Zmienne & Typy danych

#variable #main #data_types

Author: Piotr Niemczuk
Nickname: Pyoneru

Teoria

Programowanie przypomina pieczenie ciasteczek lub składanie nowego mebla z jego instrukcją.
W obu przypadkach mamy potrzebne składniki/elementy do wykonania zadania, a także zestaw instrukcji krok po kroku, co trzeba zrobić ze składnikami, aby osiągnąć pożądany efekt.

Tworzenie programu wygląda podobnie.
Potrzebujemy składników - czyli danych, oraz instrukcji które zrobią coś z nimi.

W tym materiale skupimy się na składnikach, czyli danych.

Zmienne

Dane są przetrzymywane w tzw. zmiennych.

Za obrazujmy sobie korzystanie z zmiennych jako umieszczanie przedmiotów w magazynie.

Wyobraź sobie pusty magazyn zarządzany przez firmę X.

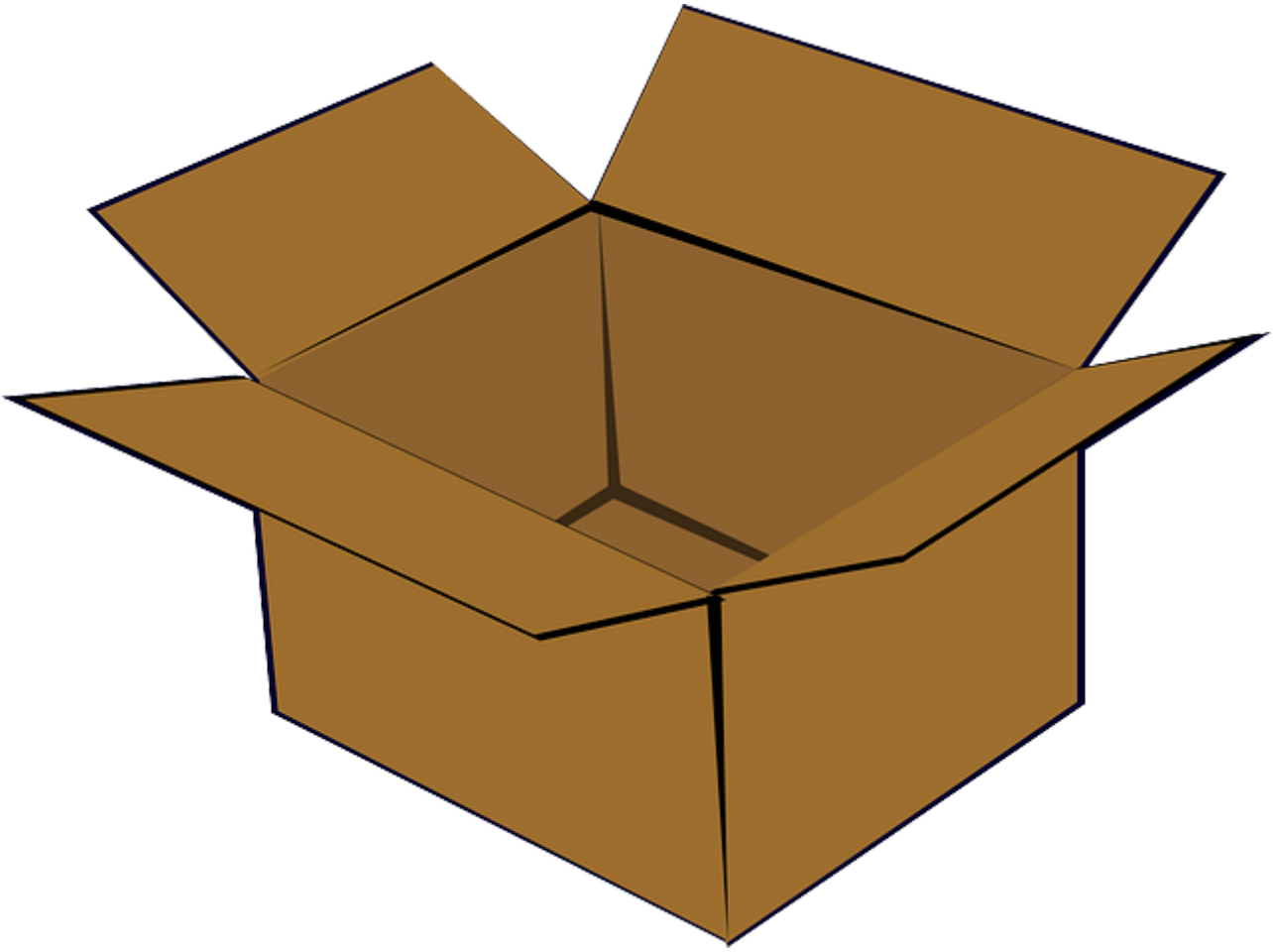
Teraz chciałbyś/chciałabyś przechować w tym magazynie swój stary telefon.

