

ntainsValue() double canWrite() @FunctionalInterface offer() Va
lterReader add() GregorianCalendar first() private BufferedInput
tAbsoluteFile() i-- InterruptedException getParent() TimeZone c
calTime abstract BigDecimal() Calendar next() byte Channel le
ansient @SuppressWarnings close() ClassCastException higher() c
tPath() int toSecondOfDay() short start() try SQLException ge
id withdrawal() Locale isHidden() switch ArrayIndexOutOfBounds
adSet() PushbackInputStream @Override canExecute() boolean peek
strictfp Thread if last() NumberFormatException setReadOnly() n
iverManager Vector() ZonedDateTime isNaN() for nanoTime()
assNotFoundException available() @Deprecated Object assert test
pedWriter System.out.println() PrintWriter this toNanoOfDay()
arArrayReader descendingSet() **Michał Suwała** boolean
Infinite flowError contains
ringWrit Warnings import perform() Collections conca
tNano() ts() Volatile lastModified() ArithmeticExcep
LowerCas sleep() @Inherited trim() IntWrapper isDir
Exceptio pported
ntimeExc Excepti
terface apacity
rtualMac inst Ex
lete() @Targ
UpperCas insert
iter li ic Fil
pedReade ppressw
bset() Path mark() catch @Native OutOfMemoryError canRead() t
ecute() AssertionError mkdir() Runnable while Vector getAbsolute
rowable tExceptio
tParentF poll() L
File() nted get
cept() ode() i+
fferedReader notifyAll() float payment() default lower() Reter
ite() ValueHolder getMinute() NaN tailSet() break print()
legalArgumentException isAbsolute() else sort() CharArrayWriter
ror clear() Reader public LocalDate createNewFile() class pri
etention notify() double canWrite() @FunctionalInterface offer()
it() FilterReader add() GregorianCalendar first() private
fferedReader getAbsoluteFile() i-- InterruptedException get
meZone containsKey() LocalTime abstract BigDecimal() Calendar
annel length() transient @SuppressWarnings close() ClassCastExc
gher() continue getPath() int toSecondOfDay() short start() t
LException getSecond() void withdrawal() Locale isHidden() s
rayIndexOutOfBoundsException headSet() PushbackInputStream @Over
nExecute() boolean peek() Buffer strictfp Thread if last()
mberFormatException setReadOnly() new addAll() DriverManager Ve
nedDateTime isNaN() for nanoTime() ClassNotFoundException availa

Java

Teoria w praktyce

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 wydawca 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 wydawca nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Redaktor prowadzący: Małgorzata Kulik

Projekt okładki: Studio Gravite/Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki

Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: helion@helion.pl
WWW: <https://helion.pl> (księgarnia internetowa, katalog książek)

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

ISBN: 978-83-289-0022-6

Copyright © Michał Suwała 2024

Printed in Poland.

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

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

Spis treści

	Wstęp	11
ROZDZIAŁ 1.	Wprowadzenie do Javy	13
	Historia języka	13
	Główne założenia języka	14
	Prostota i obiektywność	14
	Przenośność, bezpieczeństwo, niezawodność i wydajność	15
	Wielowątkowość i rozproszenie	15
	Interpretowalność i niezależność od architektury	15
	Maszyna wirtualna Javy	15
	Model pamięci	16
	Pakiet instalacyjny	17
ROZDZIAŁ 2.	Pierwszy program	18
	Witaj, świecie!	18
	Zasady składni kodu źródłowego	19
	Identyfikatory	20
	Literały	20
	Słowa kluczowe	20
	Potencjalne problemy	21
	Korzystanie z IDE	21
ROZDZIAŁ 3.	Zmienne	22
	Tworzenie zmiennych	23
	Prymitywne typy danych	25
	byte	26
	short	27
	int	28
	long	28
	float	29
	double	29
	Rzutowanie liczb zmiennoprzecinkowych na typy całkowite	30
	Utrata precyzji w obliczeniach zmiennoprzecinkowych	31
	char	32
	boolean	33

ROZDZIAŁ 4.	Operatory	34
	Operatory arytmetyczne	34
	Operatory przypisania	36
	Operatory logiczne	37
	Operatory bitowe	38
	Kolejność wykonywania operatorów	40
ROZDZIAŁ 5.	Instrukcje sterujące	42
	Instrukcja if-else	42
	Instrukcja switch	44
	Pętle	46
	Pętla while	46
	Pętla do-while	47
	Pętla for	49
	Pętla for-each	51
	Przerwania pętli	52
	Instrukcja continue	52
	Instrukcja break	53
	Etykiety pętli	54
ROZDZIAŁ 6.	Tablice	55
	Deklaracja tablicy	56
	Operowanie na tablicy	57
	Konstrukcja algorytmów opartych na tablicach	58
	Obliczanie liczby wystąpień liczb dodatnich i ujemnych w tablicy	59
	Obliczanie średniej z liczb w tablicy	59
	Obliczanie liczby wielkich i małych liter alfabetu w tablicy	60
	Sprawdzanie, czy znaki zawarte w tablicy tworzą palindrom	60
	Tablice wielowymiarowe	61
	Tablica dwuwymiarowa	61
	Tablica trójwymiarowa	65
ROZDZIAŁ 7.	Paradygmat programowania zorientowanego obiektowo	68
	Klasy i obiekty	69
	Pakiety i importy	70
	Modyfikatory widoczności	71
	Atrybuty i metody	71
	Tworzenie obiektów	72
	Enkapsulacja	73
	Konstruktory	75
	Typ wyliczeniowy enum	76
	Dziedziczenie	79
	Klasy abstrakcyjne	82
	Interfejsy	86
	Polimorfizm	87

ROZDZIAŁ 8.	Klasy — zaawansowane możliwości	90
	Przeładowanie metod	90
	Zmienna liczba i kolejność argumentów	92
	Varargs	93
	Dopasowanie wersji metody do przekazanego argumentu	94
	Przekazywanie argumentów przez wartość i referencję	96
	Przekazywanie argumentów przez wartość	96
	Przekazywanie argumentów przez referencję	97
	Statyczne składowe klas	98
	Zastosowanie	101
	Statyczne stałe	102
	Bloki kodu	103
	Instancyjny blok inicjalizacyjny	104
	Statyczny blok inicjalizacyjny	105
	Porównanie bloków	105
	Klasy zagnieżdżone	106
	Klasa wewnętrzna	106
	Klasa lokalna	110
	Klasa anonimowa	112
	Klasa zagnieżdżona statyczna	114
	Rekordy	116
	Refleksja	117
	Klasa opisująca klasę — java.lang.Class	118
	Reflection API	119
	Zabawki w służbie refleksji	120
	Podsumowanie	125
	Adnotacje	125
	Wbudowane adnotacje	126
	Tworzenie adnotacji	126
	Parametry adnotacji	128
	Użycie adnotacji	129
	Podsumowanie	131
	Rzutowanie zmiennych obiektowych	132
	Rzutowanie w dół	133
	Rzutowanie w górę	135
	Operator instanceof	135
	Rzutowanie — błędy kompilacji	137
ROZDZIAŁ 9.	Wbudowane typy obiektowe	138
	Typy opakowaniowe	138
	Tworzenie zmiennych typu opakowaniowego	140
	Operacje na dużych liczbach	143
	BigInteger	143
	BigDecimal	146

Klasa String	149
Pula stringów — String Literal Pool	150
Niemodyfikowalność napisów	151
API klasy String	151
StringBuilder i StringBuffer	160
API klas StringBuilder	160
StringBuilder append()	161
StringBuilder insert(int offset, ...)	161
StringBuilder delete(int startIndex, int endIndex)	162
StringBuilder reverse()	162
int capacity()	162
ROZDZIAŁ 10. Wyjątki	163
Jak programować bez użycia wyjątków?	163
Pierwsze zetknięcie z wyjątkiem	166
Co to jest wyjątek?	166
Definiowanie klas wyjątków	167
Wbudowane klasy wyjątków	168
Wyjątki sprawdzalne i niesprawdzalne na etapie kompilacji	170
Wyjątki sprawdzalne	171
Wyjątki niesprawdzalne	171
Rzucanie wyjątków	172
Instrukcja throws	172
Instrukcja throw	173
Łapanie wyjątków	175
Klauzula try-catch	175
Łapanie wielu typów wyjątków	178
Łapanie wyjątków powiązanych dziedziczeniem	181
Deklaracja w throws najogólniejszego typu wyjątku	182
Blok finally	183
Wyjątki a dziedziczenie metod	186
Zawężanie listy wyjątków w podklasie	187
Rozszerzane listy wyjątków w podklasie	188
Klauzula try-with-resources	189
Przykład z użyciem wyjątków	189
ROZDZIAŁ 11. Klasa Object	193
Object clone()	194
Płytką kopia	196
Głęboka kopia	197
boolean equals(Object obj)	198
Implementacja standardowa metody equals()	199
Implementacja equals() z wykorzystaniem klasy Objects	200
Przykład wywołania equals()	200
Zasady działania equals()	201

