Instrukcje warunkowe

Spis treści

Zapiski uczestnika Bootcampu Zajavka.pl w 12 tygodni by Bartek Borowczyk aka Samuraj Programowania. Dopiski na zielono od Karola Rogowskiego.

Instrukcje warunkowe if

Na tym etapie przechodzimy do sterowania programem, do którego służą **instrukcje warunkowe** i **pętle**. Czyli:

- zrób coś, jeśli (instrukcje warunkowe) - tutaj poznamy instrukcję warunkową if
- zrób coś wiele razy (pętle) - poznamy pętlę for

Bardzo często zamiast instrukcji sterujących, szczególnie pętli, wykorzystujemy funkcje, ale to jest kolejny etap rozwoju programisty. Na pewno, by programować, trzeba dobrze poznać koncepcje instrukcji warunkowych i pętli - są to fundamenty każdego współczesnego języka programowania.

Instrukcja if

Podstawowa składnia **instrukcji warunkowej if**:

```
if(warunek) {
    // blok kodu, w którym umieszczamy instrukcje do wykonania w przypadku, gdy warunek zwróci true
}

if (2 > 1) {
    System.out.println("szedłem do ciała instrukcji warunkowej");
}
```

W przykładzie powyżej warunek będzie zawsze prawdziwy, ale w programie taka sytuacja oczywiście

się nie zdarza, bo zawsze porównujemy wartości, które mogą ulec zmianie, np.

```
if(hasloWpisanePrzezUzytkownika != hasloKtorePrzechowujeBazaDanych) {
    wydrukujInformacje("nieprawidłowe hasło");
}
```

Pamiętaj, że != i == nie jest operatorem właściwym do porównania stringów ☺. W tym wypadku w Javie należałoby użyć metody equals(), np.

```
if(hasloWpisanePrzezUzytkownika.equals(hasloKtorePrzechowujeBazaDanych)) {
    wydrukujInformacje("prawidłowe hasło");
}
```

Na tym etapie zwróćmy uwagę, że instrukcja warunkowa tworzy **blok**, a więc i zakres lokalny (zakres blokowy) dla zmiennych. Zapewne pamiętasz, że wszystko, co ujęte w klamry {}, określamy blokiem. Zmienna zadeklarowana w danym bloku jest widoczna tylko w tym bloku i w blokach zagnieżdżonych, nie jest zaś widoczna w blokach nadrzędnych. Pamiętaj też, że nie można stworzyć zmiennej o danej nazwie, jeśli w bloku nadrzędnym (blokach nadrzędnych) istnieje już zmienna o takiej nazwie.

Na poniższym pseudokodzie widać też przykłady zagnieżdżenia bloków, blok w bloku to nic dziwnego i oczywiście, nie myślisz o tym, programując, bo takie rzeczy od razu widzisz na późniejszym etapie, np.

```
metoda() {
    // blok (w tym wypadku ciało metody) - tworzy swój zasięg
    instrukcja-warunkowa() {
        // blok (ciało instrukcji warunkowej) - tworzy swój zasięg
        pętla() {
            // blok (ciało pętli) tworzy swój zasięg
        }
    }
}
```

Wróćmy do składni instrukcji **warunkowej if**. W pewnych sytuacjach możemy zrezygnować z nawiasów klamrowych, ale Karol wielokrotnie odradza takie zachowanie, gdyż jego zdaniem kod jest mniej przejrzysty i bardziej podatny na ludzkie błędy.

Jeśli więc ograniczamy się **tylko do jednego wyrażenia**, to nie musimy (oczywiście możemy) używać klamer. Taki zapis jest poprawny:

```
if (a <= b)
System.out.println(b - a);
```

czy taki, który jest tym samym, ale inaczej sformatowanym:

```
if (a <= b) System.out.println(b - a);

if (a <= b)
    // jest elementem instrukcji warunkowej - tylko pierwsza instrukcja, gdy nie ma klamer
    System.out.println(b - a);
    // nie jest już elementem instrukcji warunkowej
    System.out.println(b - a);
```

