KRZYSZTOF KROCZ

Java

Podręcznik na start

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/javpos
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-283-9783-5

Printed in Poland.

Spis treści

Rozdział 2. **Zrób coś ciekawego — początki programowania**

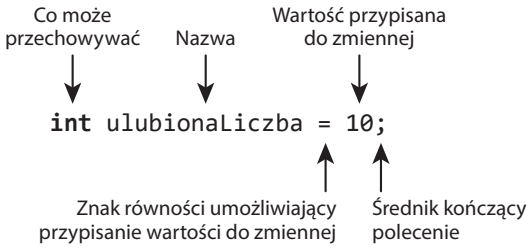
Pudełko na wartości — zmienna

Przyszedł czas, aby poznać kolejne zagadnienie, jakim jest **zmienna**. Można ją porównać do małego pudełka, które pozwala nam przechować pewną wartość, np. liczbę lub pojedynczy znak. Pudełko to musi mieć nazwę oraz typ wartości, która się w nim znajdzie (beczka na wino nie powinna zawierać ropy — w przypadku zmiennych sytuacja wygląda analogicznie). Gdybyśmy chcieli w jakiś sposób zobrazować zmienną, która miałaby reprezentować np. naszą ulubioną liczbę, wyglądałoby to tak:

Charakterystyka zmiennej:
Nazwa: `ulubionaLiczba`
Co może przechowywać: liczby całkowite
Co przechowuje: liczbę 10

Rysunek 2.1. Charakterystyka zmiennej

Teraz trzeba to „przenieść na papier", czyli utworzyć taką zmienną, używając kodu. Aby zadeklarować zmienną, musimy podać kolejno typ oraz nazwę, a następnie za pomocą znaku równości przypisać jej wartość. Poniżej znajdziesz przykład.



Rysunek 2.2. Składnia zmiennej

W ten oto prosty sposób zadeklarowaliśmy swoją ulubioną liczbę w programie. Możesz zadeklarować również pustą zmienną, niezawierającą żadnej wartości, a następnie przypisać jej wartość.

Listing 2.1. Deklaracja zmiennej

```
int ulubionaLiczba; // deklaracja zmiennej (domyślnie dla typu int
                    // wartość będzie równa 0)
ulubionaLiczba = 10; // przypisanie liczby 10 do zmiennej
```

Zapewne zastanawiasz się, co oznacza słowo `int` — określa ono typ liczb całkowitych.

Zmienne mają wiele ciekawych zastosowań. W przypadku zmiennych liczbowych możemy np. wykonywać standardowe operacje dodawania (+), odejmowania (–), mnożenia (*) i dzielenia (/). Spróbujmy zatem zapisać wynik mnożenia ulubionej liczby przez 2 i wyświetlić go w konsoli.

Listing 2.2. Operacja mnożenia z użyciem dodatkowej zmiennej

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

    int ulubionaLiczba = 10;
    int mnoznik = 2;
    int wynikMnozenia = ulubionaLiczba * mnoznik;
    System.out.println(wynikMnozenia);
}
```

W konsoli powinien wyświetlić się wynik 20. Zastanów się przez chwilę — do czego są nam tak naprawdę potrzebne zmienne? Czy nie moglibyśmy od razu napisać `10 * 2` i wyświetlić oczekiwanego wyniku? Kod programu wydałby się wtedy krótszy, a jego działanie pozostałoby takie samo.

Listing 2.3. Operacja mnożenia bez użycia dodatkowej zmiennej

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

    int wynikMnozenia = 10 * 2;
    System.out.println(wynikMnozenia);
}
```

Z pozoru może się to wydawać prostsze, jednak zmienne mają tę przewagę, że nazywają daną wartość. Skąd mamy wiedzieć, czym tak naprawdę są wskazane liczby: 10 i 2? Czy 10 stanowi cenę, wagę, długość lub szerokość czegoś? W przypadku zapisu bez użycia zmiennych nie znamy odpowiedzi na to pytanie. Pamiętaj o tym, aby program był czytelny i aby łatwiej się z nim pracowało.

Zmienne w Javie możemy nazywać tak, jak chcemy, jednak nie możemy używać polskich znaków ani stosować spacji. Przyjęło się, że w celu oddzielenia wyrazów początek zmiennej pisany jest małą literą, a każde kolejne słowo — dużą, jak np. `toJestBardzoDlugaZmienna`.

No dobrze, poznałeś już typ liczb całkowitych, pora więc omówić kolejny, czyli typ liczb zmiennoprzecinkowych (z możliwymi wartościami ułamkowymi).

W Javie mamy do czynienia z dwoma podstawowymi typami przechowywania takich liczb — są to `float` i `double`. Różnią się one od siebie przede wszystkim pojemnością oraz precyzją (`float` może przechowywać nieco mniejsze liczby niż `double` — dokładne pojemności przedstawione są w tabeli 2.1). Należy pamiętać o tym, że część ułamkową i całkowitą oddzielamy za pomocą kropki (`.`), a nie przecinka. Napiszmy zatem prosty program dodający do siebie dwie liczby zmiennoprzecinkowe z użyciem typu `double`.

