Operacje Arytmetyczne i Logiczne

#operacje #arytmetyczne #logiczne #

Author: Piotr Niemczuk
Nickname: Pyoneru

Teoria

W poprzednim materiale poznałeś/aś Zmienne & Typy danych. W tym materiale nauczysz się jak pracować ze zmiennymi.

Operacje arytmetyczne

Operacje arytmetyczne to wszelkie działania matematyczne jakich uczyłeś/aś się w szkole.

Dodawanie, odejmowanie, mnożenie, dzielenie

Lol, tego nie trzeba tłumaczyć. Jeśli nie umiesz to masz się z czego wstydzić. I słusznie.

Reszta z dzielenia

Kiedy dzielisz jedną liczbe przez drugą, to sprawdzasz ile razy ta liczba mieści się w drugiej.
co?

Idziesz z trójką znajomych na pizze. Jest was czworo (Ty + trójka znajomych, `1 + 3 = 4`).
Pizza zazwyczaj jest krojona na 8 kawałków.

Szybkie dzielenie.
8 Kawałków pizzy podzielić na 4 osoby równa się 2 kawałki pizzy na łebka.
`8 / 4 = 2`

Cyfra 4 mieści się **dokładnie** 2 razy, w cyfrze 8.
Jeżeli pomnożymy 4 przez 2 to równa się to 8 <- mnożenie jest odwrotnością dzielnia.

```
8 / 4 = 2
4 * 2 = 8
8 / 2 = 4
```

Proste.

Co w przypadku gdy dojdzie do nas jeszcze jedna osoba? To już jest pięć osób na jedną pizze.

Liczba kawałków pizzy się nie zmieniła, więc wciąż wynosi 8.
Teraz jest nas 5.

`8/5 = 1.6`

Coś tu nie pasuje. Nie podzielimy po równo 8 kawałków pizzy na 5 osób.
Każdemu będzie przysługiwał jeden kawałek pizzy (czyli łącznie 5), ale zostanie nam do podzielenia 3 kawałki pizzy na 5 osób.
To jest właśnie reszta z dzielenia

Kiedy dzieliliśmy `8 / 4` to reszta z dzielenia wynosiła `0`, ponieważ cyfrę **4** zmieściliśmy **dokładanie dwa razy** w cyfrze *8* i nic nie zostawało.

Natomiast kiedy dzielimy `8 / 5` to reszta z dzielenia wynosi `3`, ponieważ cyfrę **5** zmieścimy **tylko jeden raz** w cyfrze **8**, zostaje jeszcze **3** bo **5** już się tam nie zmieści.

ⓘ Co nam to daje?

Zauważ że reszta z dzielenia występuje w dwóch przypadakch.

- Kiedy jesteśmy w stanie umieścić dokładnie tyle cyfr z prawej strony, w cyfrze po lewej stronie(`8/4`), aby nic nam nie zostało. Reszta z dzielenia jest wtedy równa **0**.
- Kiedy nie jesteśmy w stanie w pełni umieścić tyle cyfr z prawej strony, w cyfrze po lewej stronie(`8/5`), że zostaje nam coś w co już nie zmieścimy cyfry z prawej strony (`3/5`). Wówczas otrzymujemy reszte z dzielenia większą od 0.. **i mniejszą od cyfry z prawej strony!**

Czyli reszta z dzielenia dla liczby **X** może wynieść **od 0 do X - 1**.

Liczby parzyste i nie parzyste

Liczby parzyste są podzielene przez 2, a nie parzyste - nie są.
Można to rozumieć że każda liczba parzystna po podzieleniu przez **2**, da zawsze reszte równą **0**.
W ten sposób możemy sprawdzić dowolną liczbe czy jest parzysta.