W magazynie panują jednak pewne zasady.

- Każdy przedmiot musi zostać umieszczony w specjalnym pojemniku.
- Specjalny pojemnik dostarczają pracownicy magazynu.
- Pojemniki mogą mieć różny rozmiar, w zależności od Twoich potrzeb.
- Pracownicy magazynu decydują w które miejsce wyląduje Twoje pudełko.

Dzięki temu rozwiązaniu, w magazynie udaję się zachować porządek i zachować jak najwięcej pojemników.

Zatem aby przechować swój stary telefon w magazynie potrzebujesz powiadomić pracowników że chciałbyś pojemnik. Mały.



Najmniejszy pojemnik spokojnie zmieści Twój telefon i zostanie sporo miejsca. Ale nie masz możliwości użyć mniejszego pojemnika bo ten jest już najmniejszy.

Gdy masz już pojemnik, możesz przekazać telefon do przechowania.
Ty natomiast otrzymujesz specjalną kartę dostępu z identyfikatorem Twojego pojemnika - aby mógł być w przyszłości szybko odnaleziony w magazynie.

Gdy tworzysz zmienną, to rezerwujesz miejsce w pamięci - mówisz że chcesz pojemnik o danej wielkości.
Gdy przypisujesz wartość do zmiennej, to zmieniasz kod bitowy w zarezerwowanej pamięci - przekazujesz przedmiot do umieszczenia w pojemniku.

> ⓘ **Kod bitowy**
>
> Nie będziemy działać na bitach w tym kursie, więc nie ma potrzeby wiedzieć jak wartości wyglądają pod spodem. Będziesz korzystał z dobrze Ci znanych liczb, znaków itd. (1, 3.14, "Smok"). Jednak dobrze być świadomym że wszystkie te znaki pod spodem mają swoję reprezentację bitową składającą się z zer i jedynek.

Typy danych

Mimo starań administracji, w magazynie panuje bałagan co spowolnia całą pracę. Klienci muszą długo czekać aby zarezerwować nowy pojemnik, a także z odzyskaniem swoich skarbów jest kłopot.

Dlaczego tak się stało, skoro pojemniki powinny zachować porządek i umożliwić szybki dostęp do skarbów swoich klientów?

Ludzie zaczeli wrzucać do pojemników przenajróżniejsze rzeczy, mieszać elektronike z kosmetykami, konserwy z ciuchami itd. Często wrzucali tam śmiecie, co im pod ręką się podwinęło. Oczywiście potem nie pamiętali w którym pojemniku co zostawili.

W magazynie powstał chaos, pracownicy nie nadążali za wydawaniem nowych pojemników i ich dostarczaniem. Często musieli przekopać się przez dziesiątki, a nawet setki pojemników, aż w końcu znaleźli to czego chcieli.

Administracja postanowiła ogarnąć ten bajzel i powstała nowa zasada.

- Jeden pojemnik może przechowywać tylko jeden TYP przedmiotów.
  Z rozdrobnieniem na kable, DVD, płyty DVD, szminki, szampony, mydła, mydła w

płynie, zamiast na ogólne typy, elektronikę czy kosmetyki.

Dzięki temu rozwiązaniu znów zapanował porządek w magazynie. Jednak nie możesz już wrzucać do jednego pojemnika wszystkiego co Ci się podobna.


WEDDING SHOWER
CRAFTS
GIFT BOWS
GIFT BOXES
TABLECLOTHS
CHINA
BEANIE BABIES
STUFFED ANIMALS

Dla komputera, wartość zmiennej to tylko zestaw 0 i 1. Nie interesuje go znaczenie tego zestawu.