Ten zapis wyżej jest tożsamy z tym:

```
if (a <= b) {
    System.out.println(b - a);
}
System.out.println(b - a);
```

Natomiast zobacz, że nawet gdy chcemy coś zrobić w jednej instrukcji, ale bez klamer, to może się to nie udać. Tutaj otrzymamy już błąd:

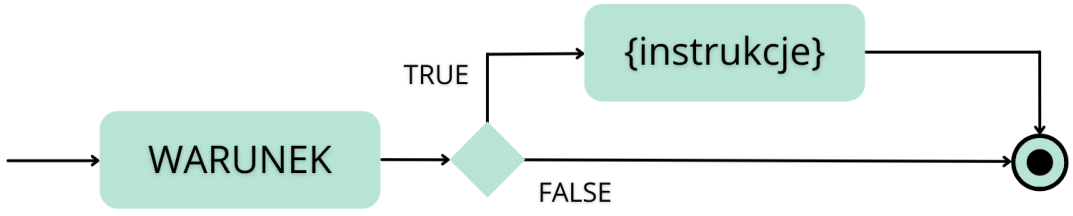
```
if (a <= b) int c = 5; // błąd - deklaracja nie jest możliwa w takiej składni (bez klamer)
```

tutaj już ok:

```
int a = 1;
int b = 3;
if (a <= b) a = 5; // przypisujemy nową wartość, takie działanie jest możliwe bez {}
```

```
// w klamrach możemy już deklarować zmienne.
if (a <= b) {
    int c = 5;
    // w klamrach możemy już umieszczać wszystkie instrukcje. Lepiej więc dawać klamry! ;)
}
```

Schemat przepływu - instrukcja if



Obraz 1. Diagram przepływu instrukcji if

Instrukcja `if` jest najprostszą odmianą instrukcji warunkowej. Może się jednak zdarzyć (i zdarzy się bardzo często), że chcemy sprawdzić nie jeden warunek a kilka i każdemu z nich przyporządkować inne działanie.

Wyobraźmy sobie taką sytuację:

```
Idę po chleb do sklepu
Wchodzę do sklepu
Jeśli jest chleb -- kupię chleb
Jeśli nie ma chleba -- nie kupuję chleba
```

To najprostsza sytuacja i tu `if` wystarczy, bo jedyne działania, które podejmujemy, to kupienie chleba w przypadku, gdy będzie w sklepie.

Ale może warto byłoby ten program modyfikować?

```
Jeśli jest chleb -- kupię chleb
Jeśli nie ma chleba -- zapytam kiedy będzie chleb!
```

Przy czym, gdybyśmy napisali to w ten sposób:

```
boolean czyJestChleb = true;
if (czyJestChleb) {
    System.out.println("Kupuję chleb");
}
System.out.println("Kiedy będzie chleb?");
```

Wydrukuje:

```
Kupuję chleb
Kiedy będzie chleb?
```

Kod powyżej sprawił, że zarówno zostało wyświetlone "*Kupuję chleb*" jak rónież "*Kiedy będzie chleb*". No bez sensu ☺. Zwróć uwagę, że drugi println jest już poza `if-em` i w tej sytuacji będzie wywołany zawsze. A tego w naszym akurat programie nie chcemy. I tu na horyzoncie pojawia się odmiana instrukcji `if`, zwana `if-else`.

Instrukcja if-else

```
if (czyJestChleb) {
    System.out.println("Kupuję chleb");
} else {
    System.out.println("Kiedy będzie chleb?");
}
```

Teraz super. Jeśli warunek "*czyJestChleb*" zwróci `true`, wykona się pierwszy blok, a jeśli nie (zwróci `false`), wtedy wykona się kod znajdujący się w bloku po `else`.