Zasięg

Wiedząc że reszta z dzielenia nie może być większa niż liczby przez którą dzielimy, to możemy w ten sposób stworzyć zasięg **od 0 do X - 1**.
Komputer potrafi nam wygenerować pseudolosową liczbę, która może być dowolnej wartości i może być na prawdę duża (milardy!). Więc jeżeli chcemy otrzymać liczbę z danego zakresu to możemy wyciągnąć reszte z dzielnia z tej liczby! Wówczas dostaniemy nasz zasięg **od 0 do X - 1**.
Do otrzymanego wyniku możemy jeszcze dodać np. **10**, wówczas nasz zasięg wyniesie **od 10 do X - 1 + 10**.
Przykład
Dostajemy z generatora liczb, liczbę **105**
Chcemy uzyskać liczbe z zakresu **10-19**.
Obliczamy reszte z dzielenia z **10** i otrzymujemy **5**
Dodajemy do reszty **+10** i otrzymujemy **15**
Jeżeli dostaniemy cyfrę **100** to otrzymamy **10** (reszta równa **0** i **+10**)
Jeżeli dostaniemy cyfrę **109** to otrzymamy **19** (reszta równa **9** i **+10**)

Zmiana wartości

Wyobraź sobie że spełniasz swoje najskrytsze marzenia i tworzysz swoją własną grę. Pełną mechanizmów RPG w których rozwiajsz swoją postać w tym zdobywanie doświadczenia za pokonane potwory.

Tworzysz kod odpowiedzialny za dodawanie doświadczenia za pokonanie potwora. Jak by to wyglądało czysto teoretycznie?

Powiedzmy że za pokonanego potwora zyskujemy 5 XP (Punkty doświadczenia *ang. Experience points*)

> **ⓘ Co to są te punkty doświadczenia? | Dla nie wtajemniczonych**
>
> W grach punkty doświadczenia mają odzwierciedlić to, że stajemy się w czymś lepszym. W realnym życiu oczywiście nie możemy zobaczyć ile wynoszą nasze "punkty doświadczenia" po tym jak poświęciliśmy czas na naukę lub trening. Ale wraz z biegiem czasu dzięki temu zyskujemy nowe umiejętności, umiemy coś zagrać na gitarze, rozumiemy coś w języku obcym. Nie jesteśmy mistrzami po jednej sesji, znajomość języka np. wyznaczamy poprzez poziom A1-C1 i zwiększamy go wraz z nagromadzoną wiedzą.
> W grach punkty doświadczenia i zdobyty poziom po przekroczeniu pewnego progu mają właśnie odzwierciedlić proces stawania się lepszym, silniejszym.

Grę rozpoczynamy z zerowym zdobytym doświadczeniem.

Także, tak mogłby wyglądać nasze statystyki

```
Level: 1
Collected XP: 0
```

Pokonujemy naszego pierwszego potwora i zyskujemy 5 XP. Czyli chcemy do naszej ogólnej póli doświadczenia dodać 5.

```
Zwiększ zebrane punkty doświadczenia o 5.
```

Tak będą wyglądać nasze statystyki po pokonaniu pierwszego potwora.

```
Level: 1
Collected XP: 5
```

Gdy pokonamy następnego potwora to ponownie chcemy zwiększyć nasze zebrane punkty doświadczenia o 5.

I znów zmieniają się nasze statystki. Po osiągnięciu pewnego progu, nasz poziom się zwiększa.

```
Level: 2
Collected XP: 100
```

W ten sposób modyfikujesz jakąś wartość która się zmienia, zdobyte doświadczenie, pozostałe śrubki, pozostałe kulki do zbicia.

Operacje logiczne

```
Mam bera + 15.
Jestem kobietą.
Dzisiaj padał śnieg.
Mam 15 lat.
Dzisiaj obudziłem się o 12.
```

Na każde z powyższych zdań możemy odpowiedzieć czy jest fałszywe czy prawdziwe.

```
Mam bera + 15                <- prawda.
Jestem kobietą               <- fałsz.
Dzisiaj padał śnieg          <- fałsz.
Mam 15 lat                   <- fałsz.
Dzisiaj obudziłem się o 12   <- prawda.
```

Takie zdania możemy również łaczyć

```
Wstałem o 7 rano i poszedłem na siłownie
```

Czy to **całe** zdanie jest prawdziwe ? Będzie tylko wtedy kiedy oba zdania są prawdziwe.

```
Wstałem o 7 rano
poszedłem na siłownie
```

Jeżeli chociaż jedno z nich jest nie prawdą to **całe** zdanie jest kłamstem.

