Instrukcje Warunkowe

#if #else #switch #condition

Author: Piotr Niemczuk
Nickname: Pyoneru

Teoria

Aby przygotować danie jakim jest nasz program potrzebujemy składników oraz instrukcji.

Poznałeś już składniki - czyli dane, oraz troszeczkę instrukcji - wszystko to co modyfikowało nasze dane.

Filmy oglądamy.
Gry ogrywamy.

Gry czy programy mają tą różnice że **wykonują** coś w **zależności** od naszych decyzji.

Oglądając film jesteśmy tylko biernymi obserwatorami, nie mamy wpływu na przebieg fabuły. Co innego gry, nie rzadko mamy wpływ nie tylko na to jakie wyzwanie podejmiemy ale też jak je zrealizujemy. Nie jesteśmy tutaj tylko biernymi obserwatorami, ale aktywnymi uczestnikami, mamy wpływ na rozwój wydarzeń.

Serfując po necie również tego doświadczamy.
Jaką stronę załadować?
Na jakie konto się zalogować?
Czy dane są poprawne?

Na te wszystkie pytania, przeglądarka daje nam odpowiedź.

Jak już się zapewne domyślasz, w dzisiejszym materiale dowiesz się jak postępować w zależności od czegoś, w zależności od warunku.

Jednak zanim przejdziemy do praktyki, przeanalizujmy sobie jakiego rodzaju warunki możemy mieć.

1. Warunek prosty
   Jeżeli napotkasz przeszkode na swojej drodze to ją omiń.

Prosty warunek który weryfikujemy spacerując, cały czas.
Jeżeli stoi coś lub ktoś nam na drodze, co nie umożliwi nam dalszą drogę to, to omijamy.
Jeżeli nic takiego nie ma to nie robimy dodatkowych rzeczy.

2. Warunek z pod warunkiem

Jeżeli napotkasz przezkode na swojej drodze to ją omiń
Jeżeli jesteś przy pasach i jest czerwone światło to się zatrzymaj

Koncentrujemy się na omijaniu przeszkód, jeżeli jednak dojdziemy do pewnego punktu (passów) i światło będzie czerwone to powinniśmy się zatrzymać.

3. Wybór
   Którą opcje z menu wybrać?

Jesteśmy w restauracji i mamy wiele opcji w menu do wyboru. Którąś musimy wybrać.

I to tyle z praktyki :)

Praktyka

Komentarze

Szczerze mówiąc nie wiedziałem gdzie to wcisnąć chociaż wykorzystuje to cały czas w materiałach.

```
int x = 5; // to jest komentarz
```

Komentarz to fragment kodu który jest kompletnie ignorowany podczas kompilacji, tak jakby go nie było. Nie ma on zupełnie wpływu na działanie naszego programu.

Jak już pewnie zauważyłeś/aś to jego celem jest opisanie framentów kodu, dopowiedzenie co się tu dzieje.

```
/*
 Jeżeli x jest równe 5 to wykonaj coś
*/
if(x == 5){
```

```
    // coś
}
```

Komentarze możemy podzielić na jedno-liniowe i wielo-liniowe.

Jedno liniowe zaczynają się od `//` i wszystko co znajduję się na prawo od nich, aż do końca linii jest ignorowane przez kompilator.

```
x += 5; // to juz jest ignorowane przez kompilator
```

Wielo linowe zaczynają się od `/*` i kończą się na `*/`. Wowczas możemy między tymi znakami pisać co nam się podoba i nie będzie to miało wpływu na nasz program.

```
/*
 * Class to represent point on the grid.
 * Basic arithmetic operation on points.
 * Key type is defined in GridSettings.h
 */
```

Komentarze są czesto stostowane do wyłączenia kodu z kompilacji.

```
int x = 5;
// x *= 10;
System.out.println(x);
```

⚠ Uważaj na zagnieżdzenie komentarzy!

Uważaj na zagnieżdzenie komentarzy. Może się zdarzyć że będziesz chciał najpierw zakomentować mniejszy fragment kody, a następnie większy w którym znajduje się ten mniejszy.