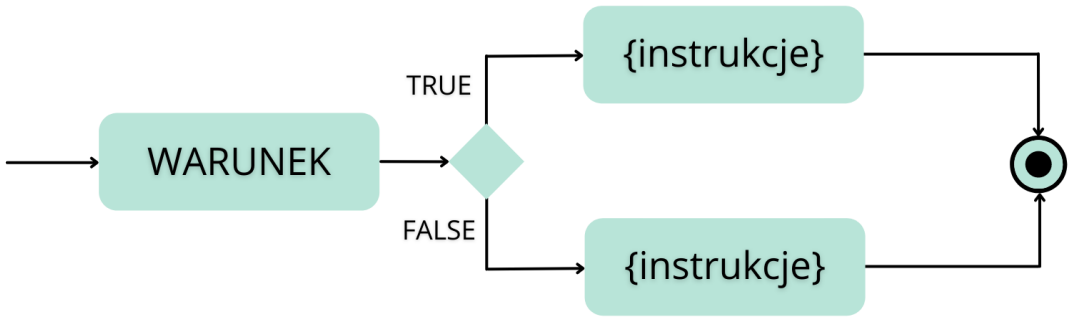
Przyjrzyjmy się składni instrukcji if-else:

```
if (bmi > 40) Wydrukuj("uuu, trzeba coś zrobić, ważysz za dużo");
else Wydrukuj("nie masz otyłości 3. stopnia");
```

Tak jest ok, ale lepiej pamiętać o klamrach:

```
if (bmi > 40) {
    Wydrukuj("uuu, trzeba coś zrobić, ważysz za dużo");
} else {
    Wydrukuj("nie masz otyłości 3. stopnia");
}
```

Schemat przepływu - instrukcja if-else



Obraz 2. Diagram przepływu instrukcji if-else

Ostatnią odmianą instrukcji warunkowej `if` jest instrukcja **if-else-if**.

Instrukcja if-else-if

Przykład:

```
String color = "red";
String canUserWalk = null;

if (color.equals("red"))
    canUserWalk = "Achtung, nein!";
    // jeśli red, to przypisz do zmiennej canUserWalk stringa "Achtung, nein!"
else if (color.equals("yellow"))
    canUserWalk = "just a moment my friend";
else if (color.equals("green"))
    canUserWalk = "sznela, now you can";
else
    canUserWalk = "światełko się zepsuło";

System.out.println(canUserWalk); // w tym wypadku "Achtung, nein!"
```

Pamiętaj, też, że program sprawdzi `else-if`, tylko jeśli `if` będzie nieprawdziwy. Kolejny else `if`, jeśli poprzedni okaże się nieprawdziwy itd. Jeśli żaden z warunków nie będzie prawdziwy, wykona `else` (o ile, oczywiście, `else` będzie, bo nie jest wymagany).

Przykład powyżej nie zawiera nawiasów blokowych, ponieważ zawiera tylko jedną instrukcję, więc nie

jest to konieczne, ale Karol sugeruje dodawać ☺.

O jeszcze jednej rzeczy musimy pamiętać. Zwróć uwagę w przykładzie poniżej, że w każdym warunku warunek jest prawdziwy, co to oznacza?

```
int value = 100;

if (value > 5) {
    // kod 1
} else if (value > 50) {
    // kod 2
} else if (value >= 100) {
    // kod 3
}
```

Taka sytuacja, oczywiście, się zdarza i nie jest to żaden błąd (przynajmniej logiczny). Oczywiście, zostanie wykonany blok tylko tej części instrukcji, który jest prawdziwy jako pierwszy. W naszym wypadku będzie to kod 1. Dalej instrukcja nie sprawdza już warunku, tylko kończy działanie pętli.