Listing 2.4. Dodawanie liczb zmiennoprzecinkowych

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

    double x = 1.2;
    double y = 1.3;
    System.out.println(x + y);
}
```

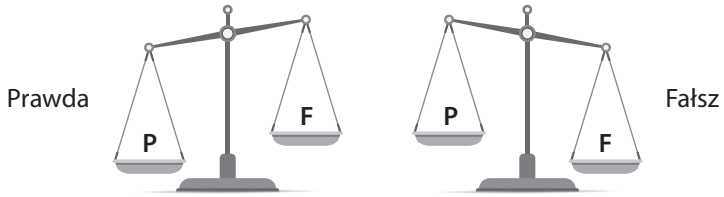
Jak nietrudno się domyślić, rezultatem działania tego programu będzie wyświetlenie liczby `2.5` w konsoli. Spróbujmy nieco tutaj namieszać i zamieńmy typ zmiennej `x` na `float`.

Listing 2.5. Dodawanie liczb zmiennoprzecinkowych o różnych typach

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

    float x = 1.2f; // litera f umieszczona po liczbie
    double y = 1.3;
    System.out.println(x + y); // dodanie zmiennej typu double
                               // do zmiennej typu float
}
```

Zwróć uwagę na to, że aby przypisać liczbę do zmiennej typu `float`, należy posłużyć się literą „f" lub „F". Dzieje się tak dlatego, iż domyślnie w Javie liczba zmiennoprzecinkowa jest traktowana jako `double`. W tym miejscu pojawia się pewna ciekawostka. Mianowicie podczas dodawania do siebie `float` i `double` (lub wykonywania innej operacji arytmetycznej) dochodzi do konwersji, która sprawia, że wynik jest niedokładny. Ważne, żeby pamiętać, że przy pracy ze zmiennymi typu `float` i `double` może dojść do niedokładnych wyników. Dlatego w praktyce przy dokładnych obliczeniach stosuje się typ `BigDecimal`. Poznasz go w jednym z kolejnych rozdziałów tej książki. Jeśli potrzebujesz skorzystać ze zmiennej typu `float` lub `double`, sugeruję raczej użycie typu `double` ze względu na większą pojemność i precyzję.

Chciałbym przedstawić Ci jeszcze jeden bardzo istotny typ prosty, jakim jest `boolean`. To bardzo ciekawy typ, gdyż może przyjmować tylko dwie wartości: `true` (prawda) i `false` (fałsz). Można wyobrazić go sobie jako wagę, na której umieszczone są prawda (P) i fałsz (F). W zależności od tego, co jej przekażemy, przechyli się ona w danym kierunku. Obrazuje to następujący rysunek:



Rysunek 2.3. Typ boolean

Brzmi banalnie, ale nie jest to takie oczywiste. Na przykład w kodzie wyrażenie `1 > 0` także ma wartość `true`. Spójrz na listing 2.6.

Listing 2.6. Przykładowe zastosowanie zmiennej typu boolean

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

    boolean domyslnie; // jeśli nic do niej nie przypiszesz, domyślnie będzie miała
                       // wartość false
    boolean wartosc = true; // przypisanie wartości true
    boolean czyJedenWiekszeOdDwoch = 1 > 2; // fałsz, ponieważ 1 jest
                                            // mniejsze od 2
    System.out.println(wartosc); // prawda
    System.out.println(czyJedenWiekszeOdDwoch); // fałsz
}
```

Jak widzisz, typ `boolean` może być pomocny nie tylko w przechowywaniu oczywistej wartości prawda/fałsz, ale także w sprawdzaniu danego wyrażenia. Takim wyrażeniem może być zarówno warunek logiczny, jak i np. porównanie dwóch fragmentów tekstu i ocena, czy są one takie same. Operacje tego typu poznasz jednak nieco później — idźmy spokojnie do przodu, krok po kroku. W tej chwili skup się na ogólnych zastosowaniach poszczególnych typów, a w kolejnych rozdziałach poznasz ich rozszerzone możliwości. Proces nauki programowania wymaga pewnych etapów i otwierania kolejnych skrzyń okrytych tajemnicą.

Dokonajmy jeszcze kilku deklaracji typów prostych, aby oswoić się z ich używaniem.

Listing 2.7. Deklaracja wszystkich typów prostych

```
char znak = "c";
boolean prawda = true;
boolean falsz = false;
boolean czyJedenWiekszeOdDwoch = 1 > 2;
double liczbaNiecalkowita = 1.3;
byte malaLiczba = 3;
short niecoWieksza = 32123;
int duzaLiczba = 434234234;
long wielkaLiczba = 2323213213121;
float mniejszaZmiennoprzecinkowa = 312.32f;
double wiekszaZmiennoprzecinkowa = 321312.4342342;
```

Tabela 2.1 przedstawia typy dostępne w Javie. Zwróć uwagę na to, że typy proste są pisane małą literą.

Tabela 2.1. Typy proste w Javie

Typ	Co reprezentuje	Możliwe wartości
`boolean`	Prawda/fałsz	`true` (prawda), `false` (fałsz)
`char`	Pojedynczy znak, np. litera	Wartości pojedynczych znaków otoczone apostrofami, np. `'a'`, `'b'`
`byte`	Bajt	Jeden bajt (w praktyce: liczba od –128 do 127)
`short`	Niewielka liczba całkowita	Liczba całkowita z przedziału: –32 768 – 32 767
`int`	Liczba całkowita średniej długości	Liczba całkowita z przedziału: –2 147 483 648 – 2 147 483 647
`long`	Długa liczba całkowita	Liczba całkowita z przedziału: –9 223 372 036 854 775 808 – 9 223 372 036 854 775 807
`float`	Średniej długości liczba zmiennoprzecinkowa (z możliwymi wartościami po przecinku)	Liczba zmiennoprzecinkowa z przedziału: $3.40282347 \times 10^{38}$ – $1.40239846 \times 10^{-45}$
`double`	Długa liczba zmiennoprzecinkowa	Liczba zmiennoprzecinkowa z przedziału: $1.7976931348623157 \times 10^{308}$ – $4.9406564584124654 \times 10^{-324}$

Pomocnik wykonujący czynności — metoda

Załóżmy teraz, że najpierw mnożysz liczby, a potem chcesz je dodatkowo zsumować z dwoma innymi. Zgodnie z tym, z czym zapoznałeś się do tej pory, kod wyglądałby mniej więcej tak:

Listing 2.8. Dodawanie wielu liczb

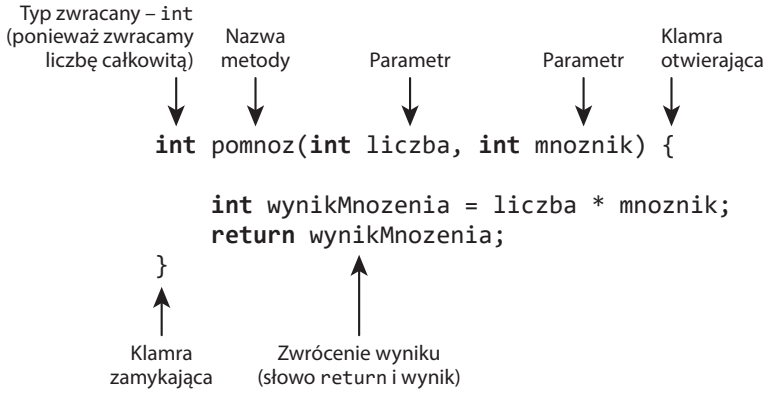
```
public static void main(String[] args) {