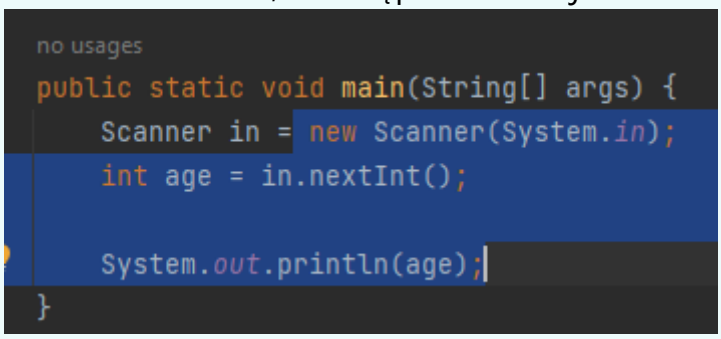
```
/*
int x = 5;
/*
x %= 3;
x = x + 1;
*/
```

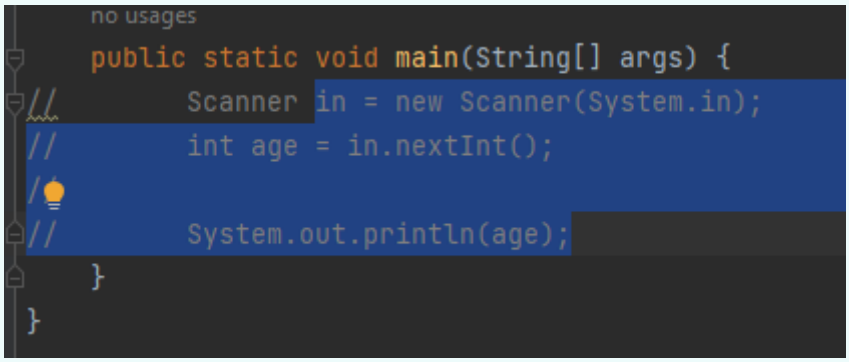
```
int y = 0;
*/
```

Jednakże, komentarze się nie zagnieżdżają. Możesz to łatwo zaobserwować po kolorze składni.
Znak `*/` mniejszego framentu tak na prawdę kończy komentarz większego fragmentu kodu. Uważaj na to!

IntelliJ TIP

Zaznacz wiele linii, a następnie naciśnij kombinacje `ctrl+/`





W ten sposób możesz w szybki sposób zakomentować wiele linii. Tą samą kombinacją klawiszy je odkomentujesz.

Warunek if

```
int age = 18;

if(age >= 18)
        System.out.println("Jesteś dorosły!");
```

Wypróbuj powyższy kod! Zmień wartość zmiennej `age` i zobacz co się stanie.

Z łatwością się domyślasz że dopóki wartość zmiennej `age` jest większa lub równa `18` to na ekranie konsoli zostanie wypisany tekst `Jesteś dorosły!`.

Następny przykład

```
int a = // coś
int b = // coś 2
String operation = "dodaj"; // co z tym zrobić?
if(name.equals("dodaj")){
        int result = a + b;
        System.out.println("Result: " + result);
}
```

Podsaw pod `a` i `b` wartości i uruchom program!

Anatomia if'a

```
if(WARUNEK)
        INSTRUKCJA
// lub
if(WARUNEK){
        INSTRUKCJA1
        INSTRUKCJA2
        INSTRUKCJA3
        //...
}
```

Jeżeli chcemy aby nasz program wykonał kod tylko pod jakimś warunkiem to należy użyć składni **if** która składa się z..

1. Słowa kluczowego **if**
2. Warunku, czyli wartości **boolean** zawartej pomiędzy nawiasami `()`
3. Intrukcje które mają się wykonać po spełnieniu warunku, czyli kiedy jego wartość jest **true**.

Jeżeli **if** zawiera tylko jedną instrukcje do wykonania to wówczas możemy pominąć nawiasy `{}`. Jeżeli ma więcej niż jedną instrukcje to musimy już uzyć nawiasów `{}`

Nic nie stoi nam na przeszkodzie żeby stworzyć ifa wewnątrz if'a!

```
int age = // coś
char sex = // coś

if(age >= 18){
        if(sex == 'k'){
                System.out.println("Jestem pełnoletnia!");
        }
        if(sex == 'm'){
                System.out.println("Jestem pełnoletni!");
        }
}
```

Blok kodu, a czas życia zmiennej

Funkcja **main** czy warunek **if** korzystają z tzw. bloku kodu. Czyli instrukcji zawartych pomiędzy nawiasami `{}`.

Dobrze znać następstwa korzystania z bloku kodu, a zwłaszcza jego zagnieżdżania.