int hashCode()	201
Implementacja standardowa metody hashCode()	202
Implementacja hashCode() z wykorzystaniem klasy Objects	202
Przykład wywołania hashCode()	202
Konsekwencje nieodpowiedniego nadpisania metody hashCode()	203
Kontrakt pomiędzy equals() a hashCode()	204
String toString()	205
Implementacja domyślna toString()	205
Nadpisanie toString()	206
void notify(), notifyAll(), wait(...)	206
void finalize()	207
Wady metody finalize()	207
ROZDZIAŁ 12. Typy generyczne	208
O co chodzi z błędami rzutowania?	209
Parametry generyczne	211
Wiele parametrów generycznych	214
Rozszerzanie typów generycznych	217
Zawężanie typu parametrów	218
Błędy kompilacji	220
Metody generyczne	220
Wildcards — generyczne argumenty w metodach	223
Ograniczenie typu parametru „z góry”	224
Przykład z kolekcją typu Car	226
Przykład z kolekcją typu Book	226
Ograniczenie typów parametrów „z dołu”	227
ROZDZIAŁ 13. Który obiekt jest większy?	229
Interfejs Comparable	230
Porównywanie rosnąco	231
Porównywanie malejąco	232
Porównywanie na podstawie więcej niż jednej zmiennej	233
Zasady porównywania obiektów	234
Interfejs Comparator	235
Różnice pomiędzy Comparable a Comparator	237
Kiedy użyć Comparable?	237
Kiedy użyć Comparator?	237
ROZDZIAŁ 14. Kolekcje	238
Interfejs Collection	239
Listy	240
ArrayList	241
LinkedList	253
Vector	255
Stack	257
ArrayDeque	259

Kolejki	265
PriorityQueue	265
Zbiory	272
HashSet	272
LinkedHashSet	282
TreeSet	283
Iteratory	298
Mapy	300
HashMap	301
LinkedHashMap	307
TreeMap	309
Operacje narzędziowe	312
Kolekcje niemodyfikowalne	312
Algorytmy	315
ROZDZIAŁ 15. Paradygmat programowania funkcyjnego	318
Interfejs funkcyjny	318
Tworzenie interfejsu funkcyjnego	319
Wyrażenia lambda	321
Wbudowane interfejsy funkcyjne	322
Function	322
BiFunction	324
UnaryOperator	325
BinaryOperator	326
Predicate	326
Consumer	327
Supplier	329
Comparator	330
Referencje do metod	330
Referencja do metody statycznej	331
Referencja do metody instancyjnej obiektu	332
Referencja do metody instancyjnej typu	333
Referencja do konstruktora	333
ROZDZIAŁ 16. Przetwarzanie strumieniowe	336
Interfejs Stream<T>	337
Tworzenie strumienia	338
Operacje pośrednie i kończące	339
Stream API	340
Operacje pośrednie	345
Operacje kończące	352
Strumień numeryczny	362
Współbieżne przetwarzanie strumieni	363
Przykład strumienia współbieżnego	364
Współbieżny dostęp do zasobu	365
Klasa Optional<T>	366

ROZDZIAŁ 17. I/O – wejście-wyjście programu	370
Reprezentacja pliku — klasa File	370
Ścieżka względna i bezwzględna	372
Określanie ścieżki w zależności od systemu operacyjnego	372
Operowanie na pliku	373
Metadane pliku	376
Strumienie wejścia-wyjścia	377
Strumienie bajtów	378
Strumienie znaków	384
RandomAccessFile	392
Serializacja	394
Wyłączenie z serializacji — słowo kluczowe transient	397
Serializacja obiektu zagnieżdżonego w obiekcie	398
Pakiet NIO	399
Bufory	399
Kanały	400
Selektory	401
Pakiet NIO 2	402
Path	402
Paths	403
Files	403
ROZDZIAŁ 18. Kalendarz, data i czas	407
Jak w Javie reprezentowany jest czas	407
Kalendarz	408
Klasa Calendar	408
Klasa TimeZone	409
Klasa Locale	409
Klasa GregorianCalendar	410
Data i czas	412
Reprezentowanie daty do wersji 1.7 Javy	412
Reprezentowanie daty od wersji 1.8 Javy	417
ROZDZIAŁ 19. Podstawy programowania współbieżnego	426
Proces a wątek	426
Wątek główny	427
Cykl życia wątku	427
NEW	427
RUNNABLE	428
BLOCKED, WAITING, TIMED_WAITING	428
TERMINATED	428
Tworzenie wątku	428
Klasa Thread	428
Interfejs Runnable	431

Synchronizacja wątków	433
Wyścig wątków	433
Atomowość operacji	435
Słowo kluczowe synchronized	435
Monitor	436
Słowo kluczowe volatile	437
Komunikacja między wątkami	437
Zakleszczenie wątków	440
Rozwiązanie problemu zakleszczenia wątków	442
Kolekcje bezpieczne w środowisku wielowątkowym	442
ROZDZIAŁ 20. Podstawy JDBC	445
Komponenty	445
Nawiązywanie połączenia z bazą danych	447
Connection	447
DriverManager	448
DataSource	449
Zapytania	449
Statement	450
PreparedStatement	455
CallableStatement	459
Mapowanie ResultSet na kolekcję	460
Skorowidz	462

ROZDZIAŁ 6.

Tablice

Poznałeś do tej pory zmienne, które pozwalają na przechowywanie wartości luźno rozmieszczonych w pamięci RAM. Dzięki nim jesteśmy w stanie zapamiętywać częściowe wyniki obliczeń naszego programu bądź reprezentować dane na potrzeby przetwarzania w algorytmach. Czasem jednak zdarza się, że zmienne tego samego typu przydałoby się pogrupować w jeden jednolicie zarządzany zbiór, aby przeprowadzać te same operacje na wszystkich wartościach naraz — bez konieczności odwoływania się do konkretnych zmiennych. Rozwiązaniem tego problemu w Javie jest tablica.

Tablica jest obiektem agregującym wartości tego samego typu. Można ją sobie wyobrazić jako sekwencję umieszczonych obok siebie komórek, z czego każda składa się z takiej liczby bitów, jaka wynika z danego typu danych. Tablica ma skończoną długość, a jej maksymalny rozmiar zależy od dostępnego miejsca w pamięci RAM. Może przechowywać elementy wyłącznie tego samego typu, którymi mogą być zarówno prymitywy, jak i referencje. Komórki tablicy numerowane są od zera; każda z nich ma przyporządkowany indeks, gdzie ostatni numer to $n - 1$, przy założeniu, że n to rozmiar tablicy.

Na rysunku 6.1 jest pokazana tablica liczb całkowitych, uporządkowanych w kolejnych komórkach o indeksach od 0 do 10. Zawiera zatem 11 elementów.

12	54	5	1	-9	34	21	99	102	-1	543
0	1	2	3	4	5	6	7	8	9	10

RYSUNEK 6.1. Tablica jednowymiarowa typu całkowitoliczbowego

Tablica ma zmienną o nazwie `length`, która zwraca liczbę całkowitą typu `int`, reprezentującą jej zadeklarowaną długość. Do poszczególnych komórek tablicy możemy uzyskać dostęp przez użycie operatora tablicowego `[]`, przekazując w nawiasie kwadratowym numer indeksu żądanego elementu. Jeśli przekazany indeks będzie mniejszy od 0 lub większy od n czy mu równy, w trakcie działania programu zostanie wyrzucony wyjątek typu `IndexOutOfBoundsException`, sygnalizujący nieuprawniony dostęp do części pamięci niebędącej tablicą.



`IndexOutOfBoundsException` jest wyjątkiem niesprawdzalnym, więc kompilator nie zgłosi błędu, jeśli w kodzie źródłowym nastąpi próba odczytu komórki o błędnym indeksie. Obowiązek operowania na tablicy w odpowiednim zakresie spoczywa na programiście.

Deklaracja tablicy

Tablicę w Javie deklarujemy tak, jakbyśmy tworzyli zwykłą zmienną określonego typu, ale dodatkowo należy posłużyć się operatorem tablicowym reprezentowanym przez nawiasy kwadratowe, żeby poinformować kompilator o jej specjalnej strukturze.

```
int [] array;  
array = new int[5];
```

Powyższy listing przedstawia deklarację i inicjalizację tablicy typu `int`. W drugiej linii deklarujemy, że tablica będzie zawierać pięć komórek gotowych do przechowywania liczb całkowitych. Można również zastosować zapis jednolinijkowy, czyli definicję tablicy.

```
int [] array = new int[5];
```

Niezależnie od tego, którym sposobem zostanie utworzona tablica, jej zawartość nie jest pusta. Zawiera bowiem w każdej komórce zero, zgodnie z zasadami opisanymi w tabeli 6.1.