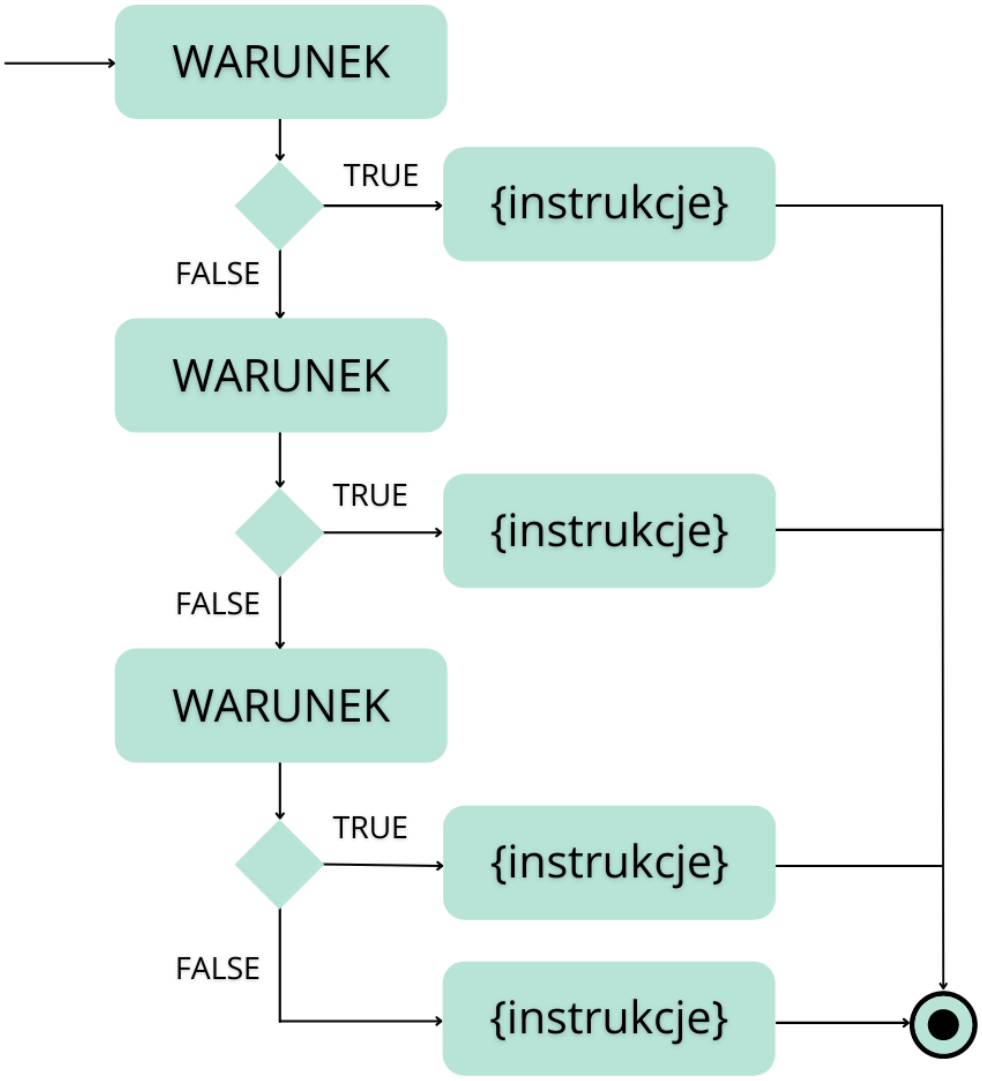
Drugi przykład:

```
int value = 10;

if (value < 5) {
// kod 1
} else if (value <= 10) {
// kod 2
} else if (value < 100) {
// kod 3
}
```

Program sprawdza warunki od góry do dołu. `If` zwróci tutaj `false`, bo 10 nie jest mniejsze od 5. Za to 10 jest równe 10, więc drugi warunek jest już spełniony. Podobnie trzeci, ale on już nie będzie sprawdzany, ponieważ wykonanie instrukcji zakończyło się na drugim warunku.

Schemat przepływu - instrukcja if-else-if



Obraz 3. Diagram przepływu instrukcji if-else-if

Ternary operator - operator trójargumentowy

Ostatnie zagadnienie omawiane tego dnia to ternary operator, po polsku operator trójargumentowy - jest to jedyny trójargumentowy* operator, więc nie będzie pytań, o który trójargumentowy operator chodzi ☺. Popularny w wielu językach i mający swoje zalety, dlatego do zapamiętania, bo się przydaje. Potwierdzam, to jest dosyć często używany zapis.

`+` to operator dwuargumentowy a `++` to operator jednoargumentowy. Argumenty operatora to operandy, np. `2 + 3` to operand pierwszy (liczba `2`), operator (znak `+`) i operand drugi (liczba `3`) - ot taka ciekawostka.

