  
 SynchronizedBuffer, 1099  
 metody cube, 1244  
 uogólnionej klasy, 929  
 TextField, 581, 597, 599, 641  
 TilePane, 608  
 TitledPane, 609, 612, 639  
 dostosowanie elementów, 612  
 ToggleGroup, 608  
 tokenizacja tekstów, 675, 688  
 tokeny, 987  
 ToolBar, 202  
 transmisja  
 bezpołączeniowa, 1317  
 połączeniowa, 1293, 1317  
 transparentność referencyjna, 804  
 treść  
 deklaracji, 115  
 klasy, 95  
 metody, 96, 158  
 tryb  
 informacji zwrotnej  
 concise, 1253  
 silent, 1254  
 verbose, 1253  
 wysokiego kontrastu, 634



- tworzenie
    - animacji, 1034
    - aplikacji współbieżnych, 1066
    - baz danych
      - w systemie Linux, 1173
      - w systemie macOS, 1172
      - w systemie Windows, 1171
    - egzemplarzy, 66
    - gniazd
      - datagramowych, 1316
      - strumieniowych, 1316
    - GUI, 611, 620, 640, 641
    - GUI aplikacji, 582, 600
    - hierarchii Payable, 512
    - interfejsu, 509, 529
      - graficznego, 720
      - graficznego aplikacji, 434, 626
    - klasy w JShell, 1231
    - kolekcji, 808
    - kontenera układu VBox, 151, 574
    - List<Employee>, 814
    - nazwy zmiennej, 1233
    - niezmiennej listy, 815
    - obiektu, 134, 139, 1232
      - BigDecimal, 432
      - IntStream, 800, 836
      - Media, 1024
      - MediaPlayer, 1024
      - Path, 700
      - PreparedStatement, 1198
      - Scanner, 103, 1230
      - SecureRandom, 294
      - ServerSocket, 1278
      - Socket, 1280
      - Timeline, 1034
    - okręgu, 1018
    - programowe komponentów, 630
    - programów strukturalnych, 259
    - programu, 74
    - prostego
      - klienta, 1280, 1316
      - serwera, 1278, 1316
    - scenki, 591
    - sekwencyjnego pliku, 714, 731
    - skórek, 637
    - Stream<Integer>, 808, 838
    - Stream<String>, 823, 840
    - strumienia, 835
    - strumienia IntStream, 797
    - systemów wykonawczych, 1358
    - tablic, 330, 357, 1132
    - tablicy typu Employee, 505
    - uogólnionych metod, 932
    - wątków, 1143
    - własnego
      - formatu komórek, 630, 642
      - kompilatora, 944
      - modułu, 1340
    - własnej kontrolki, 643
      - JavaFX, 635
    - zadania, 1120
  - typ danych, 101
    - char, 104, 117
    - Comparable<T>, 925
    - double, 104, 143
    - enum, 299, 302, 315, 440, 616, 709
    - float, 104, 160
    - int, 104, 117
    - Object, 741
  - typy
    - generyczne, 920
    - podstawowe, 104, 117, 148, 160, 199, 216, 1401
    - referencyjne, 135, 148, 160, 329, 921, 925, 927
    - uogólnione, 918
    - zagnieżdżone, 640
    - zwracane, 131, 133, 158
- ## U
- uaktywnienie pól, 587
  - UDP, User Datagram Protocol, 1275
  - układ
    - AnchorPane, 607
    - BorderPane, 204, 607, 609
    - FlowPane, 607
    - GridPane, 607, 621
    - HBox, 608
    - Pane, 607, 609, 612
    - StackPane, 607
    - TilePane, 608, 639
    - VBox, 574, 575, 608, 612
  - układ współrzędnych kanwy, 208
  - układanie węzłów, 639
  - ukośnik, 1004
  - ukrywanie informacji, 67, 139, 159
  - UML, Unified Modeling Language, 68, 136, 159
  - Unicode 8, 1390
  - uogólnione klasy i metody, 917, 925
  - uporządkowana lista znaków, 654
  - uproszczenie listy parametrów, 792