Tak jak dla magazynu, zawartość pojemnika to tylko jego waga. Nie interesuje ich co jest w środku i do czego to służy.

Natomiast nas, jako programistów już interesuje czy w zmiennej znajdziemy wiek użytkownika, jego login, czy opłacił wpisowe. Inaczej będziemy postępować w zależności czy to będzie cyfra, tekst czy wartość logiczna - prawda lub fałsz.

Rezerwując zmienną w pamięci, będziemy informować jakiego ona będzie typu. Czyli będziemy informować co będziemy przechowywać w pojemniku i jak będziemy się z nią obchodzić.

Każdy typ danych rezerwuje określoną ilość bajtów (*ang. byte*). Czyli każdy typ ma z górny narzucony rozmiar pojemnika.

Praktyka

Funkcja main

Funkcja main to początek naszego programu. Oznacza to że gdy uruchomimy nasz program, to jego wykonywanie zacznie się właśnie od funkcji main. Każdy program napisany w Javie musi posiadać taką funkcję.

Szkielet funkcji main wygląda następująco

```
public class Main
{
    public static void main(String[] args)
    {
        // tu będziemy pisać nasz kod
    }
}
```

Szkielet jest podobny do tego co stworzyliśmy ostatnim razem (Entry Point). W naszym pierwszym programie była instrukcja która wyświetliła nam w oknie konsoli napis *Hello World!*

```
System.out.println("Hello world!");
```

> ⓘ **Co to class, static, public, void, String?**
>
> Spokojnie, poznasz znaczenie wszystkich tych słów. Ale w swoim czasie. Uwierz mi proszę, że na ten moment najlepiej abyś po prostu zapamiętał/a szkielet programu.

Zmienne

Zmienne tworzymy w następujący sposób.

```
TYP NAZWA_ZMIENNEJ;
lub
TYP NAZWA_ZMIENNEJ = WARTOŚĆ;

Przykład:
int x;
float pi = 3.14f;
char znak = 'p';
boolean czyObiadGotowy = false;
```

- na początku podajemy typ zminnej (TYP)
- następnie nazwę zmiennej (NAZWA_ZMIENNEJ), którą będziemy się posługiwać aby odczytać lub zmienić jej wartość.
- możemy jeszcze przypisać wartość podczas tworzenia zmiennej (WARTOŚĆ).

Wartość rownież możemy przypisać po utworzeniu zmiennej.

```
int x;
x = 5;
```

One more time, po co nam zmienne?

Zmienne są naszymi składnikami w przepisie na nasz program. Typ zmiennej to nasz pojemnik w którym umieścimy nasz składnik. Nazwa zmiennej informuje nas co to jest, czy ma to symbolizować ilość jajek w przepisie, czy ilość potrzebnych gwoździ. Natomiast wartość którą przypisujemy poprzez znak = to jest już nas składnik.
Przykład:

```
int egg = 5;
```

będzie nas informować że potrzebujemy 5 jajek.
Wartość tą możemy zmienić, możesz to rozumieć że w pewnym momencie będziesz potrzebował ich więcej lub mniej.
Przykład

```
int nails = 10; // ang. nails = gwoździe
// Przykręć 5 gwoździ
nails = 5; // zostalo nam 5 gwoździ
```

ⓘ Nazewnictwo

Zmienne mogą składać się z dużych i malych liter alfabetu angielskiego(bez znaków specjalnych), oraz znaku podłogi '_'. Mogą również zawierać cyfry jednak nie mogą się od nich zaczynać. Zmienne nie mogą przyjmować nazwy zarezerwowanej w języku java(keyword).

```
int liczba; //poprawnie
int 1liczba; // źle
```

```
int liczba1; //poprawnie
int tak!; // źle
int _max; //poprawnie
int int; // źle, keyword
int static; // źle, keyword
```

✓ Dobry praktyki

Nazwa zmiennej powinna nam mówić po co ona jest. Jeżeli chcemy przechować wiek użytkownika to lepiej ją nazwać *wiek* niż *x*. Gdy wrócimy po roku, miesiącu, czy nawet tygodniu do kodu to nazwa *x* nam nic nie powie co robi zmienna.

```
int wiek; // zrozumiałe
int x; // yyy, a co to robi?
```