TABELA 6.1. Zasady inicjalizacji pustych tablic

TYP TABLICY	INICJALNA ZAWARTOŚĆ KOMÓREK
byte, short, int	0
long	0L
float	0.0F
double	0.0D
boolean	false
char	'\u0000'
<i>referencja</i>	null

Deklarując tablicę, mamy możliwość użycia operatora `[]` przed jej nazwą lub po. Kompilator dopuszcza oba sposoby. Jeśli spróbujemy użyć zmiennej `length`, odwołując się do nazwy tablicy za pomocą operatora kropki, otrzymamy liczbę 5 — długość tablicy.

```
System.out.println(array.length);
```

Java daje nam możliwość zdefiniowania tablicy z jednoczesnym wstawieniem do niej elementów. Wiąże się to z użyciem nawiasów klamrowych, długość tablicy jest zaś automatycznie wyliczana w trakcie działania programu na podstawie liczby wymienionych inicjalnie elementów. Poniżej znajduje się taki rodzaj definicji tablicy:

```
int [] array = {10, 20, 30, 40, 50};
```

Operowanie na tablicy

Oczywiście w trakcie działania programu istnieje możliwość, a nawet konieczność, wstawienia danych innych niż inicjalne — przecież po to używamy tablic, by zagregować dane w obrębie pewnego kontekstu. Aby móc podmienić wartość znajdującą się obecnie w danej komórce, należy wskazać jej numer indeksu.

```
array[0] = 10;  
array[1] = 20;
```

Zgodnie z powyższym listingiem, komórki o indeksach 0 i 1 będą zawierać liczby, odpowiednio, 10 oraz 20. Pozostałe nadal pozostają wypełnione zerami. Łatwo można się o tym przekonać, iterując po tablicy pętlą `for` i wypisując jej zawartość na konsolę (listing 6.1).

LISTING 6.1. Program iterujący po tablicy liczb całkowitych

```
public class ArraysExample {  
    public static void main(String[] args) {  
        int [] array = new int [5];  
        array[0] = 10;  
        array[1] = 20;  
        for (int i = 0; i<array.length; i++) {  
            System.out.print(array[i] + " ");  
        }  
    }  
}
```

W wyniku wykonania powyższego listingu dostaniemy na konsoli wylistowaną zawartość tablicy:

```
10 20 0 0 0
```

Widzimy zastosowanie w praktyce pętli `for`, gdzie zmienna `i` zadeklarowana przed rozpoczęciem iteracji świetnie nadaje się do określania indeksu komórek tablicy w poszczególnych obrotach.

Jeszcze prościej wygląda użycie pętli `for-each` do przeiterowania po elementach tablicy:

```
for (int element : array) {  
    System.out.print(element + " ");  
}
```

Pamiętajmy jednak, że użycie pętli `for-each` jest możliwe tylko w kontekście odczytu zawartości komórek, gdyż zmienna `element` stanowi kopię zawartości tablicy w kolejnych obrotach. Jeśli więc chcesz zmienić stan elementów tablicy, użyj zwykłej pętli `for`:

```
for (int i = 0; i < array.length; i++) {  
    array[i] = array[i] * 1000;  
}
```

Za każdym razem, gdy pracujemy z tablicami, musimy rozważać przypadki brzegowe warunków pętli, tak aby nie wyjść poza zakres tablicy. Indeks przekazywany

do operatora tablicowego nie może być ujemny ani równy długości tablicy lub od niej większy. W każdym przypadku naruszenia tej zasady zostanie to w trakcie działania programu zakomunikowane wyrzuceniem wyjątku i przerwaniem wykonywania kodu, jeśli sytuacja wyjątkowa nie zostanie odpowiednio obsłużona. Spójrzmy na poniższy przykład:

```
int [] array = {10, 20, 30, 40, 50};
array[-1] = 100;
```

Kompilator nie zgłosi błędu, ale po uruchomieniu program zgodnie z oczekiwaniem zakończy swoje działanie, wypisując na konsoli komunikat informujący o wyjściu poza zakres tablicy i odwołując się do indeksu o wartości -1.

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: -1
```

Taki sam rezultat otrzymamy, gdy przykładowo przez nieroztropne utworzenie warunku pętli dopuścimy do próby odczytania elementu poza jej skrajnym zakresem (listing 6.2).

LISTING 6.2. Program iterujący po tablicy liczb całkowitych, w którym następuje wyjście poza zakres tablicy

```
public class ArraysExample {
    public static void main(String[] args) {
        int [] array = {1, 2, 3};
        for (int i = 0; i <= array.length; i++) {
            System.out.print(array[i] + " ");
        }
    }
}
```

Uruchomienie powyższego kodu spowoduje wypisanie na konsoli podobnego komunikatu o błędzie, z tym że teraz próbujemy odwołać się do indeksu 3 poprzez niepoprawną konstrukcję warunku pętli:

```
1 2 3 Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 3 out
of bounds for length 3
```

Błędnie założyliśmy iteracje aż do osiągnięcia indeksu równego długości tablicy `i <= array.length`. Zmienna `i` osiągnęła wartość 3, warunek jest spełniony, ale maksymalny indeks w tym przypadku to 2. Pamiętajmy, że w tablicach komórki numerujemy począwszy od zera!

Konstrukcja algorytmów opartych na tablicach

Tablice używane do agregowania danych tego samego typu służą często jako zbiór, na podstawie którego obliczane są cechujące go parametry. Przykładowo można policzyć wystąpienia liczb z danego zakresu, obliczyć ich średnią, znaleźć liczbę liczb ujemnych czy zweryfikować zawarte w tablicy znaki pod kątem dopasowania do pewnych wzorców.

Obliczanie liczby wystąpień liczb dodatnich i ujemnych w tablicy

LISTING 6.3. Program zliczający liczbę wystąpień liczb dodatnich oraz ujemnych w tablicy

```
public class ArraysExample {
    public static void main(String[] args) {
        int [] array = {65, 123, 43, -87, -764, -934};
        int positiveSum = 0, negativeSum = 0;
        for (int element : array) {
            if (element >= 0) {
                positiveSum++;
            } else {
                negativeSum++;
            }
        }
        System.out.println("Tablica składa się z " + positiveSum + " liczb dodatnich oraz "
            + negativeSum + " liczb ujemnych.");
    }
}
```

Program pokazany na listingu 6.3 zlicza liczbę wystąpień liczb dodatnich i ujemnych w tablicy typu `int`. Warto podkreślić, że liczba 0 będzie potraktowana jako dodatnia. Po uruchomieniu wypisze się na konsoli następujące zdanie:

Tablica składa się z 3 liczb dodatnich oraz 3 liczb ujemnych.

Obliczanie średniej z liczb w tablicy

LISTING 6.4. Program obliczający średnią arytmetyczną z liczb w tablicy

```
public class ArraysExample {
    public static void main(String[] args) {
        int [] array = {1, 54, 79, 43, 14, 34};
        int sum = 0;
        double average;
        for (int element : array) {
            sum += element;
        }
        average = sum / (double) array.length;
        System.out.println("Średnia z liczb w tablicy wynosi " + average);
    }
}
```

Listing 6.4 prezentuje algorytm wyliczania średniej na podstawie wszystkich liczb w tablicy. Po uruchomieniu wypisze na konsoli następujące zdanie:

Średnia z liczb w tablicy wynosi 37.5

Zwróć uwagę, że przy dzieleniu sumy przez liczbę elementów jeden ze składników działania jest rzutowany na typ `double`, aby wynik był liczbą zmiennoprzecinkową. Dzielenie przez siebie dwóch liczb typu `int` zawsze zwróci wynik w postaci liczby całkowitej, a część przecinkowa jest odrzucana. Dzieje się tak dlatego, że wynik działania operatorów arytmetycznych jest takiego typu jak najszerszy zakres użyty w obliczeniu. Stąd jeśli jedną z wartości rzutujemy na `i`, wynik będzie zwrócony jako `double`, gdyż jest pojemniejszym typem niż `int`. Aby sprawdzić, czy rzeczywiście tak

jest, najlepiej uruchomić program ponownie, pozbywając się jednak rzutowania na typ `double`. Po ponownym uruchomieniu na konsoli pojawi się poniższe zdanie:

Średnia liczb z tablicy wynosi 37.0

Obliczanie liczby wielkich i małych liter alfabetu w tablicy

LISTING 6.5. Program obliczający liczbę wielkich i małych liter w tablicy