URI, Uniform Resource Identifier, 701  
URL, Uniform Resource Locator, 701  
uruchomienie aplikacji, 79  
    bez dostawców usług, 1369  
    po modularyzacji, 1357  
    z pliku, 1339  
urządzenia, 57  
usługi, 1363  
    sieciowe, 83  
usunięte  
    funkcjonalności platformy, 1391  
    metody, 1392  
użycie, *Patrz także* stosowanie  
    bloku finally, 551  
    dostawców usług, 1366  
    importu statycznego, 428  
    inicjalizatora tablicy, 333  
    klasy BigDecimal, 431  
    obsługi wyjątków, 545, 564  
    rekurencji, 877  
    RowSet, 1191  
    tablic mieszających, 767  
    własnych modułów, 1340

## V

VBox, 151, 574, 608, 612  
    dodanie  
        elementów ImageView, 576  
        elementów RadioButton, 612  
        etykiety, 575  
        TitledPane, 612  
    konfiguracja, 575  
    etykiety, 153  
        ImageView, 154  
    kontenera układu, 151  
    określenie klasy dla stylów, 1006,  
        1047  
    preferowany rozmiar, 575  
    tworzenie, 574  
    wyrównywanie, 575  
VideoPlayer, 1020  
VM, Virtual Machine, 75

## W

walidacja, 1388  
    danych, 728  
    danych wejściowych, 676  
    informacji, 678

wartości  
    BigDecimal  
        skalowanie, 434  
        zaokrąglanie, 433  
    domyślne, 142  
    losowe, 827, 841  
wartość  
    alfa, 618  
    argumentu, 135  
    NaN, 539  
    null, 134  
    położenia, 329  
    tożsamościowa, 802  
    wyrażenia, 105, 106  
    zmiennej, 117  
warunki  
    końcowe, 560, 566  
    wstępne, 560, 566  
wątek  
    blokady, 1075  
    blokady wzajemne, 1069  
    Consumer, 1098  
    cykl życia, 1066, 1142  
    framework Executor, 1070  
    główny, 1073  
    harmonogramowanie, 1068  
    odkładanie w nieskończoność, 1110  
    odsuwanie wykonania, 1069  
    priorytety, 1068  
    Producer, 1097  
    roboczy, 1125, 1147  
    stan, 1066, 1142  
        działający, 1066  
        działający z punktu widzenia  
            systemu operacyjnego, 1068  
    nowy, 1066  
    oczekiwania, 1097  
    oczekiwania czasowego, 1071,  
        1067, 1086  
    oczekujący, 1067  
    zablokowany, 1068  
    zakończony, 1068  
    synchronizacja, 1074, 1143  
    tworzenie, 1143  
    wyjątek InterruptedException,  
        1072  
    wykonywanie, 1143  
    zarządzanie, 1072  
wczytywanie  
    dostawców usług, 1366  
    programu do pamięci, 76