Do tej pory pisałeś/aś program tylko wewnątrz jednego bloku kodu,

```
public static void main(String[] args){
        String name = "Shiro Miro Ika Ziro";

        System.out.println(name);

}
```

więc nie zdążyłeś/aś zauważyć że zmienne mają swój czas życia który jest właśnie ograniczony blokiem kodu.

```
public static void main(String[] args){
        String name = "Shiro Miro";
        boolean addLastName = true;
        if(addLastName){
                String lastName = "Ika Ziro";
                 name += " " + lastName;
        }
        System.out.println(name);

}
```

Mamy mały program, który w zależności od zmiennej `addLastName`, doda `lastName` przed wypisaniem pełnego imienia.

Wewnątrz warunku **if** tworzymy nową zmienną `lastName`.

Konsekwencją użycia bloku kodu jest to że czas życia zmiennej `lastName` kończy się wraz z klamrą zamykającą `}` warunku **if**

```
public static void main(String[] args){
        boolean readName = false
        if(readName){
                String name = "Shiro Miro Ika Ziro";
        } // koniec if
        System.out.println(name); // Błąd! zmienna name nie istnieje tutaj

}
```

Jeżeli chcielibyśmy pobrać nazwę użytkownika, ale tylko wtedy kiedy będzie on tego chciał, to tworząc zmienną dla jego nazwy **wewnątrz** if'a, robimy błąd.

Tworzymy zmienną `name`
if się konczy, więc wszystkie zmienne stworzone wewnątrz if'a zostają wyczyszczne.

Zmienna `name` nie jest już dostępna poza if'em.

> ⓘ **A co jeśli nie ma klamr?**
>
> Brak klamr dla jednej instrukcji jest tylko dla naszej wygodny. Wciąż obowiązują wszystkie zasady tak jakby te klamry wciąż tam były.

Warunek if..else

Wiemy jak wykonać kod jeżeli zostanie spełniony **warunek**. A co jeśli chcielibyśmy wykonać inny kod, ale wtedy kiedy **warunek** nie zostanie spełniony?

```
int age = // coś
if(age >= 18){
        System.out.println("Jesteś pełnoletni!");
}
if(age < 18)
        System.out.println("Nie jesteś pełnoletni!");
```

Moglibyśmy stworzyć drugiego if'a, z warunkiem przeciwnym.

Jednak mamy do dyspozycji coś lepszego

```
int age = // coś
if(age >= 18){
        System.out.println("Jesteś pełnoletni!");
}else{
        System.out.println("Nie jesteś pełnoletni!");
}
```

else wykona się kiedy warunek z **if'a nie** zostanie spełniony.

else i if możemy zagnieżdżać

```
int age = // coś
if (age >= 100){
        System.out.println("'Sto lat' już nie wypada na urodziny ;)");
}else if(age >= 18){
        System.out.prinln("Jesteś pełnoletni!");
}else{
        System.out.println("Nie jesteś pełnoletni!");
}
```

Możemy to przetłumaczyć jako

- Jeżeli Twój wiek jest równy lub większy niż 100 to zrób.. coś
- W innym wypadku jeżeli Twój wiek jest równy lub większy niz 18 to zrób.. coś
- W innym wypadku zrób jeszcze coś innego.

Uważaj na zagnieżdżanie if'ów i else'ów. Łatwo się pomylić.

```
if(something) // cos
        if(something2) // cos
        else // cos innego
else
        // cos innego
```

Który else należy do którego if'a?

```
if(something) //..
if(something2){ //..
if(something3)//..
else //..
}
else if(something4) //..
if(something5) //..
if(something6) //..
else //..
```

A teraz ? :)

Warunek switch

Ostatnim zagadnieniem jest warunek **switch**, działa on podobnie do wielokrotnego if..else.

Wygląda on następująco

```
char operator = \\..
switch(operator){
        case '+':
                // dodawanie
        break;

        case '-':
                // odejmowanie
        break;

        case '*':
                // mnożenie
        break;

        case '\\':
                // dzielnie
        break;

        default: // opcjonalnie
                // nie rozpoznany operator
        break;
}
```

Warunek **switch** działa w ten sposób że podajemy w nawiasach `()` jakąś zmienną, a następnie sprawdzamy czy równa się ona jednej z wartości przy **case**.
Możemy również coś zrobić jeżeli podana wartość nie będzie równa żadnemu przypadkowi, jest to słowo kluczowe **default**, jednak jest ono opcjonalne.

Po co nam ten **break**?

case działa tak na prawdę jako wyznacznik gdzie ma rozpocząć się wykonywanie kodu. Więc jeżeli go nie przerwiemy to mogą się wykonać nawet kilka warunków **case**, nawet jeżeli następne **case** nie są równe przekazanej zmiennej.

