Java - to jeden z najpopularniejszy języków programowania wśród programistów.

Język programowania - służy do tworzenia programów, aplikacji czyli instrukcji które mogą zostać przetworzone i zrozumiane przez komputer.

Komputer używa języka, który jest dla nas niezrozumiały i ciężki do odczytania np.

```
01010101111011101
```

Z pomocą **języka programowania** możemy napisać coś co zwiemy instrukcją, która będzie zrozumiana przez nas, a później również przez komputer.

```
int b;
```

Powyższa instrukcja oznacza: zarezerwuj zmienną o typie int w pamięci RAM. Jest to instrukcja, która jest później przetwarzana na postać zer i jedynek w stylu:

```
01010101010
```

01010101010 to **impulsy elektryczne**

To **tzw. język (kod) maszynowy** do którego chcemy przetworzyć nasz język programowania.

Proces zmiany z

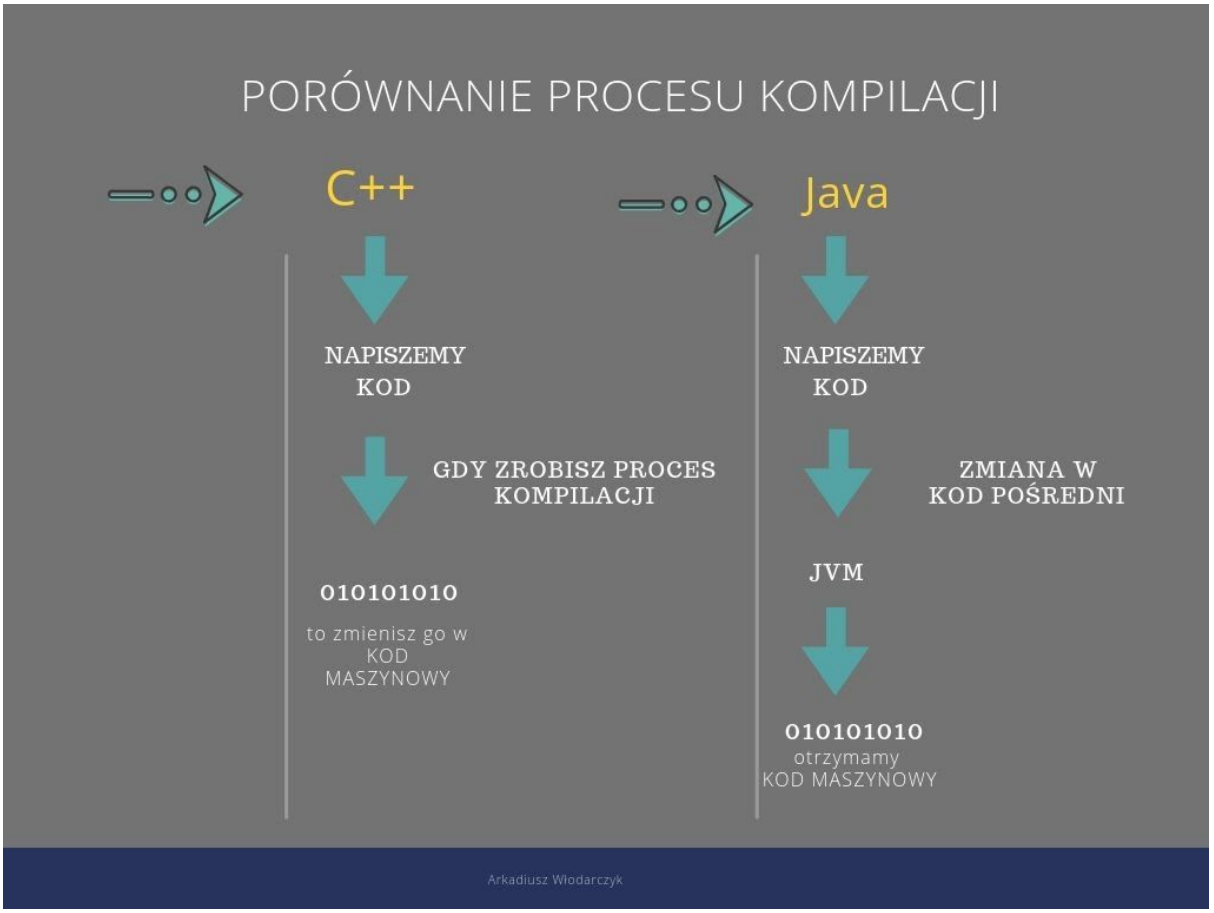
```
int b:   →   01010101010
```

to proces **kompilacji**, czyli przetwarzanie kodu z tego zrozumianego przez nas na kod rozumiany przez komputer.

Proces kompilacji nie w każdym języku jest taki sam, np. w języku C++ gdy zrobimy proces kompilacji na oprogramowaniu Windows 8 to musimy wykonać ponowną kompilację np. na Linuxie, ponieważ proces ten przystosowuje do danego procesora instrukcje, aby zostały zinterpretowane.