- WebKit, 638
- WebView, 644
- wersja
  - alfa, 87
  - beta, 87
  - ciągła beta, 87
  - kandydat do wydania, 87
  - wydanie finalne, 87
- weryfikacja kodu bajtowego, 76
- węzeł
  - grafu, 598
  - kolejki, 975
  - korzenia, 966
  - listy, 974
- węzły, 573, 606, 1045
  - położenie, 607
  - rozmiar, 606
  - układanie, 639
- wideo, 997, 1019
- widget, 570, 597
- widok, 582
- wielokrotne wykorzystanie kodu, 66, 281, 498, 501
- wielomian stopnia drugiego, 110
- wielowątkowość, 1117, 1146
- wielowydaniowe pliki JAR, 1390
- wieloznaczność, 934, 940
- wieloznaczny argument typu, 935
- wiersz poleceń, 96, 367, 621
- wieże Hanoi, 862
- Windows, 69
- własna
  - fabryka komórek, 642
  - nazwa zmiennej, 1233
- własne
  - kontrolki, 643
  - moduły, 1340
  - struktury danych, 943
  - systemy wykonawcze, 1358
- własny
  - format komórek, 642
  - kompilator, 944
  - system wykonawczy, 1359
- właściwości
  - elementu Box, 1043
  - elementu Cylinder, 1043
  - klasy, 619
  - Sphere, 1044
  - suwaka, 587
- właściwość, 641
  - accessibleHelpProperty, 643
  - accessibleRoleDescriptionProperty, 643
  - accessibleRoleProperty, 643
  - accessibleTextProperty, 643
  - CSS -fx-fill, 1048
  - CSS -fx-stroke, 1010
  - CSS -fx-stroke-line-cap, 1048
  - fx:id, 583, 611
  - labelFor, 643
  - Max Width, 640
  - maxHeight, 607
  - On Mouse Dragged, 640
  - Pref Height, 1008
  - Pref Width, 613
  - prefWidth, 607
  - samopodobieństwa, 865
  - Selected, 640
  - Spacing, 640
  - Text, 640
  - Toggle Group, 612, 640
  - width, 607
- wprowadzanie danych, 104
- współbieżność, 1063, 1142, 1390
- współczynnik wypełnienia tablicy mieszającej, 768
- współdzielenie modyfikowalnych danych, 1076
- wstrzykiwanie zależności, 1330
- wybór
  - folderu, 725
  - kolekcji, 742
  - podzbiorów danych, 1158
- wychwytywanie wyjątków, 542
- wydajność, 664, 947, 1324
  - binarnego drzewa wyszukiwania, 972
  - sortowania przez scalanie, 908
  - sortowania przez wstawianie, 903
  - wyszukiwania binarnego, 896
- wydobywanie fragmentu tekstu, 660
- wyjątek, 535, 563, 1244
  - ArithmeticException, 539, 540, 547, 564
  - ArrayIndexOutOfBoundsException, 547
  - AssertionError, 561
  - FileNotFoundException, 706
  - FormatterClosedException, 706
  - IllegalArgumentException, 404
  - IllegalMonitorStateException, 1095
  - IllegalStateException, 709
  - InputMismatchException, 539, 540, 564

- wyjatek
    - NoSuchElementException, 707, 962
    - NullPointerException, 346, 763
    - RuntimeException, 547
    - SecurityException, 706
    - SQLException, 1175
    - StringIndexOutOfBoundsException, 652, 661
  - wyjatkı
    - deklarowanie nowych rodzajów, 559
    - deklarowanie nowych rodzajów, 566
    - hierarchia, 546
    - kontrolowane, 547
    - łańcuchowe, 557, 566
    - niekontrolowane, 547, 548
    - niewychwycone, 543
    - uzyskiwanie informacji, 554, 565
    - wychwytywanie, 549
  - wykonywanie
    - aplikacji, 98, 116, 158
    - aplikacji z wieloma klasami, 136
    - fragmentów, 1223
    - kodu bajtowego, 77
    - wątków, 1143
  - wymazywanie, 926
    - na etapie kompilacji, 923
  - wypełnienie
    - liniowe, 1041
    - radialne, 1040
    - w postaci obrazu, 1040
  - wyrażenia, 117
    - algebraiczne, 105, 109
    - arytmetyczne, 108, 118
    - lambda, 790, 834
    - regularne, 649, 676, 688, 689, 1378
      - kwantyfikatory, 681
      - predefiniowane klasy znaków, 677
      - sprawdzanie danych, 679
      - walidacja informacji, 678
    - w języku Java, 109
  - wyrażenie
    - dostępu do tablicy, 330
    - warunkowe, 174
  - wyszukiwanie, 885
    - binarne, 892, 910
    - liniowe, 887, 910
  - wyświetlanie
    - grafu zależności modułów, 1347
    - kształtów dwuwymiarowych, 1006, 1047
  - liczb
    - pierwszych, 1129
    - zmiennoprzecinkowych, 145
  - List<Employee>, 814, 839
  - listy modułów JRE, 1358
  - pojedynczego wiersza, 99
  - różnic procentowych, 1132
  - tekstu, 101, 116
  - tekstu i obrazka, 150, 573
  - wyniku obliczeń, 105, 117
  - wywołania
    - blokujące, 1139
    - konstruktora, 413
    - metod, 67, 134, 158, 288
    - metod z konstruktorów, 509, 529
    - metody maximum, 925
    - niejawne, 503
    - polimorficzne, 506
    - rekurencyjne, 851
  - wzorce projektowe, 85
  - wzorzec wyszukiwania, 676
- X**
- XML, eXtensible Markup Language, 597, 713
- Z, Ż**
- zachowanie polimorficzne, 487, 527
  - zadania
    - asynchroniczne, 1136
    - działające równolegle, 1141
    - działające współbieżnie, 1141
  - zadanie, 1120, 1147
    - PrintTask, 1072
  - zagnieżdżone
    - instrukcje, 172, 175
    - struktury sterujące, 191
  - zakres
    - półotwarty, 788
    - zamknięty, 788
  - zależność od stanu, 1084
  - zamykanie kodu, 542
  - zanikanie prostokątu, 1027
  - zaokrąglenie wartości BigDecimal, 433
  - zapis
    - do pliku, 706
    - układu graficznego, 205
    - wielbłądzi, 131
  - zapobieganie tworzeniu obiektów, 521