Zobaczmy podstawową składnię:

```
x ? y : z;
```

tłumacząc:

```
wyrażenie ? jeśli wyrażenie prawdziwe, weź to : jeśli wyrażenie nie jest prawdziwe, weź to;
```

czy jeszcze inaczej:

```
x jest true? Jak true, weź to : Jak false, weź to;
```

a w praktyce:

```
int x = 2;
x <= 10 ? "Liczba x jest mniejsza od 10 lub równa 10" : "Liczba x jest większa od 10";
```

Operator trójargumentowy jest dobrym rozwiązaniem, jeśli chcemy zwrócić jakąś wartość i zapisać ją w zmiennej. W takiej sytuacji świetnie zastępuje instrukcję warunkową.

```
var age = 20; ①
boolean isAdult = (age >= 18) ? true : false; ②
System.out.println(isAdult);
```

① `var` dla przypomnienia, możemy też użyć `int`.

② Nawias nie jest konieczny, ale użyłem go tu, by pokazać, że, oczywiście, można go użyć, a niektórzy lubią, bo wtedy widzą wyraźnie sprawdzane wyrażenie, podobnie jak w instrukcji warunkowej.

Trójargumentowy operator może być wykorzystany jako zwracana wartość, wtedy tworzy nam się dodatkowe zagłębienie (sprawdzenie). Przy dwóch jak w przykładzie jest to jeszcze zrozumiałe, ale przy większej liczbie lepiej rozważyć instrukcje warunkowe.

```
long usersNumber = 1200;

String feedback = (usersNumber > 100)
    ? usersNumber > 1000
        ? "Impreza duża!"
        : "Impreza mała"
    : "Impreza się nie odbędzie :(";

System.out.println(feedback); // Impreza duża!
```

Chcę tu dołożyć jeden komentarz. Ternary operator może być używany tylko, gdy jego wywołanie zwraca jakiś rezultat. To znaczy, nie możemy zapisać czegoś takiego:

```
int x = 5;
x == 5 ? System.out.println("==5") : System.out.println("!=5");
```

Możemy to wytłumaczyć w ten sposób, że metoda `System.out.println()` podczas drukowania konsumuje przekazaną do niej wartość, a sama nic nie zwraca. Tak jak napisałem, z racji, że nic nie zwraca, nie

może zostać w tym przypadku użyta.

Switch

W większości wypadków instrukcja `if` (i jej odmiany) będzie właściwym wyborem dla instrukcji tworzących alternatywną ścieżkę wykonania programu. Jednak w pewnych przypadkach warto skorzystać z **instrukcji warunkowej switch**.

Zobaczmy najpierw schemat:

```
switch(sprawdzane-wyrażenie) {
    case wartość1:
        // kod
        break;
    case wartość2:
        // kod
        break;
    default:
        // kod
}
```

Najlepiej od razu przejść do przykładu:

```
String day = "poniedziałek";

switch (day) {
    case "poniedziałek":
        System.out.println("Zaczynamy nowy tydzień!");
        break;
    case "piątek":
        System.out.println("Zaczynamy nowy tydzień!");
        break;
    default:
        System.out.println("Ani to poniedziałek, ani piątek");
        // wydrukowane zostanie "Zaczynamy nowy tydzień!"
}
```

Instrukcję `switch` tworzymy przez użycie słowa kluczowego `switch` i określenie wyrażenia, które będzie sprawdzane. Następnie otwieramy za pomocą nawiasów klamrowych blok instrukcji warunkowej `switch` i wpisujemy, jakie wartości będziemy sprawdzać. Jedyne porównanie, jakie tu nastąpi, to to, czy dane wyrażenia są takie same (a więc w praktyce "==").

By zadeklarować sprawdzane przypadki, używamy słowa kluczowego `case` i wskazujemy wartość. W naszym przykładzie mamy sprawdzenie:

```
day == case1; day == case2
```

Każdy z tych warunków jest sprawdzany.