```
public class ArraysExample {
    public static void main(String[] args) {
        char [] array = {'a', 'R', 'U', 'g', 'd', 'z', 'P'};
        int upperCaseSum = 0, lowerCaseSum = 0;
        for (int element : array) {
            if (element >= 65 && element <= 90) {
                upperCaseSum++;
            } else if (element >= 97 && element <= 122) {
                lowerCaseSum++;
            }
        }
        System.out.println("Tablica składa się z " + upperCaseSum + " wielkich liter oraz "
            + lowerCaseSum + " małych liter.");
    }
}
```

Listing 6.5 prezentuje program sprawdzający liczbę wielkich i małych liter zgromadzonych w tablicy typu `char`. Algorytm opiera się na wykorzystaniu automatycznej konwersji typu `char` na `int`, gdzie liczba całkowita odpowiadająca znakowi stanowi jego przyporządkowanie w tablicy ASCII. Znając kolejne numery znaków otwierających i zamykających alfabet, na podstawie ich wielkości możemy w prosty sposób zweryfikować, czy liczba reprezentująca znak należy do zakresu dużych czy małych liter. Uruchomienie programu spowoduje wypisanie na ekran następującego napisu:

Tablica składa się z 3 wielkich liter oraz 4 małych liter.

Sprawdzanie, czy znaki zawarte w tablicy tworzą palindrom

Palindrom to wyrażenie, które czytane od lewej do prawej strony i odwrotnie brzmi tak samo, co technicznie oznacza, że znaki składające się na nie zachowują symetrię, jeśli ujmemy je jako całość. Przykład programu sprawdzającego, czy podany w tablicy ciąg znaków jest palindromem, jest pokazany na listingu 6.6.

LISTING 6.6. Program sprawdzający, czy podany w tablicy ciąg znaków jest palindromem

```
public class ArraysExample {
    public static void main(String[] args) {
        char [] array = {'k', 'a', 'j', 'a', 'k'};
        boolean isPalindrome = true;
        int arrayLength = array.length;
        for (int i = 0; i < arrayLength / 2; i++) {
            if (array[i] != array[arrayLength - i - 1]) {
                isPalindrome = false;
            }
        }
    }
}
```



```

    }
    System.out.println("Czy znaki w tablicy tworzą palindrom? " + (isPalindrome ?
    ↪ "TAK" : "NIE"));
}
}

```

Uruchomienie listingu spowoduje wypisanie na konsoli następującego zdania:

Czy znaki w tablicy tworzą palindrom? TAK

Powyższy program weryfikuje, czy tablica typu `char` zawiera palindrom. W tym celu w pętli `for` odczytywane są jednocześnie komórki od lewej i prawej strony, w każdej iteracji z przesunięciem o jedną pozycję do środka. Jeśli porównywane znaki różnią się od siebie, z góry zakładamy, że nie jest to palindrom. Ale gdy pętla skończy działanie po osiągnięciu połowy długości tablicy i nie będzie różnic w znakach, wynik sprawdzenia będzie pozytywny. Zwróć uwagę, że pętla obraca się tyle razy, ile wynosi iloraz długości tablicy, a w celu zajrzenia do ostatniej komórki posługujemy się jej długością pomniejszoną o 1 w pierwszym obrocie pętli, aby uniknąć wyjścia poza zdefiniowany zakres. Ostatni indeks, dla którego możemy odczytać zawartość komórki, to 4 przy 5-elementowej tablicy.

W celu sprawdzenia różnych wariantów spróbujemy uruchomić ponownie program, przekazując tablicę z parzystą liczbą elementów, przykładowo zawierającą słowo ANNA.

```
char [] array = {'A', 'N', 'N', 'A'};
```

Po ponownym uruchomieniu na konsoli znów wypisze się zdanie:

Czy znaki w tablicy tworzą palindrom? TAK

Tablice wielowymiarowe

Tablice omawiane w poprzednim podrozdziale nazywa się jednowymiarowymi ze względu na tylko jeden wymiar, który można zamodelować w przestrzeni. Takie tablice charakteryzuje tylko długość. Niemniej jednak w Javie możemy tworzyć także tablice wielowymiarowe. Przykładowo tablica dwuwymiarowa będzie miała długość i szerokość, a trójwymiarowa dodatkowo głębokość. Ale to nie wyczerpuje możliwych do zdefiniowania wymiarów. Możemy utworzyć tablicę o dowolnej krotności wymiarów, lecz trudniej będzie znaleźć zastosowanie w algorytmach na przykład dla tablic pięciowymiarowych.

Tablica dwuwymiarowa

Tablica dwuwymiarowa może stanowić macierz elementów, w której będziemy poruszać się w dwóch osiach — przyjmijmy X i Y, jak na rysunku 6.2, przedstawiającym liczby całkowite.

		X									
		0	1	2	3	4	5	6	7	8	9
Y	0	1	2	3	4	5	6	7	8	9	10
	1	11	12	13	14	15	16	17	18	19	20
	2	21	22	23	24	25	26	27	28	29	30
	3	31	32	33	34	35	36	37	38	39	40
	4	41	42	43	44	45	46	47	48	49	50

RYSUNEK 6.2. Tablica dwuwymiarowa typu całkowitoliczbowego

Tablicę wielowymiarową w Javie tworzy się niemal identycznie jak jednowymiarową. Należy jedynie użyć operatorów tablicowych odpowiednio do liczby wymiarów.

```
int [][] array2D = new int[5][10];
```

Powyższy listing przedstawia definicję tablicy dwuwymiarowej, gdzie deklarujemy 5 wierszy i 10 kolumn. Aby uzupełnić tablicę liczbami od 1 do 50, tak jak na rysunku, można posłużyć się implementacją składającą się z dwóch pętli for zagnieżdżonych jedna w drugiej.

```
int counter = 1;
int[][] array2D = new int[5][10];
for (int i = 0; i < array2D.length; i++) {
    for (int j = 0; j < array2D[i].length; j++) {
        array2D[i][j] = counter++;
    }
}
```

Pętla zewnętrzna odpowiada za iterowanie po wierszach (oś X), a pętla wewnętrzna — po kolumnach (oś Y). Warunek pętli sprawdzający, czy w kolejnych obrotach zmienne *i* oraz *j* odwołują się do odpowiedniego zakresu tablicy, jest tak skonstruowany, że pobiera odpowiednią długość tablicy *i* i sprawdza, czy indeksy są od niej mniejsze. Tablica `array2D.length` zwraca liczbę wierszy, `array2D[i].length` zaś — liczbę kolumn w danym wierszu. Jeśli tak na to spojrzeć, można dojść do wniosku, że tablica dwuwymiarowa to po prostu tablica tablic. W każdym zerowym indeksie kolejnego wiersza znajduje się tablica z kolumnami, jak pokazano na rysunku 6.3.

		X									
		0	1	2	3	4	5	6	7	8	9
Y	0	1	2	3	4	5	6	7	8	9	10
	1	11	12	13	14	15	16	17	18	19	20
	2	21	22	23	24	25	26	27	28	29	30
	3	31	32	33	34	35	36	37	38	39	40
	4	41	42	43	44	45	46	47	48	49	50

RYSUNEK 6.3. Tablica dwuwymiarowa typu całkowitoliczbowego

Aby odczytać elementy takiej tablicy, ponownie wystarczy użyć pętli zagnieżdżonych (listing 6.7).

LISTING 6.7. Program uzupełniający i wypisujący zawartość tablicy dwuwymiarowej typu całkowitoliczbowego

```
public class ArraysExample {
    public static void main(String[] args) {
        int counter = 1;
        int[][] array2D = new int[5][10];
        //Wypełnianie tablicy liczbami od 1 do 50
        for (int i = 0; i < array2D.length; i++) {
            for (int j = 0; j < array2D[i].length; j++) {
                array2D[i][j] = counter++;
            }
        }
        //Wypisanie zawartości tablicy na konsolę
        for (int i = 0; i < array2D.length; i++) {
            for (int j = 0; j < array2D[i].length; j++) {
                System.out.print(array2D[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Powyższy listing prezentuje odczyt w ramach pętli wewnętrznej kolejnych komórek tablicy dwuwymiarowej, poczynwszy od komórki o współrzędnych [0] [0], gdzie znajduje się liczba 1. Uruchomienie programu wyświetli na konsoli następujący wynik:

```
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
```

Przykładowo, gdybyśmy chcieli odczytać liczbę 37, powinniśmy szukać jej na osi X i Y pod indeksami [3] [6]:

```
System.out.println(array2D[3][6]);
```

Jeśli już wiemy, że tablica dwuwymiarowa to tablica tablic, nic nie stoi na przeszkodzie, aby tworzyć struktury o nieregularnych wymiarach, a także wykorzystać inicjalizację za pomocą nawiasów klamrowych.

Rysunek 6.4 przedstawia tablicę, w której pierwszy wiersz składa się z 10 kolumn, drugi z 7, trzeci z 9, czwarty z 6, piąty zaś z 3.

