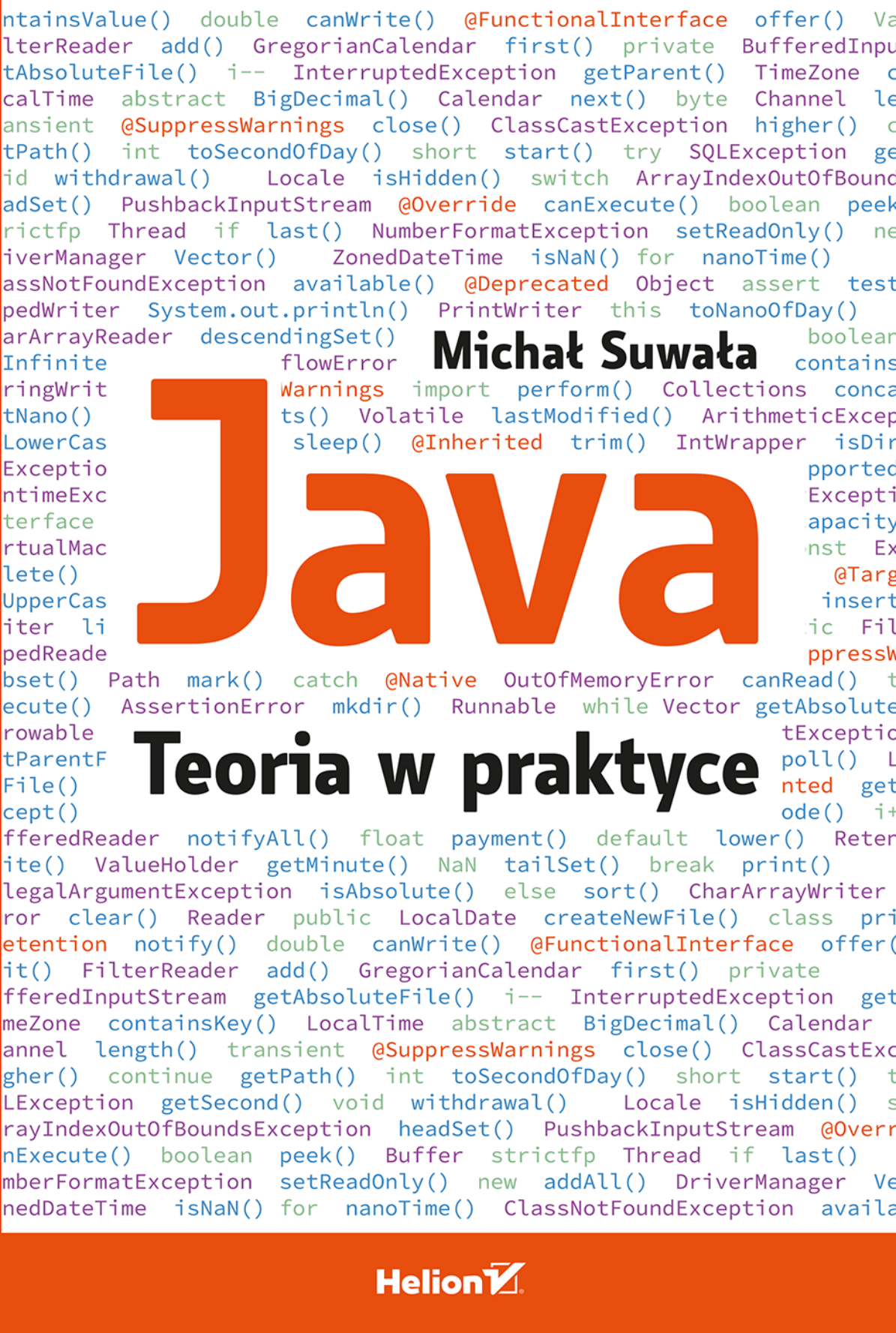

Michał Suwała
Java
Teoria w praktyce
Helion

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

Printed in Poland.

Spis treści

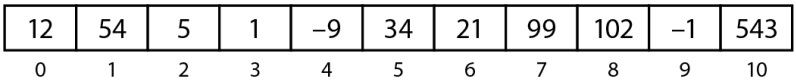
ROZDZIAŁ 6.

Tablice

Poznałeś do tej pory zmienne, które pozwalają na przechowywanie wartości luźno rozmieszczonych w pamięci RAM. Dzięki nim jesteśmy w stanie zapamiętywać cząstkowe 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.



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'`
referencja	`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 ujmiemy 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 spróbujmy 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.

Y \ X	0	1	2	3	4	5	6	7	8	9
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 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.

Y \ X	0	1	2	3	4	5	6	7	8	9
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, począwszy od komórki o współrzędnych `[0][0]`, gdzie znajduje się liczba 1. Uruchomienie programu wyświetli na konsoli następujący wynik:

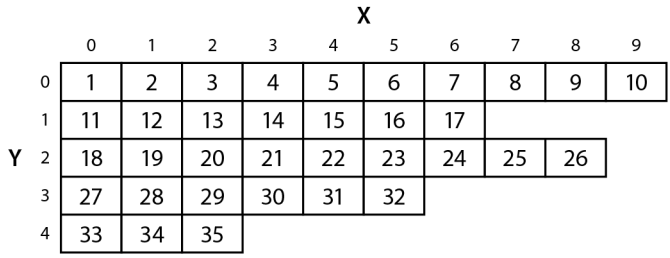
```
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
```

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.



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 pętle 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:

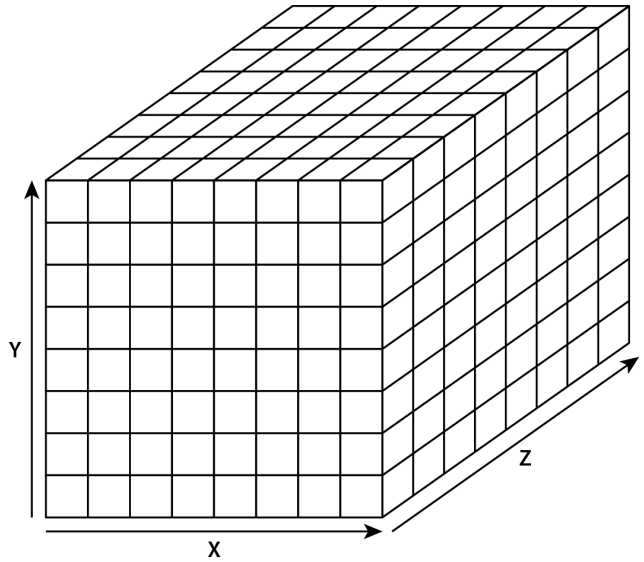
```
int [][] array2D = {
        {1, 2, 3, 4, 5, 6, 7, 8, 9, 10},
        {11, 12, 13, 14, 15, 16, 17},
        {18, 19, 20, 21, 22, 23, 24, 25, 26},
        {27, 28, 29, 30, 31, 32},
        {33, 34, 35}
    };
```

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 sześciennej 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 sześciennej 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 sześcianem 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

G

H

I

J

K

L

M

N

O

P

R

S

Ś

T

U

V

W

Z

Proszę państwa, oto Java

Mówisz: język programowania, myślisz: Java! Jeśli tak, masz rację, ten język niewątpliwie jest filarem współczesnej informatyki. Java należy do czołówki 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. Zaczniesz 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 szczerość, zaangażowanie i otwartą komunikację. W wolnych chwilach sięga po literaturę popularnonaukową, a także aktywnie spędza czas z rodziną.

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

9 788328 900226

Cena: 109,00 zł