Zmienne piszemy rownież tzw. camelCase. Zmienną rozpoczynamy od małej litery i gdy zawiera ona więcej słów to następne słowa rozpoczynamy od wielkiej litery.

```
int wiekUzytkownika;
```

Oczywiście dobrze jest też pisać angielskie nazwy :)

```
int age;
int userAge;
```

Typy danych

Typy danych dzielimy na dwie grupy, najbardziej podstawowe - typy proste (fundamentalne), są to:

- liczny całkowite -> 25
- zmienno przecinkowe -> 3.14
- pojedyczny znak -> 'p'
- wartość boolean, prawda/fałsz -> true/false.
  oraz typy złożone które skladają się z typów prostych. Nauczysz się jak tworzyć typy złożone w innym materiale, przykladowo typ "Punkt" będzie się składał z dwóch liczb całkowitych (x i y).

ⓘ **Pamiętaj**

Każda zmienna, niezaleznie od typu jest reprezentowana przed kod bitowy(zera i jedynki). Po kompilacji (przetłumaczeniu na kod zrozumiały dla komputera), typ nie ma znaczenia, pozostają tylko zera i jedynki. Kompilator (czyli tłumacz) pilnuje dla nas czy używamy zmiennej danego typu w prawidłowy sposób.
Jest to ważne dlatego że na różnych typach danych wykonujemy różne operacje, nie wykonamy przecież mnożenia na tekście("Dzisiaj jest piątek" * 5), bo jaki to ma sens?

Typy liczb całkowitych (byte, short, int, long)

Mamy kilka typów podstawowych dla przechowywania liczb całkowitych. Różnią się one minimalną i maksymalną wartością jaką mogą przyjmować. Różnice w zakresie wpływają na to ile pamięci zostanie zarezerwowane - jak duże pudełko zarezerwujemy.

```
public class Main {
    public static void main(String[] args) {
        System.out.println("byte\t" + Byte.SIZE/8 + " bytes\t" +
                Byte.MIN_VALUE + " to " + Byte.MAX_VALUE);
        System.out.println("short\t" + Short.SIZE/8 + " bytes\t" +
                Short.MIN_VALUE + " to " + Short.MAX_VALUE);
        System.out.println("int\t\t" + Integer.SIZE/8 + " bytes\t" +
                Integer.MIN_VALUE + " to " + Integer.MAX_VALUE);
        System.out.println("long\t" + Long.SIZE/8 + " bytes\t" +
                Long.MIN_VALUE + " to " + Long.MAX_VALUE);
    }
}
```

Powyższy program wypisze rozmiar jaki zajmują te typy oraz ich zakres, wypróbuj go!

ⓘ **Bity i Bajty**

Bit to pojedynczy znak - 0 lub 1. Bajt składa się z 8 bitów.
SIZE mówi nam ile bitów zajmuję dana zmienna więc jeżeli chcemy uzyskać rozmiar w bajtach, dzielimy przez 8.

⚠ **Typy opakowujące**

W powyższym programie, typ piszemy z dużej i do tego zamiast Int to piszemy Integer.
Java posiada typy opakowujące dla typów prostych które dostarczają dodatkowych funkcjonalności. Z racji, że mogą więcej to też więcej zajmują miejsca w pamięci, dlatego używaj ich tylko wtedy, kiedy potrzebujesz tych dodatkowych funkcjonalności.

Tak stworzysz zmienne tych typów

```
byte b = 1;
short age = 25;
int something = 11;
long big = 1_000_000L;
```

Pomiędzy cyframi możemy dodać znak '_', nie zmienia on nic ale poprawia czytelność. Możemy też powiedzieć kompilatorowi że dana liczba ma być koniecznie typu **long** dodając małą lub dużą literę **L** na końcu.

Którego typu użyć?

Najczęściej używanym jest - int, reszta jest do "zadań specjalnych".
byte -> najczęściej spotkasz w pracy z plikami, obrazami.
short -> w sumie to nie wiem, nigdy go na poważnie nie użyłem, lol.
long -> najczęściej do przechowywania klucza głównego z bazy danych.
int -> cała reszta.

Typy zmienno przecinkowe (float, double)

Analogicznie do typów całkowitych, mamy kilka typów do przechowywania liczb zmienno przecinkowych (w sensie takich z przecinkiem), a dokładnie to dwa.