Zobaczmy to na kolejnym przykładzie:

```
String userName = "Jan";

switch (userName) { // wybieramy wyrażenie, które sprawdzimy
    case "Stefan": // wskazujemy przypadek 1
        System.out.println("Zaczynamy nowy tydzień!");
        // wskazujemy instrukcje (oczywiście może być więcej niż jedna)
        break;
        // break mówi, że tutaj należy przerwać wykonywanie instrukcji
    case "Adam": // wskazujemy przypadek 2 (możemy ile chcemy)
        System.out.println("Zaczynamy nowy tydzień!");
        break;
    default:
        System.out.println("Ani to poniedziałek, ani piątek");
        // jeśli default jest na końcu, to "break" nie jest już potrzebny
}
// wydrukowane zostanie "Ani to poniedziałek, ani piątek", ponieważ żadna inna wartość tu nie pasuje.
```

Na co zwrócić uwagę!

Dobrą praktyką jest umieszczanie default na końcu, choć nie jest to obowiązkowe. Jeśli default jest umieszczony na końcu, nie trzeba już w nim stosować instrukcji break (bo nic po nim nie ma i instrukcja warunkowa zakończy działanie).

Zobacz, na czym może polegać problem:

```
String userName = "Jan";

switch (userName) {
    default:
        System.out.println("Ani to poniedziałek, ani piątek");
    case "Adam":
        System.out.println("Zaczynamy nowy tydzień!");
        break;
}
/* Wydrukowane zostanie zarówno "Ani to poniedziałek, ani piątek",
jak i "Zaczynamy nowy tydzień!" ponieważ zastosowany zostanie default
(żaden case nie jest równy wyrażeniu userName), default nie ma za sobą w tym wypadku break,
dlatego program pójdzie dalej aż do końca ciała instrukcji switch albo do pierwszego breaka,
w naszym przykładzie napotka break dopiero po wydrukowaniu "Zaczynamy nowy tydzień"! */
```

Rozwiązanie? Albo default na końcu, albo break także w default.

Grupowanie warunków

```
String day = "piątek";
    switch (day) {
        case "poniedziałek":
        case "wtorek":
        case "środa":
        case "czwartek":
        case "piątek":
            System.out.println("Do roboty");
            break;
        default:
            System.out.println("W weekend odpoczywam!");
    }
    // wydrukuje w tym przypadku "do roboty"
```

Zwróć uwagę, że możemy napisać, po sobie, różne `case`, które wykonają te same instrukcje. Wystarczy, że jeden z nich pasuje, tak jak w naszym przypadku.

Pamiętaj, że kod zacznie wykonywać się przy pierwszym spełnionym warunku (czyli wyrażenie równe `case` lub brak `case` równego wyrażeniu i wtedy, o ile jest, wykona się `default`). Kod będzie się wykonywał do zakończenia instrukcji albo do napotkania słowa `break`, co zakończy instrukcję. W takiej sytuacji, jak poniżej, zostanie zgłoszony błąd, bo będzie to duplikat.

```
switch (day) {
    case "piątek":
        System.out.println("Do roboty");
        break;
    case "piątek": // kompilator zgłosi błąd: Duplicate case
        System.out.println("Piątek");
}
```

Pamiętaj też, że `default` nie jest wymagany. Jest on jak else w instrukcji warunkowej. Przykład poniżej zapiszemy w instrukcji `if` i w instrukcji switch. Efekt ten sam. Jaką wybrać, zależy od wrażliwości programisty ☺.

```
switch (day) {
    case "piątek":
        System.out.println("Jak dobrze");
        break;
    case "środa":
        System.out.println("Jeszcze tylko dwa dni, dam radę");
        break;
    default:
        System.out.println("gdzie ja jestem?");
}
// To samo w instrukcji if
if (day.equals("piątek")) {
    System.out.println("Jak dobrze");
} else if (day.equals("środa")) {
    System.out.println("Jeszcze tylko dwa dni, dam radę");
} else {
    System.out.println("gdzie ja jestem?");
}
```

Goto

W Javie mamy słowo zastrzeżone "goto", ale nic ono nie robi ☺. I tyle!