    int liczba = 5;
    int mnoznik = 2;
    int wynikMnozenia = liczba * mnoznik;
    int pierwszaDoDodania = 2;
    int drugaDoDodania = 3;
    int suma = wynikMnozenia + pierwszaDoDodania + drugaDoDodania;
    System.out.println(suma);
}
```

Powyższy kod wygląda z pozoru normalnie, ale czy jest czytelny? Co jeśli operacji tych byłoby o wiele, wiele więcej? Z pomocą przychodzą tak zwane metody. Metoda to coś w rodzaju naszego pomocnika, który ma określone zadanie. Tak jak prawdziwy pomocnik, np. stolarza, może on dostać do obróbki kawałek drewna (w kodzie byłby to parametr metody), a rezultatem jego pracy może być choćby drewniana figurka (w kodzie byłaby to wartość zwracana przez metodę). Poniżej znajdziesz ilustrację obrazującą poszczególne elementy metody.


```
int pomnoz(int liczba, int mnoznik) {

    int wynikMnozenia = liczba * mnoznik;
    return wynikMnozenia;
}
```

Rysunek 2.4. Porównanie metody z prawdziwym pracownikiem

Poszczególne elementy metody wyglądają następująco:



Rysunek 2.5. Składnia metody

A zatem przy deklaracji metody musimy wskazać zwracany przez nią typ zmiennej, nazwę metody oraz parametry. Dalej znajduje się jej ciało w nawiasach klamrowych, a wewnątrz nich musimy zwrócić jakąś wartość zwracanego typu, używając słowa `return`. Słowo to oznacza także koniec wykonania metody, a zatem nie możemy po nim wykonywać żadnych innych operacji. Niedopuszczalna jest taka konstrukcja:

Listing 2.9. Instrukcja wywoływana po słowie return

```
int metoda() {

    return 0; // koniec metody
    System.out.println("Test"); // błąd
}
```

Istnieje pewne — ale tylko pozorne — odstępstwo od tej reguły, o którym dowiesz się więcej w kolejnym podrozdziale, dotyczącym instrukcji warunkowych `if` i `switch`. Metoda może również nie przyjmować żadnych parametrów i niczego nie zwracać (wtedy jako typ zwracany podajemy `void` oraz nie musimy używać `return`). Przykład takiej metody znajdziesz poniżej.

Listing 2.10. Metoda, która niczego nie zwraca

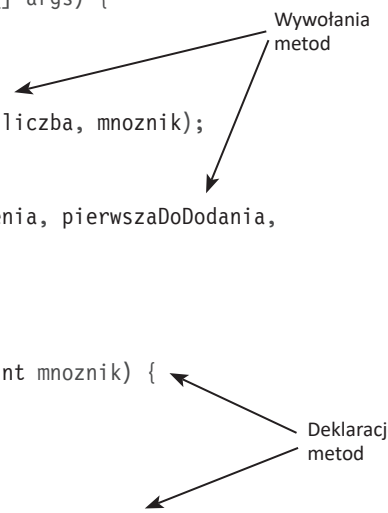
```
void wyswietlMilyTekst() {

    System.out.println("Lubię Cię! :)");
}
```

Takich metod używamy zazwyczaj wtedy, gdy musimy wykonać pewną odseparowaną czynność, która nie jest sprzężona z innymi. Przykładowo, jeśli chcemy napisać metodę wyświetlającą okienko, gdy użytkownik naciśnie przycisk, to wystarczy zaimplementować metodę, która niczego nie zwraca (zwraca `void`), a ta po prostu wyświetli okienko, czyli zrobi to, co miała zrobić, i rezultat jej działania nie będzie nikomu dalej potrzebny. Metody zwracające jakiś wynik zazwyczaj biorą udział w jakimś większym przedsięwzięciu. Ich rezultat często jest poddawany dalszej obróbce przez inne powiązane metody, co tworzy coś w rodzaju sieci.

Metodę wywołujemy, podając jej nazwę, a w nawiasach okrągłych — wartości, które przekazujemy do jej parametrów. Wartość zwracaną przez metody możemy przypisać do zmiennej. Cały kod z listingu 2.8 wzbogacony o metody służące dodawaniu i mnożeniu liczb wyglądałby następująco:

Listing 2.11. Przykłady metod