Program do wyświetlenia rozmiaru i zakresu

```
public class Main {
    public static void main(String[] args) {
        System.out.println("float\t" + Float.SIZE/8 + " bytes\t" +
                Float.MIN_VALUE + " to " + Float.MAX_VALUE);
        System.out.println("double\t" + Double.SIZE/8 + " bytes\t" +
```

```
                Double.MIN_VALUE + " to " + Double.MAX_VALUE);
    }
}
```

Tworzenie zmiennych

```
float pi = 3.14f;
double idk = 15.45;
```

Dodając małą lub dużą literę **F** na końcu liczby, mówimy kompilatorowi że jest to liczba typu **float**.

⚡ Problem precyzji

Liczby zmiennoprzecinkowe są niedokładne w programowaniu. Oznacza to że, wykonanie działania 0.1 + 0.1, nie jest dokładnie równe 0.2.

```
        float x = 0.1f + 0.1f;
        System.out.printf("%.20f", x); // %.20f = wyświetl liczbę zmienno
przecinkową z 20 cyframi po przecinku.
```

Powyższy program wyświetlił u mnie wynik

```
0.20000000298023224000
```

Jeżeli nie zależy nam na dokładnej precyzji to nie musimy się tym przejmować. Zatem do przechowywania i obliczania pieniędzy ten typ się nie nadaje ;)

Kiedy to ma znaczenie
Dlaczego tak się dzieje

Typ boolean (true or false)

Typ boolean jest dość prosty, przechowuje wartość prawdy lub fałszu, tyle i aż tyle. Często właśnie potrzebujemy wiedzieć czy coś jest prawdziwe lub fałszywe, czy użytkownik nacisnął odpowiedni klawisz, czy pole na hasło jest puste, czy przycisk w programie został wciśnięty. To wszystko możemy przechować jako wartość prawda lub fałsz (true/false).

Tworzenie takiej zmiennej

```
boolean prawda = true;
boolean falsz = false;
```

> ⓘ **Tylko tyle?**
>
> W następnym materiale nauczysz się o operacjach arytmetycznych i logicznych. Wówczas stanie się to znacznie ciekawsze ;)

Typ znakowy (char)

Do przechowywania **pojedynczego** znaku służy typ **char**.

Tworzenie zmiennej

```
char sign = 'p';
```

Pojedyncze znaki zapisujemy pomiędzy apostrofami, wyznaczają one początek i koniec znaku (pojedynczy znak nie musi się składać z pojedynczego symoblu).
Zatem jak zapisać sam apostrof?

```
char blad = '''; // błąd!
```

Powyższy zapis nie zadziała ponieważ środkowy znak będzie oznaczał zakończenie znaku, a ostatni apostrof po prostu będzie błędem.
Aby zapisać apostrof musimy użyć znaku specjalnego, znaki specjalne są poprzedzane backslash'em. W typ przypadku znak specjalny zwalnia apostrofa ze swojej funkcji.

```
char poprawnie = '\''; // zmienna poprawnie przechowa nam znak pojedynczego
apostrofa, bez znaku \
```

Najczęściej wykorzystywane znaki specjalne.

Znak specjalny	Znaczenie
\`	Apostrof
\"	Cudzysłów
\\	Backslash
\t	Tabulator

Znak specjalny	Znaczenie
\n	Nowa linia (enter)

Nie są to wszystkie dostępne, jeżeli chcesz dowiedzieć się więcej po prostu wygooglaj frazę *java znaki specjalne*.

> ⓘ No, a jak zapisać całe słowo, tekst, wiersz?
>
> Do zapisywania całych tekstów służy *typ złożony* String, który sobie teraz po krótce omówimy.

Typ złożony String

String oznacza łańcuch znaków i jest w pewien sposób wyjątkowy.

Każdy typ prosty, ma swój odpowiednik typu opakowującego i możemy z niego korzystać jak z typu prostego.

```
int x = 5;
Integer y = 6;
```

Co nie jest normalne. Typy złożone są tworzne za pomocą operatora **new**.