		X									
		0	1	2	3	4	5	6	7	8	9
Y	0	1	2	3	4	5	6	7	8	9	10
	1	11	12	13	14	15	16	17			
	2	18	19	20	21	22	23	24	25	26	
	3	27	28	29	30	31	32				
	4	33	34	35							

RYSUNEK 6.4. Tablica dwuwymiarowa typu całkowitoliczbowego o nieregularnych wymiarach

W celu utworzenia tablicy o niestandardowych wymiarach należy do każdej komórki tablicy reprezentującej wiersze przypisać tablicę z kolumnami o odpowiedniej długości.

```
int [][] array2D = new int[5][];
array2D[0] = new int[10];
array2D[1] = new int[7];
array2D[2] = new int[9];
array2D[3] = new int[6];
array2D[4] = new int[3];
```

Tak zdefiniowana tablica będzie zawierać same zera, co można sprawdzić przez przeiterowanie po jej zawartości z wykorzystaniem pętli zagnieżdżonych z poprzedniego przykładu (listing 6.8).

LISTING 6.8. Program uzupełniający i wypisujący zawartość tablicy dwuwymiarowej typu całkowitoliczbowego o nieregularnym wymiarze

```
public class ArraysExample {
    public static void main(String[] args) {
        int [][] array2D = new int[5][];
        array2D[0] = new int[10];
        array2D[1] = new int[7];
        array2D[2] = new int[9];
        array2D[3] = new int[6];
        array2D[4] = new int[3];
        for (int i = 0; i < array2D.length; i++) {
            for (int j = 0; j < array2D[i].length; j++) {
                System.out.print(array2D[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Uruchomienie powyższego kodu spowoduje wypisanie na konsoli wyniku:

```
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0
0 0 0
```

Aby zera zastąpić liczbami od 1 do 35, wystarczy uruchomić ponownie pętlę z listingu 6.7, uzupełniającego tablicę dwuwymiarową. Następnie możemy ponownie wypisać zawartość tablicy array2D na ekran (listing 6.9).

LISTING 6.9. Program uzupełniający i wypisujący zawartość tablicy dwuwymiarowej typu całkowitoliczbowego o nieregularnym wymiarze

```
public class ArraysExample {
    public static void main(String[] args) {
        int [][] array2D = new int[5][];
        array2D[0] = new int[10];
        array2D[1] = new int[7];
        array2D[2] = new int[9];
        array2D[3] = new int[6];
```

```

        array2D[4] = new int[3];
        int counter = 1;
        for (int i = 0; i < array2D.length; i++) {
            for (int j = 0; j < array2D[i].length; j++) {
                array2D[i][j] = counter++;
            }
        }
        for (int i = 0; i < array2D.length; i++) {
            for (int j = 0; j < array2D[i].length; j++) {
                System.out.print(array2D[i][j] + " ");
            }
            System.out.println();
        }
    }
}

```

Uruchomienie powyższego kodu spowoduje wypisanie na konsoli następującego wyniku:

```

1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24 25 26
27 28 29 30 31 32
33 34 35

```

W celu uproszczenia definicji tablicy wielowymiarowej, jeśli z góry znamy elementy, które chcemy w niej przechować, możemy posłużyć się inicjalizacją z wykorzystaniem nawiasów klamrowych, znaną z tablic jednowymiarowych:

```

int [][] array2D = {
    {1, 2, 3, 4, 5, 6, 7, 8, 9, 10},
    {11, 12, 13, 14, 15, 16, 17},
    {18, 19, 20, 21, 22, 23, 24, 25, 26},
    {27, 28, 29, 30, 31, 32},
    {33, 34, 35}
};

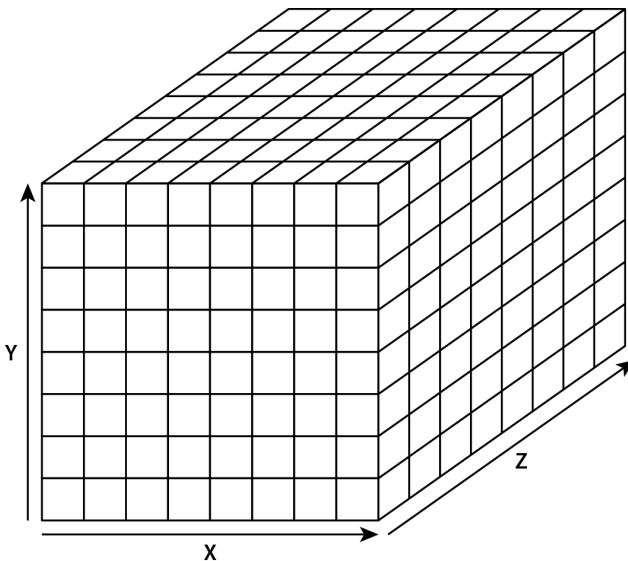
```

Powyższy listing prezentuje tworzenie tablicy dwuwymiarowej o różnej liczbie kolumn w poszczególnych wierszach. Jest to odpowiednik sposobu inicjalizacji tablicy z użyciem pętli zagnieżdżonych.

Tablice dwuwymiarowe stosuje się w implementacji algorytmów wykorzystywanych na przykład w grafice 2D, kartografii lub matematyce, gdy trzeba wizualizować dane na płaszczyźnie określonej przez szerokość i wysokość. Traktując takie tablice jak układ współrzędnych, możemy precyzyjnie wskazać na daną komórkę przez odniesienie się do indeksów tablic, co sprzyja przetwarzaniu ich w pętlach zagnieżdżonych.

Tablica trójwymiarowa

Analogicznie jak w tablicach dwuwymiarowych, operowanie na trzech wymiarach wymaga tylko użycia odpowiedniej liczby pętli. Aby zobrazować sobie taką strukturę, wystarczy wyobrazić sobie kostkę Rubika. Mamy do czynienia z trzema osiami: X, Y, Z, odpowiadającymi, odpowiednio, szerokości, wysokości i głębokości (rysunek 6.5).



RYSUNEK 6.5. Tablica trójwymiarowa

Kostka Rubika to szczególny przykład wykorzystywany do zobrazowania tablicy trójwymiarowej jako sześcienniej figury geometrycznej. Niech jednak nie zwiedzie Cię ta wizualizacja, ponieważ tablica trójwymiarowa wcale nie musi mieć wszystkich wymiarów równych! Osie X, Y, Z mogą przyjmować różne rozmiary, ale odpowiednie wykorzystanie warunków pętli pozwala na bezpieczne operowanie nawet na takich nieregularnych strukturach. Na listingu 6.10 pokazany jest przykład utworzenia sześcienniej tablicy, uzupełnienia jej danymi oraz wypisania na ekran jej zawartości.

LISTING 6.10. Program uzupełniający i wypisujący zawartość tablicy trójwymiarowej typu całkowitoliczbowego

```
public class ArraysExample {
    public static void main(String[] args) {
        int[][][] array3D = new int[3][3][3];
        int counter = 1;
        for (int i = 0; i < array3D.length; i++) {
            for (int j = 0; j < array3D[i].length; j++) {
                for (int k = 0; k < array3D[i][j].length; k++) {
                    array3D[i][j][k] = counter++;
                }
            }
        }
        for (int i = 0; i < array3D.length; i++) {
            for (int j = 0; j < array3D[i].length; j++) {
                for (int k = 0; k < array3D[i][j].length; k++) {
                    System.out.print(array3D[i][j][k] + " ");
                }
                System.out.println();
            }
            System.out.println();
        }
    }
}
```

Skoro mamy do czynienia z sześcianiem o boku równym 3, oznacza to, że liczba elementów możliwych do przechowania w takiej tablicy trójwymiarowej wynosi 27, zgodnie z działaniem 3^3 . Uruchomienie programu z powyższego listingu spowoduje wypisanie na ekran następującego wyniku:

```
1 2 3
4 5 6
7 8 9

10 11 12
13 14 15
16 17 18

19 20 21
22 23 24
25 26 27
```

Liczby zostały pogrupowane w ten sposób, że po pierwszym obrocie pętli zewnętrznej otrzymujemy zawartość tablicy dwuwymiarowej będącej pierwszą, frontową warstwą, a kolejne iteracje powodują wyświetlenie drugiej i trzeciej warstwy sześcianu. Tablice trójwymiarowe mają zastosowanie na przykład w grafice i modelowaniu przestrzennym, kiedy trzeba określić współrzędne danego punktu w określonym układzie. Przykładowo, gdy chcemy określić położenie lecącego samolotu, podamy nie tylko długość i szerokość geograficzną, ale także wysokość nad poziomem morza. Określenie lokalizacji w przestrzeni powietrznej z użyciem tylko dwóch wymiarów nie będzie precyzyjne.

Skorowidz

A

- adnotacje, 125
 - dziedziczone, 131
 - parametry, 128
 - tworzenie, 126
 - używanie, 129
 - wbudowane, 126
- API
 - klasy String, 151
 - klasy StringBuilder, 160
- argument wieloznaczny, 223
- argumenty
 - generyczne, 223
 - kolejność, 92
 - zmienna liczba, 92
- ArrayDeque, 259
 - API, 261
 - konstruktory, 261
 - schemat struktury, 260
 - złożoność, 261
- ArrayList, 241
 - API, 242
 - konstruktory, 242
 - lista typu String, 241
 - schemat struktury, 241
 - złożoność, 242
- atomowa operacja, 435
- atrybuty, 71
 - statyczne, 98
- autoboxing, 139

B

- biblioteka
 - JDBC, 445
 - joda-time, 417
- BiFunction, 324
- BinaryOperator, 326
- blok kodu inicjalizacyjny
 - instancyjny, 104
 - statyczny, 105

błędy

- kompilacji, 137, 220
- rzutowania, 134, 209
- BST, Binary Search Tree, 309
- bufory, 399

C

- Comparator, 330
- Consumer, 327
- czas, 407

D

- data i czas, 412
 - formatowanie, 415, 424
 - parsowanie, 416, 425
- DCL, Data Control Language, 450
- DDL, Data Definition Language, 450
- deklaracja
 - klasy, 18, 69
 - tablicy, 56
 - typu generycznego, 211
- deserializacja, 397
- diamond operator, 211
- DML, Data Manipulation Language, 449
- DQL, Data Query Language, 450
- DRY, Don't Repeat Yourself, 79
- drzewo przeszukiwań BST, 285, 286, 309
- dziedziczenie, 79
 - klas wyjątków, 167, 169, 186
 - metod, 186

E

- enkapsulacja, 73
- etykiety pętli, 54

F

- FIFO, First In First Out, 255, 259, 265
- formatowanie daty i czasu, 415, 424
- Function, 322

G

Garbage Collector, 15, 88
głęboka kopia, deep copy, 197

H

HashMap, 301
 API, 301
 konstruktor, 301
HashSet, 272
 API, 277
 konstruktor, 276
 schemat struktury, 275
 złożoność, 276
haszowanie, 273

I

IDE, Integrated Development
 Environment, 21
identyfikatory, 20
instrukcja
 break, 53
 continue, 52
 if-else, 42
 switch, 44
 throw, 173
 throws, 172
interfejs, 86
 CallableStatement, 459
 Collection, 239
 Comparable, 230, 237
 Connection, 447
 DataSource, 449
 funkcyjny, 318
 BiFunction, 324
 BinaryOperator, 326
 Comparator, 235, 237, 330
 Consumer, 327
 Function, 322
 Predicate, 326
 Supplier, 329
 UnaryOperator, 325
 Iterable, 239
 Iterator, 298
 List, 240
 Map, 300
 NavigableMap, 309
 NavigableSet, 283
 Path, 402
 PreparedStatement, 455
 Queue, 265
 ResultSet, 451

Runnable, 431
Serializable, 395
Set, 272
SortedMap, 309
Statement, 450
Stream, 337

interfejsy, 86
 funkcyjne, 318
 tworzenie, 319
 wbudowane, 322
iteratory, 298

J

Java, 13
 instalacja, 17
 maszyna wirtualna Javy, 15
 założenia, 14
JDBC, Java Database Connectivity, 445
 architektura systemu, 446
 Connection, 447
 DataSource, 449
 DriverManager, 448
 komponenty, 445
 nawiazywanie połączenia, 447
JDK, Java Development Kit, 17
JEE, Java Enterprise Edition, 17
język
 Java, 13
 SQL, 449
JRE, Java Runtime Environment, 17
JSE, Java Standard Edition, 17
JVM, Java Virtual Machine, 15

K

kalendarz, 408
 gregoriański, 410
kanały, 400
klasa
 ArrayDeque, 259
 BigDecimal, 146
 BigInteger, 143
 Buffer, 399
 Calendar, 408
 Channel, 400
 Class, 118
 Date, 414
 driverManager, 448
 File, 370
 Files, 403
 GregorianCalendar, 410
 HashMap, 301
 InputStream, 378, 390

klasa

- LinkedHashMap, 307
- LocalDate, 418
- LocalDateTime, 421
- Locale, 409
- LocalTime, 419
- Object, 193
- Optional, 366
- OutputStream, 379
- Paths, 403
- PrintStream, 391
- RandomAccessFile, 392
- Reader, 385
- Scanner, 391
- Selector, 401
- String, 149
- StringBuffer, 160
- StringBuilder, 160
- Thread, 427, 428
- TimeZone, 409
- TreeMap, 309
- TreeSet, 283
- Vector, 255
- Writer, 386
- ZonedDateTime, 422

klasy, 69

- abstrakcyjne, 82
- anonimowe, 112
 - cechy, 114
- atrybuty, 71
- dziedziczenie, 79
- enkapsulacja, 73
- generyczne, 211
- generyczne wieloparametrowe, 214
- konstruktory, 75
- lokalne, 110
- metody, 71
- modyfikatory widoczności, 71
- przeładowanie metod, 90
- statyczne, 114
- statyczne składowe, 98
- wewnętrzne, 106
- wyjątków, 167
- wyjątków wbudowane, 168
- zagnieżdżanie, 106
- zewnętrzne, 107

klauzula

- try-catch, 175
- try-with-resources, 189

kolejki, 259, 265–272

- typu FIFO, 255, 259, 265
- typu LIFO, 255, 259

kolekcja

- ArrayDeque, 259
- ArrayList, 241
- HashSet, 272
- LinkedHashSet, 282
- LinkedList, 253
- PriorityQueue, 265
- Stack, 257
- TreeSet, 283

kolekcje

- hierarchia interfejsów i klas, 239
- kolejki, 265
- listy, 240
- mapy, 300
- niemodyfikowalne, 312
- zbiory, 272

kolizja haszy, hash collision, 275, 278

konstrukcja

- try-with-resources, 382
- varargs, 93

konstruktory, 75

- klasy StringBuilder, 161

kubek, bucket, 278

L

liczby zmiennoprzecinkowe, 30

LIFO, Last In First Out, 255, 259

LinkedHashMap, 307

- API, 308
- konstruktory, 307

LinkedHashSet, 282

- API, 283
- konstruktory, 282
- złożoność, 282

LinkedList, 253

- API, 255
- konstruktory, 254
- schemat struktury, 254
- złożoność, 254

listy, 240–264

literały, 20

M

mapowanie ResultSet, 460

mapy, 300–312

maszyna wirtualna Javy, 15

metoda, 71

- accept(), 375
- add(), 213, 239, 265, 277, 408
- addAll(), 239, 244, 289
- addFirst(), 260

addLast(), 260
after(), 414
allMatch(), 338, 354
allocate(), 400
anyMatch(), 338, 354
append(), 92, 161, 386
asList(), 245
atDate(), 420
atTime(), 419
atZone(), 421
available(), 379
before(), 414
binarySearch(), 316
canExecute(), 370, 371
canRead(), 370
canWrite(), 370
capacity(), 162
ceiling(), 284, 296
ceilingEntry(), 310
ceilingKey(), 310
charAt(), 155
clear(), 240, 249, 400
clone(), 194
close(), 379, 385, 386, 450
collect(), 338, 358
comparator(), 284, 309
compare(), 235, 237
compareTo(), 159, 230–234, 289, 290
concat(), 152
contains(), 157, 240, 250, 281
containsAll(), 240
containsKey(), 300, 304
containsValue(), 300, 304
copy(), 404
count(), 338
createDirectory(), 404
createFile(), 403
createNewFile(), 371
currentThread(), 429
currentTimeMillis(), 413
delete(), 162, 371
deleteIfExists(), 404
descendingKeySet(), 310
descendingMap(), 310
descendingSet(), 284, 297
distinct(), 337, 348
dropWhile(), 337, 351
element(), 265, 270
elementAt(), 256
empty(), 367
entrySet(), 300, 306
equals(), 117, 158, 198–201, 204, 276–278
equalsIgnoreCase(), 159
execute(), 450
executeQuery(), 450
executeUpdate(), 450
exists(), 223–225, 371–374, 404
filter(), 345, 368
filter(Predicate<? super T> predicate), 337
finalize(), 207
findAny(), 338, 356
findFirst(), 338, 356
first(), 284, 294, 451