- zapytanie SELECT, 1162, 1207
- zarządzanie
  - bazą danych, 1206
  - wątkami, 1072
- zasada
  - jak najmniejszego dostępu, 508
  - najmniejszych przywilejów, 306
- zasięg
  - deklaracji, 303, 315
  - leksykalny, 806
  - na poziomie klasy, 424
- zasoby w modułach, 1354
- zastosowania
  - kolejek, 963
  - stosu, 960
- zbiory, 764, 778
  - posortowane, 765
- zbiór znaków ASCII, 1397
- zdarzenia, 150, 581, 599, 641
  - interfejsu graficznego, 1301
  - MediaPlayer, 1025
- zdarzenie
  - HyperlinkEvent, 1275, 1277
  - myszy, 608, 610
    - onMouseClicked, 593, 610
    - onMouseDragEntered, 610
    - onMouseDragExited, 610
    - onMouseDragged, 610
    - onMouseDragOver, 610
    - onMouseDragReleased, 610
    - onMouseEntered, 610
    - onMouseExited, 610
    - onMouseMoved, 610
    - onMousePressed, 610
    - onMouseReleased, 610
- zgłaszanie wyjątków, 403, 552
  - ponowne, 553
- zintegrowane środowisko
  - programistyczne, IDE, 75
- zliczanie, 822, 840
  - wystąpień słów, 824
- złączenie, 1166
- zmiana
  - rozmiaru elementu ImageView, 577
  - rozmiaru przycisku, 586
- zmienna, 102
  - instancji books, 629
  - instancji name, 131
  - środowiskowa
    - CLASSPATH, 51, 959, 1175, 1246
    - JAVA\_HOME, 51, 1360
    - PATH, 50, 76
- typu, 921
- zmienne
  - instancji, 67, 129, 131, 157, 160, 617, 624, 1024
  - instancji typu final, 429, 440
  - lokalne, 132, 158, 181, 290, 304, 545
    - efektywnie finalne, 562
  - mutowalne, 827
  - statyczne, 283, 424, 592
  - typu podstawowego, 160
  - typu referencyjnego, 160
- znacznik
  - końca pliku, 244, 698, 708, 713
  - otwierający, 1004
  - strzałki, 791
  - zamykający, 1004
- znajdowanie
  - liczb pierwszych, 1125
  - położenia znaków, 658
- znak, 61, 649
  - @, 1232
  - białej spacji, 133, 688
  - gwiazdki, 107, 136, 958
  - kropki, 312
  - numeryczny, 688
  - przecinka, 239
  - separatora, 704, 731
  - spacji, 99
  - średnika, 97, 563
  - ucieczki, 100, 688
  - wieloznacznym, 937
  - wyrazu, 688
- znaki ASCII, 1397
- znakowe dane wejściowe, 730
- zrzut stosu, 538, 556
- zwalnianie zasobów, 550, 562
  - automatyczne, 566
- związek typu
  - jeden do wielu, 1161
  - producent – konsument, 1101, 1109, 1083, 1091, 1144
  - wiele do wielu, 1162
- związki między tabelami, 1161
- zwinne wytwarzanie oprogramowania, 85
- zwolnienie blokady, 1094, 1105
- zwracany typ, 137
- źródło danych, 788