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

Tylko do typów opakowujących możemy bezpośrednio przypisać wartość... i do Stringów.

```
String ala = "Ala ma kota, i kot ma Ale.";
```

Typy opakowujące i Stringi możemy również tworzyć za pomocą operatora **new**

```
Integer x = new Integer(5);
String ala = new String("Ala ma kota, i kot ma Ale.");
```

> ✓ Ciekawostka | String Pool
>
> Do Stringów przypisujemy bezpośrednio wartość zamiast korzystać z operatora **new**. Jeżeli korzystając z bezpośredniego przypisania, stworzymy dwie zmienne które będą

miały taką samą wartość (np. "Ala") to obie zmienne będą odwoływały się do tej samej komórki pamięci.
...co?
Będziesz miał jedno pudełko z wartością "Ala", ale dostaniesz dwa klucze dostępu do tego samego pudełka. To tak jak kilka osób ma klucze do tego samego mieszkania. Dzięki temu duplikaty są tworzne tylko raz i nie zajmują nie potrzebnie miejsca.

Jak już zauważyłeś/aś, ciąg znaków zapisujemy pomiędzy cudzysłowy.

```
String slow = "Mass slow";
```

Typ String jest tak na prawdę tablicą znaków **char**, czyli zawiera wiele, uporządkowanych, pojedynczych znaków **char**.

ⓘ **Dobrze wiedzieć | Znak końca tekstu**

Aby program mógł wiedzieć gdzie kończy się tekst, a gdzie zaczyna całkiem inna zmienna to na końcu tekstu jest dodawany znak specjalny **\0** który oznacza koniec tekstu. W Javie nie zauważysz go wypisując tekst, ani nie zobaczysz tego wypisując długość Stringa za pomocą **length**.

```
String ala = "Ala ma kota";
System.out.println(ala.length());
```

Do tego problemu wrócę przy omawianiu tablic.

Rzutowanie

Ostatnim zagadnieniem na dziś jest rzutowanie, czyli konwersja typu na inny.

Aby zmienić typ na inny to przed jego użyciem należy go poprzedzić typem na który chcemy rzutować zamkniętym w nawiasy okrągłe.

```
int x = 5;
short y = (short)x; // W sensie że tak
```

Ta zasada dotyczy typów prostych i złożony... ale nie możemy rzutować wszystkiego na wszystko inne, są pewne zasady. Typami złożonymi zajmiemy się kiedy indziej.

Rzutować możemy między typami całkowitymi, typami zmienno przecinkowymi i całkowitymi na zmienno przecinkowe i odwrotnie.

> ⚡ **Konwersja stratna**
>
> Jeżeli rzutujemy z typu o większym zakresie na mniejszy to musimy liczyć się z tym że jeżeli wartość przekracza zakres typu docelowego to utracimy naszą wartość. Po prostu mniejszy typ nie pomieści większej wartości i w rezultacie wyjdą tzw. śmieci, czyli wartości które nie mają żadnego znaczenia.
> Natomiast gdy chcemy rzutować typ zmienno przecinkowy na typ całkowity to musimy liczyć się z tym że utracimy informacje o precyzji, czyli wartość po przecinku.

Przykłady

```
long big = 3_000_000_000L;
int small = (int)big; // -1294967296 <- śmieci

float pi = 3.14f;
int three = (int)pi; // 3

char c = 'a';
int code = (int)c; // 97
```

Pojedyncze znaki są reprezentowane przez liczbę - kod znaku. Możemy go odczytać rzutując **char** na **int**.
Kod znaku nie jest przypadkowy, możemy odczytać go z tablicy ASCII.

> ⓘ **I am sorry**
>
> ASCII opisuje znaki do wartości 255 (1 bajt), nie zawierają one znaków specjalnych (ą, ę, ł, itd.), natomiast możemy przypisać **'ą'** do **char** i dostaniemy wartość 261. Typ **char** w Javie ma rozmiar 2 bajtów, czyli może przechować 65 535 unikalnych wartości. Prawdodobnie Java korzysta z kodów Unicode ale po prostu nie wiem. I am sorry.

Podsumowanie

W tym materiale przyswoiłeś/aś solidną dawkę wiedzy i dowiedziałeś/aś się o fundamentach programowania. Bez zmiennych programowanie nie miałby sensu, nie

mielibyśmy z czego lepić.

Zadanka

Zadanka do tego materiału mają gównie wymusić na Tobie sprawdzenie czegoś samodzielnie. W ten sposób najefektywniej przyswoisz wiedze.