Spójnik 'i'

Zauważ że spójnik '**i**' wyznacza nam że zdanie **po lewej i po prawej** musi być **prawdziwe**, aby **całe** zdanie było **prawdziwe**.

```
Zostanę programistą lub leśniczym.
```

Powyższą obietnice spełnimy jeżeli zostaniemy albo programistą/ką, albo leśniczym. Nie musimy zostać jednocześnie programistą/ką i leśniczym aby jej dotrzymać.

Spójnik 'lub'

Zauważ że spójnik '**lub**' wyznacza że wystarczy aby **jedno** zostanie było **prawdziwe**, aby **całe** zdanie bylo **prawdziwe**

Możemy też zaprzeczać, czyli negować zdania. Wrócimy do pierwszych zdań .

```
Mam bera + 15                <- prawda.
Jestem kobietą               <- fałsz.
Dzisiaj padał śnieg          <- fałsz.
Mam 15 lat                   <- fałsz.
Dzisiaj obudziłem się o 12   <- prawda.
```

i je zanegujmy

```
Nie mam bera + 15                <- fałsz.
Nie jestem kobietą               <- prawda.
Nie padał dzisiaj śnieg          <- prawda.
Nie mam 15 lat                   <- prawda.
Nie dzisiaj obudziłem się o 12   <- fałsz.
```

Negacja, inaczej zaprzeczenie

Poprzez zanegowanie zdania zmieniamy prawdziwość zdania (czy jest prawdziwe czy fałszywe) na przeciwny.
Zauważ że wszędzie tam gdzie najpierw była **prawda**, to po negacji jest tam **fałsz**.
Wszędzie tam gdzie był **fałsz**, po negacji jest teraz **prawda**.

Operacje logiczne mają nam powiedzieć czy coś jest fałsze czy prawdziwe. Możemy je łączyć ale wynikiem i tak pozostanie prawda lub fałsz.

```
Nie nacisnąłem klawisza 'SHIFT' lub 'Caps Lock' i naciśnąłem klawisz 'S'.
```

Zdanie jest prawdziwe tylko tedy kiedy nie nacisnąłem ani 'SHIFT' ani 'Caps lock' i nacisnąłem jednocześnie 'S'.
Ta operacja logiczna jest już bardziej złożona bo występują w niej więcej warunków które muszą zostać spełnione.
Wykorzystujemy tutaj spójnik '**i**', '**lub**' oraz **negację**.

ⓘ Co nam to daje?

Operacje logiczne wykorzystujemy aby podjąć jakieś działanie, właśnie w zależności od warunku.
Przykład

```
Jeżeli robisz herbate to zrób mi też.
```

Ktoś widzi że zamierzasz zrobić herbate to prosi nas żeby też mu zrobić.
Jeżeli zdanie "robisz herbate" jest prawdziwe to akcja (prośba) "to zrób mi też" jest wykonywana.
Prośba o herbate została podjęta **pod warunkiem** że ktoś zaczął robić herbate.

Praktyka

Operator przypisania

W poprzednim materiale nauczyłeś się jak przypisywać wartość do zmiennej.

```
int age = 25;
```

Znak '=' nazywamy operatorem przypisania i jest on operatorem dwu argumentowym. Oznacza to że potrzebujemy czegoś po lewej stronie i czegoś po prawej stronie.

```
LEWA_STRONA = PRAWA_STRONA
```

Przypisać wartość możemy tylko do zmiennej, więc lewa strona zawsze musi być zmienną.

```
int zmienna = 3 + 2; // Dobrze!
5 = 3 + 2; // Źle!
```

> ⚠ To nie oznacza 'równa się'
>
> Może to być dla ciebie mylące na początek, ale znak '=' **nie** oznacza że lewa i prawa strona są sobie równe.
> W matematyce znak '=' określna czy lewa i prawa strona są sobie równe -> `8 / 4 = 2`
> Natomiast w programowaniu '=' **oznacza** on operator przypisania, przypisuje on wartość z prawej strony do zmiennej z lewej strony. Nie ma to nic wspólnego z porównywaniem tych wartości.
> Przy operacjach logicznych poznasz jak porównać ze sobą dwie wartości.