```
public class Main {

    public static void main(String[] args) {

        int liczba = 5;
        int mnoznik = 2;
        int wynikMnozenia = pomnoz(liczba, mnoznik);
        int pierwszaDoDodania = 2;
        int drugaDoDodania = 3;
        int suma = sumuj(wynikMnozenia, pierwszaDoDodania,
drugaDoDodania);
        System.out.println(suma);
    }

    static int pomnoz(int liczba, int mnoznik) {

        return liczba * mnoznik;
    }

    static int sumuj(int liczba, int liczbaDoDodania,
int kolejnaDoDodania) {

        return liczba + liczbaDoDodania + kolejnaDoDodania;
    }
}
```



Jak możesz zauważyć, metody `pomnoz()` i `sumuj()` zostały umieszczone poza metodą `main()`, którą poznałeś wcześniej jako wyznacznik początku i końca wykonywania programu. Zapytasz pewnie: „Ale jak to? Przecież miała ona stanowić początek i koniec, więc jakim cudem coś znajduje się poza jej ramami?". Już wyjaśniam: deklaracje metod znajdują się poza metodą `main()`, ale są wewnątrz niej wywoływane, czyli zasada początku i końca działania programu zostaje zachowana. Ważne, aby pamiętać, że metody muszą być umieszczone wewnątrz klasy (w tym wypadku: `class Main`). Nie będziemy teraz zagłębiać się w tę tematykę — klasy poznasz później, podobnie jak słowo kluczowe `static` (umożliwia ono wywołanie metody w metodzie `main()`, dlatego jest konieczne). Wiem, wiem — pojawiło się już sporo nowych elementów, które musisz pominąć. Nie chcę Ci jednak robić mętliku w głowie, dlatego idźmy stopniowo do przodu. Obecnie najważniejsze, co musisz zapamiętać, to struktura metody, miejsce, w którym może być ona zadeklarowana (poza inną metodą, wewnątrz klasy), oraz sposób jej wywołania.

Zadanie 2.1.

Napisz metodę o nazwie `sprawdz`, która przyjmie jako parametry dwie liczby całkowite (*x* i *y*) i zwróci wartość `true`, jeśli *x* jest większy od *y*, a w przeciwnym wypadku zwróci wartość `false`. Wyświetl w konsoli wynik dla liczb *x* = 2, *y* = 1 oraz *x* = 1, *y* = 2.

Wskazówka: pamiętaj o słowie `static` przed typem zwracanym, żebyś mógł użyć tej metody w metodzie `main()`.

Jeśli tak, to zrób tak — instrukcje warunkowe if i switch-case

Nadszedł czas, aby poznać kolejne możliwości, jakie niesie za sobą pisanie programów komputerowych. Na przykład w przypadku aplikacji do przelewów bankowych użytkownik nie powinien mieć możliwości wykonania przelewu na kwotę wyższą niż ta, którą posiada na koncie — a zatem Ty, jako programista, musisz być w stanie to sprawdzić. Z pomocą przychodzi instrukcja warunkowa `if`. Jej konstrukcja jest bardzo prosta:

Listing 2.12. Składnia instrukcji if

```
if(warunek) {
    // wykonaj, jeśli warunek został spełniony
}
```

Warunek musi odpowiadać typowi `boolean`, czyli stanowić prawdę lub fałsz (`true` lub `false`). W instrukcji `if` nie możemy podać litery ani cyfry — to nie miałoby sensu, więc przy takiej próbie środowisko zwróciłoby Ci błąd.

Napiszmy zatem prosty program, który „wykona przelew" i sprawdzi, czy jest on w ogóle możliwy. Można to zrobić w następujący sposób (celowo pominąłem metodę `main()` w celu skrócenia kodu i skupienia się na konkretnym fragmencie):

Listing 2.13. Program sprawdzający możliwość wykonania przelewu

```
int stanKonta = 2000;
int kwotaPrzelewu = 100;
if(stanKonta > kwotaPrzelewu) {

    stanKonta = stanKonta - kwotaPrzelewu; // odjęcie kwoty przelewu
    System.out.println("Wykonano przelew");
}
```

Program wydaje się dość prosty: deklarujemy stan konta i kwotę przelewu, a następnie sprawdzamy, czy stan konta jest większy. Jeśli tak, to przelew zostaje wykonany, a jeśli nie... No właśnie: czegoś tu brakuje, prawda? Co jeśli ilość środków na koncie jest mniejsza od kwoty przelewu? W tym przypadku

program nie przedstawi żadnej informacji na ten temat — po prostu pominie czynność zawartą w instrukcji `if`. Aby obsłużyć przeciwny przypadek, należy posłużyć się słowem `else`.

Listing 2.14. Użycie konstrukcji if-else

```
int stanKonta = 2000;
int kwotaPrzelewu = 100;
if(stanKonta > kwotaPrzelewu) {

    stanKonta = stanKonta - kwotaPrzelewu; // odjęcie kwoty przelewu
    System.out.println("Wykonano przelew");
} else {
    System.out.println("Błąd - za mało środków na koncie");
}
```

Oczywiście istnieje możliwość konfigurowania wielu przypadków. Przykładowo, jeśli chcielibyśmy wyświetlić komunikat, w sytuacji gdy kwota przelewu byłaby identyczna jak kwota znajdująca się na koncie, musielibyśmy posłużyć się konstrukcją `else-if`.

Listing 2.15. Użycie konstrukcji else-if

```
int stanKonta = 2000;
int kwotaPrzelewu = 100;
if(stanKonta > kwotaPrzelewu) {

    stanKonta = stanKonta - kwotaPrzelewu; // odjęcie kwoty przelewu
  System.out.println("Wykonano przelew");
} else if(stanKonta == kwotaPrzelewu) {

    System.out.println("Kwota przelewu jest równa ilości środków
na koncie");
} else {
    System.out.println("Nie można wykonać przelewu - za mało środków
na koncie");
}
```

Do porównywania wartości (sprawdzania, czy są one sobie równe) używamy podwójnego znaku równości (==). Pojedynczy znak równości (=) jest zarezerwowany na potrzeby przypisywania wartości do zmiennej (z czym już się spotkałeś). Jeżeli chcemy sprawdzić, czy dane wartości nie są sobie równe, używamy do tego operatora `!=` (znak `!` jest zaprzeczeniem).

Listing 2.16. Zaprzeczenie w instrukcji if