# PROGRAM PARTNERSKI

— GRUPY HELION —

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

Zmień swoją stronę WWW w działający bankomat!

**Dowiedz się więcej i dołącz już dzisiaj!**

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

GRUPA  
**Helion**

## Programuj profesjonalnie — ucz się od najlepszych!

Tworzenie oprogramowania to wspaniała umiejętność. Zdolny programista może w zasadzie pracować w dowolnej dziedzinie. Co więcej, szalony rozwój technologii informatycznych właściwie codziennie otwiera nowe rynki i nowe nisze. Niemal wszędzie potrzebne są procesory i oprogramowanie. I spora część tego cyfrowego tortu przypada programistom Javy. Najlepsze książki jednak dostają się tym najlepszym, najzdolniejszym i najbardziej profesjonalnym.

Jeśli chcesz być właśnie takim programistą, wzięteś do ręki właściwą książkę. Jest to klasyczny podręcznik, dzięki któremu wiele osób osiągnęło mistrzostwo w programowaniu w Javie. Zawarto tu wyjątkowo rzetelne, praktyczne i aktualne wprowadzenie do języka. W książce znajdziesz sporo informacji o nowej wersji języka — Javie 9 — oraz o świetnym narzędziu JShell, które ułatwia szybkie poznanie języka. Szczegółowo opisano JavaFX — najnowszy GUI i równocześnie zestaw narzędzi do nowych projektów. W przystępny i zrozumiały sposób przedstawiono dość trudne pojęcia, takie jak współbieżność, dzięki czemu bez problemu wykorzystasz moc systemów wielordzeniowych.

### W tej książce między innymi:

- Solidne wprowadzenie do Javy — klasy, obiekty, metody
- Podstawy programowania, w tym programowania obiektowego
- Struktury danych, kolekcje, lambdy i strumienie
- Rozwiązania bazodanowe
- System modułów platformy Java 9

**Paul Deitel** jest czempionem Javy. Ma ponad 35 lat doświadczenia informatycznego. Ukończył MIT, otrzymał tytuły Java Certified Programmer i Java Certified Developer. Prowadził setki kursów programowania dla pracowników takich jednostek jak Cisco, IBM, Siemens, Sun Microsystems (obecnie Oracle), Dell, NASA, Boeing, Nortel Networks, Puma, iRobot.

**Harvey Deitel** ma ponad 55 lat doświadczenia informatycznego. Również ukończył MIT, a na Uniwersytecie Bostońskim zdobył stopień doktora matematyki. Uczył programowania na uczelniach, w urzędach, firmach i wojsku.

Publikacje Deitelów są rozpoznawane na całym świecie i były tłumaczone na dziesiątki języków.

	<i>Sprawdź nasze szkolenia!</i>	<b>KOD KORZYŚCI</b> Sięgnij po więcej! ▶	
 <a href="http://helion.pl">helion.pl</a>		ISBN 978-83-283-3973-6	
 <b>HELION SA</b> ul. Kościuszki 1c 44-100 Gliwice tel.: 32 230 98 63 helion@helion.pl	<b>AKADEMIA IT &amp; BUSINESS</b> <a href="http://WWW.SZKOLENIA.HELION.PL">WWW.SZKOLENIA.HELION.PL</a>	 9 788328 339736	
<b>INFORMATYKA W NAJLEPSZYM WYDANIU</b>			Cena: 179,00 zł