firstElement(), 256
firstEntry(), 310
firstKey(), 309
flatMap(), 337, 349
flip(), 400
floor(), 284, 296
floorEntry(), 310
floorKey(), 310
flush(), 379, 386, 390
forEach(), 337
forEachRemaining(), 298
forLanguageTag(), 409
format(), 419, 420
get(), 213, 246, 303, 367, 400, 408
getAbsoluteFile(), 370
getAbsolutePath(), 370
getBigDecimal(), 451
getClass(), 118
getConnection(), 448, 449
getDate(), 451
getDayOfMonth(), 418
getDayOfYear(), 418
getDefault(), 409
getDouble(), 451
getFileName(), 402
getFilePointer(), 393
getFirst(), 260
getHour(), 420
getInstance(), 408, 410
getInt(), 451
getLast(), 260
getMinute(), 420
getMonthValue(), 418
getName(), 370, 402, 429
getNameCount(), 402
getNano(), 420
getParent(), 370, 402
getParentFile(), 370
getPath(), 370
getPriority(), 429
getRoot(), 402
getSearchKey(), 225

metoda

- getSecond(), 420
- getString(), 451
- getTime(), 409, 414
- getTimeInMillis(), 409
- getTimeZone(), 409
- getYear(), 418
- hashCode(), 117, 201–204, 276–277
- hasNext(), 298
- hasRemaining(), 400
- headMap(), 309, 310
- headSet(), 284, 294
- higher(), 284, 295
- higherEntry(), 310
- higherKey(), 310
- ifPresent(), 367
- ifPresentOrElse(), 367
- indexOf(), 157, 241, 247
- insert(), 161
- insertElementAt(), 256
- interrupt(), 429
- isAbsolute(), 370, 402
- isAfter(), 418, 420, 421
- isAlive(), 429
- isBefore(), 418, 420, 421
- isBlank(), 156
- isDirectory(), 371, 374
- isEmpty(), 156, 240, 300, 367
- isEqual(), 418, 421
- isExecutable(), 404
- isFile(), 371
- isHidden(), 371
- isInfinite(), 142
- isInterrupted(), 429
- isLeapYear(), 418
- isNaN(), 142
- isPresent(), 367
- isReadable(), 404
- isSameFile(), 404
- isWritable(), 404
- iterate(), 337, 345
- iterator(), 240, 298, 402
- join(), 429, 431
- keySet(), 300, 304
- last(), 284, 294, 451
- lastElement(), 256
- lastEntry(), 310
- lastIndexOf(), 157, 241, 247
- lastKey(), 309
- lastModified(), 371
- length(), 156, 371, 393
- lines(), 156
- list(), 371, 374, 405
- listFiles(), 371
- lower(), 284, 295
- lowerEntry(), 309
- lowerKey(), 310
- main(), 19
- map(), 337, 348, 368
- mark(), 379, 385
- markSupported(), 379, 385
- MathContext(), 147
- max(), 338, 353
- min(), 338, 353
- minus(), 418, 420
- minusDays(), 418
- minusHours(), 419
- minusMinutes(), 419
- minusMonths(), 418
- minusNanos(), 420
- minusSeconds(), 420
- minusWeeks(), 418
- minusYears(), 418
- mkdir(), 371
- move(), 404
- nanoTime(), 413
- navigableKeySet(), 310
- newBufferedReader(), 404
- newBufferedWriter(), 404
- next(), 298, 451
- noneMatch(), 338, 354
- notExists(), 404
- notify(), 206, 438
- notifyAll(), 206, 438, 440
- now(), 417, 419
- of(), 337, 367, 402, 417, 422
- offer(), 265, 269
- offerFirst(), 259
- offerLast(), 260
- ofNullable(), 367, 369
- ofPattern(), 424
- or(), 368
- orElse(), 367
- orElseGet(), 367
- orElseThrow(), 368
- parallelStream(), 363
- parse(), 416, 417, 425
- parseByte(), 141
- peek(), 259, 265, 270, 337, 350
- peekFirst(), 260
- peekLast(), 260
- plus(), 418, 420
- plusDays(), 418
- plusHours(), 419
- plusMinutes(), 419
- plusMonths(), 418

plusNanos(), 420
 plusSeconds(), 419
 plusWeeks(), 418
 plusYears(), 418
 poll(), 265, 271
 pollFirst(), 260, 284, 296, 297
 pollFirstEntry(), 310
 pollLast(), 260, 284, 296, 297
 pollLastEntry(), 310
 previous(), 451
 print(), 19, 391
 printf(), 391
 println(), 391
 put(), 300, 301, 400
 putAll(), 300
 read(), 378, 385, 392
 readAllBytes(), 378, 405
 readAllLines(), 405
 readLine(), 390, 392
 readNBytes(), 378
 readObject(), 395, 396
 readString(), 405
 ready(), 385
 reduce(), 338, 352
 removeElement(), 257
 remove(), 240, 265, 280, 298, 303
 removeAllElements(), 257
 removeFirst(), 260
 removeLast(), 260
 renameTo(), 371
 repeat(), 152
 replace(), 152
 reset(), 379, 385
 reverse(), 162
 rewind(), 400
 run(), 429, 432, 442
 seek(), 393, 394
 set(), 240, 245, 408
 setExecutable(), 371
 setLastModified(), 371
 setPriority(), 429
 setReadable(), 371
 setReadOnly(), 371
 setTime(), 409, 414
 setTimeInMillis(), 409
 setWritable(), 371
 size(), 240, 300, 404
 skip(), 379, 385
 sleep(), 429, 430, 439
 sort(), 315
 sorted(), 337, 347
 start(), 428–430
 startsWith(), 158
 stream(), 336, 368
 strip(), 153
 stripLeading(), 154
 stripTrailing(), 154
 subList(), 241, 251
 subMap(), 309, 310
 subpath(), 402
 subSet(), 284, 291
 substring(), 154, 155
 synchronizedCollection(), 443
 synchronizedList(), 443
 synchronizedMap(), 443
 synchronizedNavigableMap(), 443
 synchronizedNavigableSet(), 443
 synchronizedSortedMap(), 443
 synchronizedSortedSet(), 443
 tailMap(), 309, 310
 tailSet(), 284, 294
 takeWhile(), 337, 351
 toAbsolutePath(), 402
 toArray(), 240, 252, 357
 toCharArray(), 155
 toFile(), 402
 toList(), 338, 358
 toLocalDate(), 421
 toLocalTime(), 421
 toLowerCase(), 151
 toNanoOfDay(), 420
 toSecondOfDay(), 420
 toString(), 117, 205, 243, 269, 414
 toUpperCase(), 152
 transferTo(), 385
 trim(), 154
 truncatedTo(), 420, 421
 unmodifiableList(), 312
 unmodifiableMap(), 312
 unmodifiableNavigableMap(), 312
 unmodifiableNavigableSet(), 312
 unmodifiableSet(), 312
 unmodifiableSortedMap(), 312
 unmodifiableSortedSet(), 312
 valueOf(), 141, 249
 values(), 300, 305
 wait(), 206, 437
 withZoneSameInstant(), 422
 write(), 379, 386, 392, 405
 writeBytes(), 392
 writeString(), 405
 metody, 71
 argumenty, 92
 dopasowanie wersji, 94
 generyczne, 220
 argumenty, 223

metody

interfejsu

- Collection, 239
- Deque, 259
- Iterator, 298
- List, 240
- Map, 300
- NavigableMap, 309
- NavigableSet, 284
- Path, 402
- Queue, 265
- ResultSet, 451
- SortedMap, 309
- Statement, 450
- Stream, 337, 340

klas opakowaniowych, 142

klasy

- ArrayDeque, 259–264
- BigDecimal, 146
- BigInteger, 144
- Buffer, 400
- Calendar, 408
- Class, 118
- Date, 414
- File, 370
- Files, 403
- HashMap, 301–306
- InputStream, 378
- LocalDate, 418
- LocalDateTime, 421
- LocalTime, 419
- Object, 193
- Optional, 367
- OutputStream, 379
- RandomAccessFile, 392
- Reader, 385
- String, 151–160
- Thread, 428
- Vector, 256–258
- Writer, 386

kolekcji

- ArrayList, 242–253
- HashSet, 277–281
- PriorityQueue, 268–272
- TreeSet, 288–298

kończące, terminal, 340

pośrednie, intermediate, 340

przekazywanie argumentów, 96

przeładowanie, 90

statyczne, 101

z adnotacją, 130

model pamięci, 16

modyfikatory widoczności, 71

monitor, 436

muteks, 436

N

nadpisywanie, override, 81

narzędzia JDK

- appletviewer, 17
- jar, 17
- java, 17
- javac, 17
- javadoc, 17
- jdb, 17
- jmap, 17

niemodyfikowalność, unmodifiable, 312

NIO, New Input Output, 399

NIO 2, New Input Output 2, 402

notacja wildcard, 223

O

obiekty, 69

tworzenie, 72

obliczenia zmiennoprzecinkowe, 31

odśmiecacz pamięci, 15

operacje narzędziowe, 312

operator

- <>, 211
- instanceof, 135
- przypisania, 23

operatory, 34

- arytmetyczne, 34
- bitowe, 38
- logiczne, 37
- priorytet, 40
- przypisania, 36

P

pakiet

- NIO, 399
- NIO 2, 402

pakiety, 70

pamięć RAM, 22

para klucz-wartość, 300

parametry generyczne, 211, 214

ograniczenie z dołu, 227

ograniczenie z góry, 224

parsowanie daty i czasu, 416, 425

pętla

- do-while, 47