```
if(stanKonta != kwotaPrzelewu) // czy stan konta jest różny od kwoty przelewu?
```

W tym miejscu może pojawić się pytanie: „Czy istnieje możliwość łączenia kilku warunków w jeden, np. w celu sprawdzenia, czy ilość środków na koncie jest większa od kwoty przelewu *i* czy użytkownik należy do programu

promocyjnego?". Tak, jest to możliwe dzięki operatorowi &&. W praktyce oznacza on to samo co „i" w języku polskim. Instrukcja `if` dokonująca takiego sprawdzenia wyglądałaby następująco:

Listing 2.17. Użycie operatora && („i")

```
int stanKonta = 2000;
int kwotaPrzelewu = 100;
boolean nalezyDoPromocji = true;
if(stanKonta > kwotaPrzelewu && nalezyDoPromocji) {

    stanKonta = stanKonta - kwotaPrzelewu; // odjęcie kwoty przelewu
  System.out.println("Wykonano przelew");
    stanKonta = stanKonta + 2;
    System.out.println("Przyznano także środki w ramach promocji");
}
```

Użycie operatora &&, czyli „i"

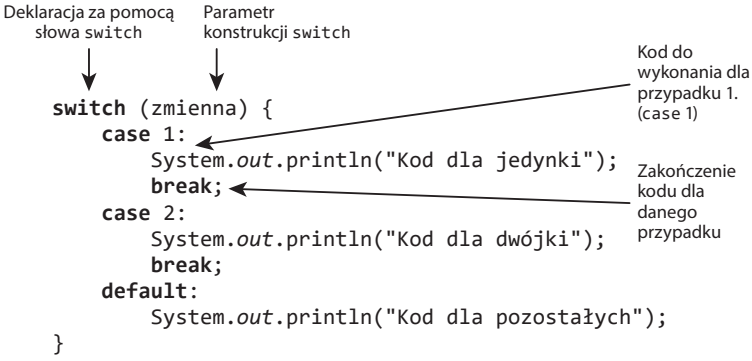
Istnieje jeszcze jeden podobny operator: `||`, który oznacza „lub". Sprawdźmy zatem, czy ilość środków na koncie użytkownika jest większa od kwoty przelewu *lub* czy ilość środków na drugim koncie jest większa od kwoty przelewu (kod ma charakter poglądowy).

Listing 2.18. Użycie operatora || („lub")

```
int stanKonta = 2000;
int kwotaPrzelewu = 2500;

int stanDrugiegoKonta = 5000;
if(stanKonta > kwotaPrzelewu || stanDrugiegoKonta > kwotaPrzelewu) {

    stanKonta = stanKonta - kwotaPrzelewu; // odjęcie kwoty przelewu
    System.out.println("Wykonano przelew");
}
```

Użycie operatora `||`, czyli „lub"

Alternatywą dla instrukcji `if` jest konstrukcja `switch`. Ma ona następującą strukturę:



Rysunek 2.6. Składnia przełącznika switch

Najprościej będzie to zobrazować na przykładzie. Załóżmy, że chcemy napisać aplikację wyświetlającą nazwę dnia tygodnia w zależności od numeru, który zostanie podany, czyli jedynka to poniedziałek, dwójka — wtorek itd. Zrobimy to za pomocą przełącznika `switch`. Jego wartością będzie numer dnia tygodnia.

Listing 2.19. Deklaracja zmiennej

```
int dzienTygodnia = 1;
```

Następnie przekazujemy tę wartość do przełącznika `switch` i otwieramy klamrę.

Listing 2.20. Deklaracja przełącznika switch

```
switch (dzienTygodnia) {
```

Używając słowa `case`, przekazujemy kolejne wartości i kod, który powinien zostać wykonany w przypadku ich wystąpienia.

Listing 2.21. Przypadki przełącznika switch

```
case 1: // wartość zmiennej z przełącznika
System.out.println("Poniedziałek"); // kod do wykonania
break; // zakończenie
case 2:
System.out.println("Wtorek");
break;
case 3:
System.out.println("Środa");
break;
```

Na końcu możemy umieścić fragment kodu, który ma być wykonany dla pozostałych wartości zmiennej przełącznika. W tym celu użyjemy słowa `default`, zamykając konstrukcję `switch`.

Listing 2.22. Domyślny przypadek przełącznika switch

```
default:
System.out.println("Nieznany dzień tygodnia");
```

Całość kodu będzie prezentowała się tak jak na listingu 2.23.

Listing 2.23. Instrukcja switch określająca dzień tygodnia

```
int dzienTygodnia = 1; // zmienna, której wartość steruje przełącznikiem switch
switch (dzienTygodnia) {
    case 1: // jeżeli 1
        System.out.println("Poniedziałek"); // kod do wykonania dotyczący
                                            // przypadku 1.
        break; // zakończenie wykonania kodu dotyczącego przypadku 1.
    case 2: // jeżeli 2
        System.out.println("Wtorek"); // kod do wykonania dotyczący przypadku 2.
        break; // zakończenie wykonania kodu dotyczącego przypadku 2.
    case 3: // jeżeli 3
        System.out.println("Środa"); // kod do wykonania dotyczący przypadku 3.
```

```
            break; // zakończenie wykonania kodu dotyczącego przypadku 3.
        default: // jeżeli żadne z powyższych
            System.out.println("Niezdefiniowany dzień"); // domyślny kod
                                                         // do wykonania
}
```

W podrozdziale dotyczącym metod wspomniałem o pewnym odstępstwie od reguły, polegającym na tym, że w metodzie po słowie `return` nie możemy zawierać żadnych instrukcji. W przypadku instrukcji warunkowych takich jak `if` lub `switch` możemy umieścić `return` na końcu kodu danego warunku.

Listing 2.24. Kilka słów return w metodzie używającej przełącznika switch