Operacje arytmetyczne

Liczby całkowite i zmienno przecinkowe

Operatory dodawania, odejmowania, mnożenia, dzielenia i reszty z dzielnia (podstawowe operatory)

```
5 + 3 // dodwanie
5 - 3 // odejmowanie
4 * 2 // mnożenie
8 / 4 // dzielenie
8 % 5 // reszta z dzielenia
```

Podstawowe operatory są dwu argumentowe, ponieważ żeby wykonać jakieś działanie potrzebujemy dwóch liczb.

Jeżeli tego nie zapiszemy to wynik działania nigdzie nie zostanie zapisany.

```
System.out.println(5 + 3); // Wypisze 8 ale wynik nie zostanie nigdzie zapisany
```

```
int pizzaPerPerson = 8 / 4; // Poprzez operator przypisania, wynik zostanie zapisany do zmiennej 'pizzaPerPerson'.
```

```
3 - 2 // program się uruchomi, instrukcja się wykona ale nie będzie miała żadnego sensu. Nic nie robimy z wynikiem działania.
```

Kolejność wykonywania działań

W programowaniu występują takie same zasady dotyczące kolejności wykonywania działań co w matematyce.

```
2 + 2 * 2 // jest równe 6 ponieważ najpierw mnożenie, potem dodawanie
```

Tak samo jak w matematyce, możemy wymusić kolejność poprzez nawiasy okrągłe `()`

```
(2 + 2) * 2 // jest równe 8, teraz pierwszeństwo ma to co jest w nawiasie
```

Możemy ująć również całe wyrażenie, a następnie je rzutować na inny typ.

```
(int)(0.1 + 0.5 + 0.7) // najpierw obliczany jest wynik działania. Następnie, jako kolejna czynność, jest rzutowany do typu int.
```

⚠ Uwaga na typy!

Pamiętaj że najpierw jest wykonywane działanie które zwraca również typ wynikowy, a **dopiero** potem jest rzutowany typ.
Dlatego

```
(double)(5 / 3)
```

da wynik 1.0 ponieważ wynikiem działania `5 / 3` jest typ **int** z wartością 1, to w rezultacie rzutujesz wartość **1** typu **int** na typ **double**.
Więcej o tym przeczytasz w dodatku.

Zmiana wartości

Aby zmienić aktualną wartość, możemy wykorzystać to co poznaliśmy w tym materiale. Operator przypisania i operatory arytmetycznne.

```
int collectedXP = 0; // Zaczynamy z 0
collectedXP = collectedXP + 5; // do wartości zmiennej collectedXP dodaj 5, a następnie przypisz wynik działania do zmiennej collectedXP
```

W programowaniu bardzo często modyfikujemy aktualną wartość w zmiennej dlatego też mamy specjalne operatory na to.

```
value += 5;
value -= 5;
value *= 5;
value /= 5;
value %= 5;
```

Łączymy jeden z operatorów arytmetycznych z lewej strony (**to ważne!**) i operator przypisania z prawej strony.

Jest to po prostu skrócony zapis tego co jest powyżej.

```
int value = 0;
value = value + 5; // Zwiększ value o 5
value += 5; // Zwiększ value o 5
```

Operacje logiczne

Wszystkie operatory logiczne zwracją jeden typ - `boolean`

Operatory porównania

Mamy do dyspozycji kilka operatorów logicznych, dzięki którymi możemy sprawdzić czy dwie wartości są sobie równe, czy jedna jest większa od drugiej lub czy jest mniejsza.

```
== // porównanie
> // większa
< // mniejsza
>= // większa lub równa
<= // mniejsza lub równa
```

Operatory czytamy od lewej do prawej, czyli

```java
int age = 17;

boolean isAdult = (age >= 18); // Nawiasy są dla czytelności. Nic nie zmieniają.
```

możemy przeczytać jako

```
Czy wartość zmiennej age jest większa lub równa 18?
```

> ⚠ = to nie to samo ==
>
> Jest to mylące, zwłaszcza na początku Twojej drogi ale postaraj sie zapamiętać że = to nie to samo co ==
> = (pojedyczny) to operator **przypisania**
> == (podwójny) to operator **porównania**