Słowo kluczowe **break** służy do przerwania danego bloku kodu. Możemy go używać wewnatrz właśnie warunku **switch** lub pętli które poznasz w następnym materiale.

```
switch(x){
        case 1:
        // cos
        case 2:
        // cos
        break;
        case 3:
        // cos
        break;
}
```

Jeżeli mamy taki kod to gdy:

- x = 1, wykonywanie kodu rozpocznie się od **case 1**, jednak nie ma tam **break** więc przejdzie on do **case 2**. Po **case 2** się zatrzyma ponieważ napotkał **break**
- x = 2, wykonywanie kodu rozpocznie się od **case 2**, po nim napotka **break** więc **switch** się zakończy
- x = 3, wykonywanie kodu rozpocznie się od **case 3**, po nim napotka **break** więc **switch** się zakończy

> ⚠ **default również dotyczy ta zasada**
>
> Sekcje **default** możemy umieścić w dowolnej koleności, równiez na sam początek. Jednak jeżeli zabraknie jej **break** to po **default** przejdzie do następngo w kolejce **case**.

```
switch(x){
        default:
        System.out.println("default")
        case 1:
        Sytem.out.println("1")
}
```

Output kiedy **x** jest różny od 1

```
default
1
```

Scanner

Na zakończenie coś ciekawego :) Nauczysz się jak odczytywać dane od użytkownika, co sprawi że Twoje programy staną się bardziej dynamiczne!

Aby móc odczytywać dane od użytkownika (input), musimy stworzyć obiekt **Scanner**.

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

Pamiętasz jeszcze operator new? ;)

Scanner jest typem złożonym wbudowanym w język java, więc aby go stworzyć musimy użyć operatora **new**.

`System.in` oznacza że mówimy iż Scanner ma odczytywać dane z konsoli. Scanner'a również możemy użyć do odczytywania z plików. Skupmy się dziś jednak na konsoli.

Kiedy mamy już gotową zmienną typu **Scanner** to możemy zacząć używać jej metod!

ⓘ Metody

W przeciwieństwie do typów prostych, typy złożone mogą mieć metody. Metoda to blok funkcji który wykonuje dla nas typ złożony, a następnie zwraca dla nas jakąś wartość, jakiegoś typu.
Aby skorzystać z metody najpierw musimy stworzyć obiekt (zmienną) tego typu. Następnie po nazwie zmiennej dodajemy `.` i nazwe metody, na końcu metody

zawsze znajdują się nawiasy `()` i średnik. Nawiasy mogą przyjmować jakieś paramtetry tak jak **Scanner** przyjmuje paramtetr `System.in` przy jego tworzeniu.

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

String line = in.nextLine(); // String
String age = in.nextInt();  // int
String ageOfTheEarth = in.nextLong(); // long
String pi = in.nextDouble(); // double
```

ⓘ Scanner może więcej

Kiedy korzystasz z IntelliJ to zapewne zauważyłeś/aś że po dodaniu `.` wyświetliła się lista. IDE podpowiada nam co dla nas może zrobić **Scanner**. Nie wszystkie jego metody mogą być dla Ciebie jasne na ten moment :)

Przykład użycia Scanner'a, wypróbuj sam/a!

```
Scanner in = new Scanner(System.in);
System.out.print("Podaj wiek: "); // print, nie println. Podmień na println i zobacz różnice
int age = in.nextInt(); // W tym miejscu działanie kodu się 'zatrzyma', dopóki nie podamy wartości

if(age >= 18){
        System.out.println("Jesteś pełnoletni!");
}else{
        System.out.println("Nie jesteś pełnoletni!");
}
```

⚠ Error!

Kiedy użyjemy metody **nextInt()**, to program oczekuje od nas liczby całkowitej. Jeżeli jednak podamy coś innego, litere, tekst czy liczbe zmienno przecinkową to dostaniemy błąd.

```
asd
Exception in thread "main" java.util.InputMismatchException Create breakpoint
    at java.base/java.util.Scanner.throwFor(Scanner.java:939)
    at java.base/java.util.Scanner.next(Scanner.java:1594)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
    at Main.main(Main.java:8)
```

Obsługą takich błędów zajmiemy się kiedy indziej. Ale nikt Ci nie zabrania kombinować ;)

Podsumowanie

Dzisiaj poznałeś/aś intrukcje warunkowe. Pierwszy duży krok w programowaniu, od tej pory będziesz pisać więcej programów!

> *Aby pisać, trzeba pisać.* ~Ktoś tam