```
char podajPierwszaLitereDniaTygodnia() {

    int dzienTygodnia = 1;
    switch (dzienTygodnia) {
        case 1:
            return 'p'; // nie trzeba dodawać słowa break, bo wychodzimy
                        // z przypadku za pomocą słowa return
        case 2:
            return 'w';
        case 3:
            return 's';
        default:
            return 'x';
    }
}
```

Listing 2.25. Kilka słów return w metodzie używającej instrukcji if

```
char podajPierwszaLitereDniaTygodnia() {

    int dzienTygodnia = 1;
    if (dzienTygodnia == 1) {
        return 'p';
    } else if (dzienTygodnia == 2) {
        return 'w';
    } else if (dzienTygodnia == 3) {
        return 's';
    } else {
        return 'x';
    }
}
```

Możemy podać kilka słów `return` w jednej metodzie, gdyż i tak wykonany zostanie tylko jeden z nich. Program pójdzie jedną ze ścieżek wymienionych w instrukcjach `if` lub `switch`, a zatem `return` ostatecznie będzie użyty *raz, na końcu metody*.

Zadanie 2.2.

Napisz program, który przyjmie jedną z trzech pierwszych liter alfabetu (a, b lub c) oraz wyświetli liczbę określającą jej pozycję w alfabecie (1, 2 lub 3), a dla pozostałych liter zwróci 0. Utwórz dwie osobne metody: z instrukcją `switch` oraz `if`, zwracające odpowiednią liczbę według podanego parametru.

Powtórz kod wielokrotnie — pętla for

Pętla umożliwia nam wielokrotne powtórzenie danego fragmentu kodu. Trudno sobie wyobrazić jej odzwierciedlenie w świecie rzeczywistym, gdyż jest to mechanizm sterujący, ale można go porównać do osiołka goniącego za marchewką. Dopóki osiołek nie dostanie marchewki, dopóty cały czas pracuje i próbuje ją dogonić i zjeść. Tak samo działa pętla, co obrazuje poniższa ilustracja:


```
for (int zmienna = 1; zmienna <= 100; zmienna++) {

    System.out.println(zmienna);
}
```

Rysunek 2.7. Analogia między pętlą a światem rzeczywistym

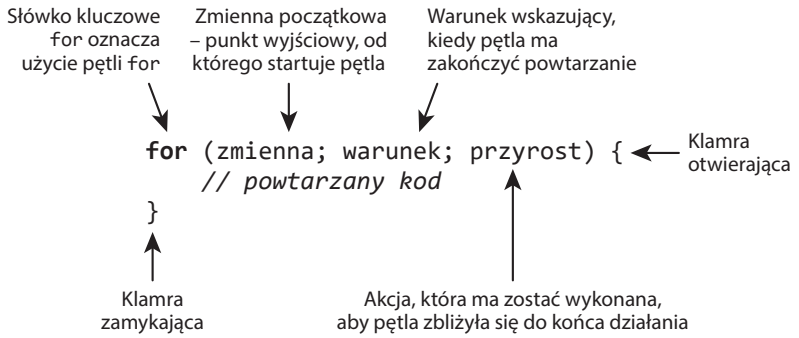
Rozważmy pewien problem dotyczący kodu programu. Gdybyśmy chcieli wyświetlić liczby od 1 do 100, kod musiałby wyglądać mniej więcej tak:

Listing 2.26. Poglądowy kod służący do wyświetlenia 100 liczb bez użycia pętli

```
public static void main(String[] args) {
    System.out.println(1);
    System.out.println(2);
    System.out.println(3);
    System.out.println(4);
    // i tak dalej aż do 100...
}
```

Pisanie 100 linijek kodu byłoby dość uciążliwe, prawda? A co w przypadku, jeśli mielibyśmy wyświetlić tyle liczb, ile wskaże użytkownik? Przykładowo, jeśli poda wartość 50, wyświetlimy tylko 50 liczb od 1 do 50 i tak dalej. Z pomocą przychodzą nam pętle. Nie chodzi tutaj o pętlę sznurka, tylko raczej o rodzaj zapętlonego mechanizmu. W języku Java istnieją trzy rodzaje pętli: `for`, `while` i `do-while`.

Pętla `for` ma następującą składnię:



Rysunek 2.8. Składnia pętli for

Z pewnością zdążyłeś już się wystraszyć liczby elementów występujących w pętli `for`. Spokojnie, omówimy tę kwestię dokładnie, aby nie było żadnych wątpliwości. Wróćmy na chwilę do naszego przykładu z wyświetlaniem liczb od 1 do 100 i spróbujmy zdefiniować kluczowe elementy sterowania pętlą.

- **Zmienna:** `int zmienna = 1`
- **Warunek:** `zmienna <= 100`
- **Przyrost:** `zmienna++`

Ponieważ zaczynamy od 1, zmienną początkową będzie właśnie 1. Kończymy na liczbie 100, a zatem warunek wskazujący, kiedy pętla ma się zakończyć, to `zmienna <= 100` (wartość zmiennej mniejsza lub równa 100). Przyrost, czyli to, co się dzieje po wykonaniu kodu wewnątrz pętli, stanowi wyrażenie `zmienna++`, które w praktyce oznacza to samo co `zmienna = zmienna + 1`. Analogicznie stosowany jest minus (możemy napisać: `zmienna--`). Możemy również używać np. zapisu `zmienna+= 2`, który oznacza `zmienna = zmienna + 2`.

Listing 2.27. Przyrost liczbowy z użyciem skróconego zapisu