> ⚠ Ten przeklęty operator == !
>
> Jeżeli będziesz używać operatora porównania dla typów prostych to program będzie działać tak jak tego oczekujesz.
>
> ```java
> int x = 5;
> boolean isFive = (x == 5)
> ```
>
> `x == 5` sprawdzi czy wartość `x` jest równa `5`. Logiczne.
> Jednak, jeżeli będziesz chciał użyć operatora porównania dla typów złożony, jak np. `String`, to prawdopobnie nie zadziała tak jak oczekujesz.
> Sprawdź poniższy kod programu aby się o tym przekonać.
>
> ```java
> String text = "same";
> String theSame = "same";
> String notTheSame = new String("same");
> System.out.println(text == theSame);
> System.out.println(text == notTheSame);
> ```

Spójnik 'i', 'lub' oraz negacja

Operacje logiczne możemy łączyć w *zdania*. Łączymy je za pomocą spójników logicznych.

```
! // negacja
&& // spójnik 'i'
|| // spójnik 'lub'
!= // nie jest równe
```

> ⚠ &&, nie &
>
> Tak, to ma znacznie. Podwójny znak '&', czyli && oznacza spójnik 'i'. Pojedynczy ma inne zaczenie i jest to operator =bitowy= AND.
> Nie działają one tak samo więc zwracaj uwagę na to!
> Spójnika '**lub**' || dotyczy to samo.

Jak pamiętasz, negacja zmienia wartość zdania na przeciwną

```
true -> false
false -> true
```

```
int age = 17;

boolean isNotAdult = !(age >= 18); // isNotAdult przyjmie true jeżeli liczba będzie mniesza od 18. Ten sam efekt możemy osiągnąć zmieniając >= na <
```

Nawiasy `()` mają już tutaj znaczenie, ponieważ negujemy całe wyrażenie.

```
!age >= 18 // error
```

Bez nawiasów, negacja `!` będzie dotyczyła tylko zmiennej `age`, a to nie ma sensu bo nie zanegujemy liczby. Negować możemy tylko wartość `true/false`.

Negacja nie zmieni znaku liczby, jeżeli chcemy zmienić jej znak to musimy dodać znak `-` przed stałą dosłowną lub zmienną

```
-5 // -5
int x = 5; // 5
int y = -x; // -5
```

Jeżeli chcemy sprawdzić czy wartość nie jest równa jakieś liczbie to możemy stworzyć takie wyrażenie

```
!(x == 0)
```

czyli najpierw sprawdzamy czy liczba jest równa 0, a następnie negujem wartość.

Jeżeli x jest różne od 0 to samo wyrażenie `x == 0` zwróci nam false, ponieważ sprawdzamy czy jest równe.

Jeżeli wyrażenie `x == 0` zwróci nam false i zanegujemy tą wartość to otrzymamy true.

W reztulatcie dostaniemy wartość czy x nie jest równe 0.

To samo możemy osiągnąć używając operatora `!=`.

```
x != 0
```

Zdanie połączone spójnikiem '**i**', zwróci nam `true` tylko wtedy, kiedy obie strony zdania są **prawdziwe**. Natomiast spójnik '**lub**' zwróci `true` wtedy, kiedy chociaż jedna strona będzie **prawdziwa**.

Warto poznać tablkę która przedstawia wszystkie możliwe wyniki

x	y	&& 'i'	|| 'lub'
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

> ⓘ **Lenistwo 'lub'**
>
> Jeżeli pierwsza część zdania jest **prawdziwa**, to drugie zdanie **nie zostanie sprawdzone**.
> Bo po co? Wystarczy nam że tylko jedna część jest prawdziwa.

```
x == 5 || x == -1
```

Jeżeli `x` jest równe `5` to drugi warunek(zdanie) `x == -1` nie zostanie sprawdzone. W przyszłości dowiesz się dlaczego to ma znaczenie.

Podsumowanie

Dzisiaj dowiedziałeś/aś się jak pracować z zmiennymi. Masz już solidne podstawy aby zacząć uczyć się już czegoś ciekawszego. W następnym materiale dowiesz się jak **sterować** swoim programem. Czyli co ma zrobić w zależności od **warunku**.