- for, 49

- for-each, 51, 279
- nieskończona, 186
- while, 46
- pętle, 46
 - etykiety, 54
 - przerwania, 52
- planista, thread scheduler, 430
- pliki, 370
 - metadane, 376
 - operacje, 373
 - ścieżka względna i bezwzględna, 372
- płytką kopia, shallow copy, 196
- polimorfizm, 87, 132
- porównywanie obiektów, 229
 - malejąco, 232
 - na podstawie zmiennych, 233
 - rosnąco, 231
 - zasady, 234
- precyzja, 31
- Predicate, 326
- PriorityQueue, 265
 - API, 268
 - konstruktory, 267
 - schemat struktury, 266
 - złożoność, 267
- proces, 426
- program IntelliJ IDEA, 21
- programowanie funkcyjne, 318
- programowanie
 - współbieżne, 426
 - zorientowane obiektowo, 68
- przekazywanie argumentów
 - przez referencję, 97
 - przez wartość, 96
- przeładowanie metod, overloading, 90
- przetwarzanie strumieniowe, 336
 - współbieżne, 363
- pula stringów, 150

R

- referencja
 - do konstruktora, 333
 - do metody instancyjnej obiektu, 332
 - do metody instancyjnej typu, 333
 - do metody statycznej, 331
 - do obiektów, 89
- Reflection API, 119
- refleksja, 117, 120, 125
- rekordy, 116
- removeAll(), 240

- rzutowanie, 132, 209
 - błędy kompilacji, 137
 - błędy rzutowania, 134
 - liczb, 30
 - w dół, 133
 - w górę, 135

S

- selektory, 401
- semafor, 436
- serializacja, 394
 - obiektu zagnieżdżonego, 398
- słowo kluczowe, 20, 21
 - class, 18, 69
 - extends, 225
 - final, 149, 193
 - native, 193
 - new, 427
 - package, 70, 71
 - private, 71
 - protected, 71
 - public, 18, 71
 - record, 117
 - static, 98, 114
 - super, 227
 - synchronized, 435
 - transient, 397
 - volatile, 437
- sortowanie, 315
- SQL, 449
 - Injection, 455
- Stack, 257
 - API, 258
 - konstruktory, 258
 - złożoność, 258
- stałe
 - typu ElementType, 127
 - typu RetentionPolicy, 128
 - typu RoundingMode, 147, 148
- statyczne
 - składowe klas, 98
 - stałe, 102
 - zmienne, 101
- sterta, heap, 16, 97, 150, 266
- stos, stack, 96
- Stream API, 340
- strefy czasowe, 409, 423
- struktura danych
 - ArrayDeque, 260
 - ArrayList, 241
 - HashSet, 273

- struktura danych
 - LinkedHashSet, 282
 - LinkedList, 253
 - PriorityQueue, 266
 - Stack, 258
 - TreeSet, 285
 - Vector, 255
- strumienie, 336
 - bajtów, 378
 - operacje, 380
 - hierarchia dziedziczenia klas, 378
 - numeryczne, 362
 - obiektów, 337
 - operacje kończące, 352
 - operacje pośrednie, 345
 - tworzenie, 338
 - wejścia-wyjścia, 377
 - współbieżne, 364
 - znaków, 384
 - operacje, 386
- Supplier, 329
- synchronizacja wątków, 433

Ś

- ścieżka, 402
 - względna i bezwzględna, 372

T

- tablice, 55
 - agregowanie danych, 58
 - algorytmy, 58
 - deklaracja, 56
 - dwuwymiarowe, 61
 - haszujące, hashtable, 273, 275
 - operacje, 57
 - trójwymiarowe, 65
- TreeMap, 309
 - API, 311
 - konstruktory, 311
- TreeSet, 283
 - API, 288
 - konstruktory, 288
 - schemat struktury, 285
 - złożoność, 287
- tworzenie
 - adnotacji, 126
 - interfejsu funkcyjnego, 319
 - obiektów, 72
 - programu, 18
 - strumienia, 338
 - wątku, 428
 - zmiennych, 23

- typ danych
 - BigDecimal, 146
 - BigInteger, 143
 - boolean, 25, 33
 - byte, 25, 26
 - char, 25, 32
 - double, 25, 29
 - float, 25, 29
 - int, 25, 28
 - long, 25, 28
 - short, 25, 27
 - String, 149
 - wyliczeniowy enum, 76
- typy
 - danych
 - całkowite, 30
 - prymitywne, 25
 - zmiennoprzecinkowe, 30
 - generyczne, 208
 - błędy kompilacji, 220
 - błędy rzutowania, 209
 - rozszerzanie, 217
 - zawężanie, 218
 - obiektywne wbudowane, 138
 - opakowaniowe, 138
 - tworzenie, 140
 - referencyjne, *Patrz* referencja

U

- UnaryOperator, 325
- unboxing, 139
- utrata precyzji, 31

V

- varargs, 93
- Vector, 255
 - API, 256
 - konstruktory, 255

W

- wątek, 426
 - główny, 427
 - stan
 - BLOCKED, 428
 - RUNNABLE, 428
 - TERMINATED, 428
 - TIMED_WAITING, 428
 - WAITING, 428

wątki

- cykl życia, 427
- dostęp do obiektu, 435
- komunikacja, 437
- synchronizacja, 433
- tworzenie, 428
- wyścig, race condition, 433
- zakleszczenie, 440, 442

wejście-wyjście, I/O, 370

- schemat, 390

wielowątkowość, 426

- bezpieczne kolekcje, 442

winorośl, 286

współbieżny dostęp do zasobu, 365

wyciek pamięci, memory leak, 16

wyjątek

- ClassCastException, 252, 253, 291
- IllegalArgumentException, 293
- IndexOutOfBoundsException, 246
- InterruptedException, 429
- NoSuchElementException, 270, 369
- NotSerializableException, 395

wyjątki, 163, 166

- blok finally, 183
- klauzula try-with-resources, 189
- łapanie, 175
- łapanie wielu typów, 178
- niesprawdzalne, 171
- powiązane dziedziczeniem, 181

rozszerzanie listy, 188

rzucanie, 172

sprawdzalne, 171

użycie, 189

zawężanie listy, 187

wyrażenia lambda, 321

wyszukiwanie, 316

Z

zakleszczenie, deadlock, 440

zapytania SQL, 449

interfejs CallableStatement, 459

interfejs PreparedStatement, 455

interfejs Statement, 450

zbiory, 272–298

zintegrowane środowisko

programistyczne, IDE, 21

zmienne, 22

obiektowe, 132

rzutowanie, 132

statyczne, 98, 101

tworzenie, 23

typu opakowaniowego, 140

znacznik czasu, timestamp, 407

znak

%, 391

?, 224, 457

@, 126

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

Proszę państwa, oto Java

Mówisz: język programowania, myślisz: Java! Jeśli tak, masz rację, ten język niewątpliwie jest filarem współczesnej informatyki. Java należy do czołówek najpopularniejszych technologii i to właśnie nią szczególnie warto się zainteresować na początku nauki kodowania.

Książka **Java. Teoria w praktyce** została pomyślana tak, by krok po kroku przybliżać specyfikę tego języka programowania. Zaczynasz od podstaw — poznasz między innymi główne założenia, zgodnie z którymi działa Java: maszynę wirtualną, zmienne, operatory, instrukcje sterujące i tablice — by następnie przejść do bardziej zaawansowanych zagadnień. Dowiesz się, czym jest programowanie zorientowane obiektowo, zapoznasz się z paradygmatem programowania funkcyjnego i z zagadnieniem przetwarzania strumieniowego.

Ponadto nauczysz się:

- korzystać z plików w swoich programach
- implementować komunikację z bazą danych
- stosować w praktyce pojęcia związane z wielowątkowością
- przetwarzać datę i czas

W efekcie będziesz w pełni przygotowany do tego, by zacząć programować w Javie, a przecież o to chodzi!

Michał Suwała — absolwent studiów licencjackich na UMCS w Lublinie i magisterskich w SGGW na kierunku informatyka. Od dwunastu lat zawodowo zajmuje się programowaniem w języku Java. Obecnie rozwija się w kierunku architektury oprogramowania. Od ośmiu lat z pasją dzieli się wiedzą na temat technologii Java w ramach kompleksowych szkoleń typu bootcamp. Na co dzień pracuje jako lider techniczny — wraz z zespołem zajmuje się wytwarzaniem oprogramowania dla branży telekomunikacyjnej. Ceni proste, ale pragmatyczne rozwiązania. Jest zwolennikiem pracy zespołowej, stawia w niej na szczerść, zaangażowanie i otwartą komunikację. W wolnych chwilach sięga po literaturę popularnonaukową, a także aktywnie spędza czas z rodziną.

PATRONI:



BULLDOGJOB
Think IT

programista



Helion



helion.pl



HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
Sięgnij po więcej! ▶



ISBN 978-83-289-0022-6



Cena: 109,00 zł