```
zmienna++; // dodaj 1 do zmiennej
zmienna--; // odejmij 1 od zmiennej

zmienna+= 2; // dodaj 2 do zmiennej
zmienna-= 2; // odejmij 2 od zmiennej
zmienna*= 2; // pomnóż zmienną przez 2
zmienna/= 2; // podziel zmienną przez 2
```

Oczywiście nie ma konieczności stosowania takiego zapisu, niemniej jest on sporo krótszy i przy odrobinie wprawy można go łatwo odczytać.

Zatem po podstawieniu pod strukturę pętli `for` danych, których potrzebujemy do wyświetlenia liczb od 1 do 100, otrzymamy kod z listingu 2.28.

Listing 2.28. Wyświetlenie liczb od 1 do 100 z użyciem pętli for

```
for (int zmienna = 1; zmienna <= 100; zmienna++) {

    System.out.println(zmienna);
}
```

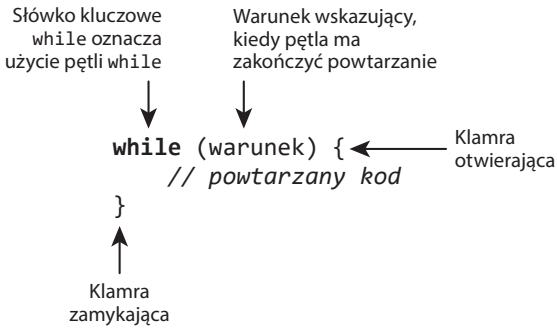
Pętla musi być umieszczona wewnątrz metody — jest ona konkretną instrukcją, a zatem nie może się znajdować poza metodą. Aby wyświetlić co drugą liczbę od 1 do 100, wystarczy zwiększyć przyrost z 1 do 2.

Listing 2.29. Wyświetlenie co drugiej liczby od 1 do 100

```
for (int zmienna = 1; zmienna <= 100; zmienna+=2) {
    System.out.println(zmienna);
}
```

Pętla while i do-while

Mam nadzieję, że udało Ci się zrozumieć koncept pętli. Poznajmy zatem jej drugi typ, czyli pętlę `while`. Ma ona o wiele prostszą konstrukcję, widoczną poniżej.



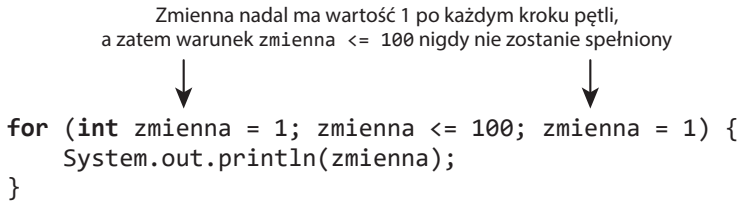
Rysunek 2.9. Składnia pętli while

Widać tutaj tylko warunek zakończenia pętli. Nasuwa się pewna myśl: czy przez podanie warunku, który zawsze będzie prawdziwy, uzyskamy efekt pętli wywoływanej bez końca? Dokładnie tak — wtedy nasz „osiołek" będzie gonił „marchewkę" w nieskończoność. Poniżej znajduje się przykład takiej pętli.

Listing 2.30. Nieskończona pętla while

```
while(1 == 1) {
    // powtarzany kod
}
```

W ramach eksperymentu możesz spróbować uruchomić program z taką pętlą, jednak nie zdziw się, gdy Twój komputer zacznie się zacinać. Wydarzy się tak, ponieważ będzie ją powtarzał w nieskończoność w nadziei na jej zakończenie. Dlatego tak ważne jest zapewnienie zakończenia działania pętli. Również pętla `for` może działać w nieskończoność.



Rysunek 2.10. Nieskończona pętla for

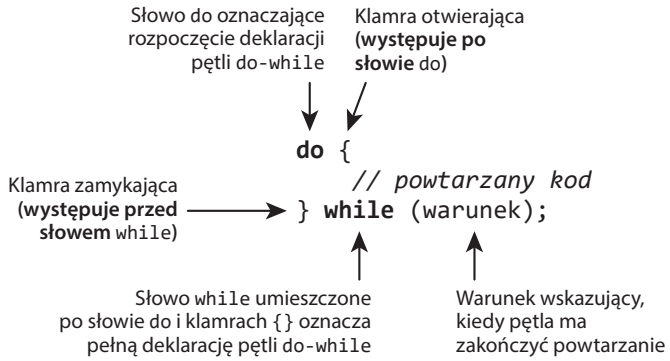
No dobrze, zastanówmy się teraz przez chwilę, jak skonstruować warunek i zapewnić jego spełnienie w pętli `while`. Posłużmy się tym samym przykładem, czyli wyświetleniem liczb od 1 do 100.

Listing 2.31. Wyświetlenie liczb od 1 do 100 w pętli while

```
int zmienna = 1; // zmienna początkowa
while (zmienna <= 100) {

    System.out.println(zmienna);
    zmienna++; // przyrost
}
```

Dostrzegasz tutaj zapewne analogię do pętli `for`. Sposób zapisu nieco się jednak różni.

Przejdźmy do omówienia ostatniego typu (jak wynika z mojego doświadczenia — najrzadziej wykorzystywanego), czyli pętli `do-while`.



Rysunek 2.11. Pętla do-while

Różnica między pętlą `while` i `do-while` jest nieznaczna. W pętli `do-while` *najpierw wykonuje się powtarzany kod, a później sprawdza się warunek*. A zatem mamy tutaj dwa podejścia:

1. Pętla `while` najpierw sprawdza warunek, a później ewentualnie wykonuje powtarzany kod (jeśli warunek nie jest spełniony).
2. Pętla `do-while` najpierw wykonuje powtarzany kod, a później sprawdza warunek i ewentualnie znowu wykonuje powtarzany kod. Zatem powtarzany *kod z pętli `do-while` zostanie wykonany co najmniej raz*.

Zadanie 2.3.

Napisz program zawierający trzy metody wyświetlające liczby mniejsze od 20, ale większe od liczby całkowitej podanej jako parametr metody. Użyj pętli `for`, `while` i `do-while` (po jednej na każdą metodę).

Wskazówka: pamiętaj, że pętla `do-while` jest wykonywana przynajmniej raz. Nie wyświetl przez pomyłkę liczby, która nie spełnia kryterium z zadania.

Szybka powtórka

Zmienna — pojemnik na dane określonego typu. Przykładowe typy zmiennych to:

- `int` — liczby całkowite;
- `double` — liczby zmiennoprzecinkowe;
- `char` — pojedynczy znak;
- `boolean` — wartość logiczna `true` lub `false`.

Zmienną deklarujemy, podając jej typ i nazwę (pisaną małą literą, kolejne wyrazy — dużą), oraz opcjonalnie przypisujemy jej wartość.

Listing 2.32. Deklaracja przykładowej zmiennej

```
int przykladowaZmienna = 2;
```

Metoda — umożliwia wydzielenie fragmentu kodu i pewnej czynności, za którą odpowiada. Zawiera: typ zwracany, parametry (opcjonalnie), nazwę (nazewnictwo jest takie samo jak w przypadku zmiennych). Typ zwracany metody, która niczego nie zwraca, określa się jako `void`. Metody zwracające jakąś wartość muszą zawierać na końcu słowo `return`. Poniżej przykłady metod — jedna niczego nie zwraca, druga zwraca rezultat.

Listing 2.33. Przykładowe deklaracje metod

```
void metoda(int parametr) {
    System.out.println(1);
}

int metodaZwracajaca(int parametr) {
    System.out.println("Zwracam liczbę większą o 1 od parametru");
    return parametr + 1;
}
```

Instrukcje warunkowe `if` i `switch` — umożliwiają sterowanie działaniem programu i sprawdzanie różnych warunków oraz przypadków.

Listing 2.34. Przykładowa instrukcja if

```
int zmienna = 1;
if (zmienna > 0) {
    // wykonaj, jeśli zmienna jest większa od 0
} else if (zmienna == 0) {
    // wykonaj, jeśli zmienna jest równa 0
} else {
    // wykonaj dla pozostałych przypadków
}
```

Listing 2.35. Przykładowe użycie instrukcji switch-case

```
int zmienna = 1;
// instrukcja switch
switch (zmienna) {
    case 0:
        // kod dla 0
        break;
    case 1:
        // kod dla 1
        break;
    default:
        // kod dla innych
}
```

Warunki możemy dodatkowo łączyć i negować.

Listing 2.36. Przykładowe łączenie i negowanie warunków

```
boolean warunekAnd = zmienna > 1 && zmienna < 10; // większa od 1
                                                  // i mniejsza od 10
boolean warunekOr = zmienna > 1 || zmienna < 0; // większa od 1
                                                // lub mniejsza od 0
boolean warunekNor = !(zmienna > 3); // zmienna nie większa od 3
```

Pętle — umożliwiają wielokrotne wykonanie kodu zgodnie z podanym warunkiem wyjścia z pętli. W Javie wyróżniamy pętle `for`, `while` i `do-while`. Pętla `do-while` jest wykonywana zawsze minimum raz.

Listing 2.37. Przykładowe deklaracje pętli

```
// pętla for - wyświetl liczby od 0 do 19
for (int i = 0; i < 20; i++) {
    System.out.println(i);
}
// pętla while - wyświetl liczby od 0 do 19
int i = 0;
while (i < 20) {
    System.out.println(i);
    i++;
}
// pętla do-while - wyświetl liczby od 0 do 19
int x = 0;
do {
    System.out.println(x);
    x++;
} while (x < 20);
```

Java od lat należy do najpopularniejszych backendowych języków programowania — i do najbardziej rozpowszechnionych języków programowania w ogóle. Zdecydowana większość aplikacji webowych obecnych na rynku konsumenckim i w biznesie powstaje z jej użyciem. Java może się więc okazać świetnym wyborem na początek przygody z programowaniem, a także wtedy, gdy jej poznanie ma być pierwszym krokiem na drodze do zmiany zawodowej ścieżki.

Java. Podręcznik na start to pozycja idealna dla każdego, kto chce podjąć taki krok. W przystępny sposób zaznajamia nie tylko z językiem, ale też z podstawowymi koncepcjami stosowanymi podczas programowania — w obrazowy sposób przyrównuje je do sytuacji znanych spoza świata wirtualnego. Autor nie poprzestaje na podstawach i objaśnia także bardziej zaawansowane zagadnienia, dokonuje również przeglądu zmian, jakie w ciągu poprzednich lat zaszły w kolejnych wersjach Javy. Ostatnie rozdziały poświęca bardziej złożonym aspektom korzystania z tego języka programowania, w tym funkcjonowaniu mechanizmów odśmiecania pamięci, maszynie wirtualnej Javy i jej kompilatorom.

Dzięki książce:

- **opanujesz podstawy programowania**
- **gruntownie zapoznasz się z językiem Java**
- **zgłębisz obsługę bibliotek**
- **nauczysz się przeprowadzać testy**
- **przyswoisz zasady pisania czystego kodu**

Krzysztof Krocz

Absolwent studiów magisterskich i inżynierskich na Politechnice Lubelskiej, na kierunku informatyka. Od ośmiu lat działa aktywnie w obszarach związanych z wytwarzaniem oprogramowania i przekazywaniem wiedzy za pośrednictwem kanału YouTube (*https://www.youtube.com/@JavaSolutions*). Na co dzień pracuje jako lider techniczny i architekt rozwiązań, w pracy stawia na jakość, prostotę i optymalizację projektowanych rozwiązań. Chętnie dzieli się wiedzą; uważa, że należy ją przekazywać w możliwie prosty sposób i powoływać się przy tym na oczywiste przykłady.

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 978-83-283-9783-5

9 788328 397835

Cena: 69,00 zł