Natomiast język Java zmienia napisany kod w kod pośredni, który jest rozumiany przez **JVM**.

JVM - czyli Java Virtual Machine - wirtualna maszyna Javy. Program który zmienia w czasie rzeczywistym kod pośredni w na kod maszynowy, czyli 010101010.



JRE - Java Runtime Environment czyli środowisko do odpalania programów Javy. Ma w sobie m.in JVM i inne biblioteki, które można zainstalować na różnych oprogramowaniach (Windows, Linux, Android)

IDE - z ang. **I**ntegrated **D**evelopment **E**nvironment - zintegrowane środowisko do programowania.

Zalety IDE :

- cały proces **kompilacji**, czyli przetworzenia kodu napisanego przez nas w kod pośredni i uruchomieniu go przez JVM jesteśmy w stanie zrobić jednym przyciskiem:

   (zielona strzałka w pasku głównym programu)

- edytor wykrywa na bieżąco ewentualne błędy, zaznacza linię w której wykrył błąd za pomocą czerwonego wykrzyknika

   po lewej stronie przy numerze linii w której wykrył błąd

Komentarze - służą do dodania dodatkowej informacji o kodzie. Pozwalają zaoszczędzić dużo czasu np.:
a) gdy po pewnym czasie wracamy do naszego kodu - nie musimy zastanawiać się w jaki sposób będzie działał nasz kod
b) gdy nad programem pracuje więcej niż jedna osoba nie będzie musiała się domyślać co autor w danym momencie miał na myśli.

Jeżeli chcemy dodać komentarz :

- wielolinijkowy (slash a następnie gwiazdka)

  / *

  ***/**

- jednolinijkowy (za pomocą dwóch slashy)

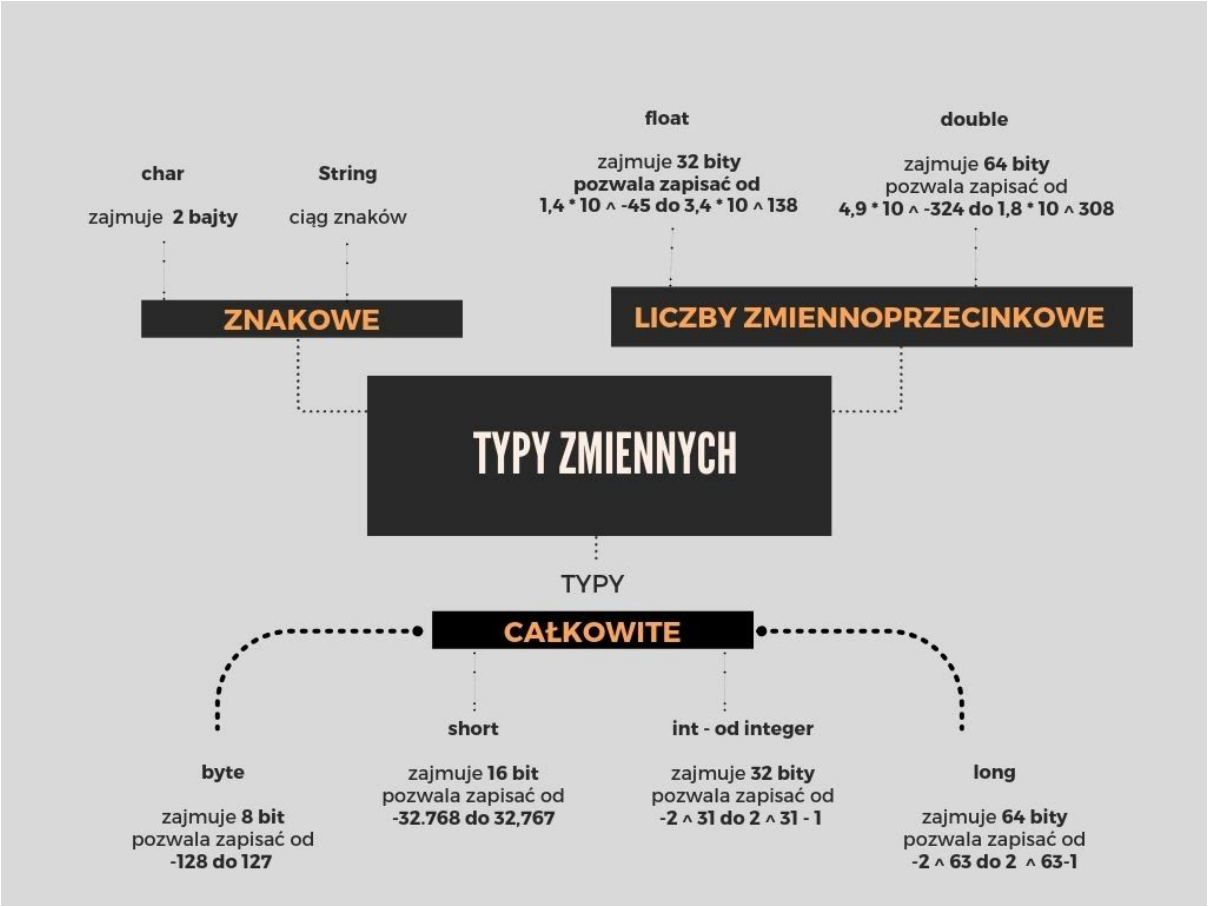
  //

Podstawy, które musisz znać:

- warto wstawiać białe znaki, czyli tzw. wcięcia, tabulatory, entery - dla lepszej przejrzystości kodu i ułatwienia odczytywania go
- każdy rozkaz powinien się kończyć średnikiem ;
- klamry { } mówią jaki jest zakres działania
- class - klasa - pojemnik do przechowywania informacji oraz funkcji, które chcemy opisać znajdują się w { } klamrach
- każdy program będzie rozpoczynał się od wywołania instrukcji, wewnątrz funkcji main

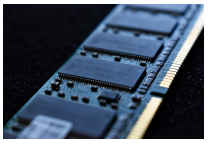
system.out.println(“TO JEST TEKST”);

system - odwołujemy się do systemu
out - wyjście
print - drukuj (funkcja)
ln - dodawanie lini, enter
(“TO JEST TEKST”); tekst, który chcemy wydrukować



Zmienna tj. pojemnik do którego możesz zapisać daną wartość. Każdy z tych pojemników może przechowywać różnego rodzaju elementy np. liczby, tekst. Zmienne przechowywane są w **pamięci RAM.**

RAM (od ang. random-access memory) - czyli tymczasowa pamięć do przechowywania danych



Charakterystyka zmiennych :

- ❑ **TYP CAŁKOWITY :**

- **int - integer** (przechowuje liczby całkowite) np.

```
int a = 10;
```

- **byte** do 8 bitów np.

```
byte liczba = 127;
```

- **short** do 16 bitów np.

```
short liczba = 128;
```

- **long** do 64 bitów np.
  long a = 123123123;

❑ **TYPY ZNAKOWE :**

- **char** - 2 bajtowy (**tylko jeden znak, piszemy w apostrofie)**

```
char znak = 'A';
```

- **String - ciąg znaków (piszemy z dużej litery String w “ “ cudzysłowach)**

```
String imie = "Arkadiusz";
```

Należy pamiętać, że nie można dodawać zmiennych, które mają różne typy np. zmiennej typu String do zmiennej typu int. Możemy tylko dodawać zmienne o tym samym typie.

<u>Jak dodać zmienne na przykładzie Stringa?</u>

```
String imie = “Arek”;
String nazwisko = “Włodarczyk”;

system.out.println(imie  + “ “ +  nazwisko);
```

“ “ = doda spację między imieniem a nazwiskiem

❑ **TYPY ZMIENNOPRZECINKOWE :**

to takie, które mają wartość dziesiętną. Do ich wypisania możemy użyć:

- **float**

  float liczba = 4.67**f**;

  musimy na końcu zaznaczyć, że chcemy użyć typu float poprzez użycie na końcu liczby literki **f**

- **double**

  double liczba = 53.6435;

  typ ten charakteryzuje się dużo większą precyzją niż float

Aby zarezerwować miejsce w pamięci RAM i wydać rozkaz :

1. Piszemy **typ zmiennej -** mówi on co będziemy przechowywać w zmiennej. Najpopularniejszym typem jest :
   int z ang integer - l. całkowita (który pozwoli przechowywać liczby całkowite)

   ```
   int
   ```

2. **Nazwa zmiennej**, która powinna być samoopisująca się.

   ```
   int a
   ```

3. **Operacja przypisania** za pomocą **operatora,** który operuje na zmiennej, czyli przypisuje nową **wartość**

   ```
   int a = 10;
   ```

Zmiennym w każdej chwili możemy zmienić wartość. Możemy np. zmiennej a przypisać wartość 5 np.

```
a = 5;
```

lub stworzyć kolejne zmienne np.

```
int a = 10;
int b = 6;
int c;

c = a + b;
```

Aby wypisać powyższe zmienne piszemy **sout,** a następnie klikamy przycisk tab - **tabulator** a nasz edytor automatycznie uzupełni funkcje **println**.

Jak poprawnie nazywać zmienne ?

➔ **Każda zmienna powinna się zaczynać od małej litery np.**

```
String imie = "Wiola";
```

Jeżeli chcemy zmienić wartość zmiennej nie korzystamy już z nazwy typu np. String.

```
imie = " Arek";
```

➔ **Pierwsza litera zawsze powinna być mała natomiast druga wielka**, ułatwi to odczytywanie naszego kodu oraz będzie wyglądał profesjonalnie np.

```
daneOsobowe = imie + " " + nazwisko;
```

➔ **Zmienne powinny być samoopisujące się tzn.** jeśli chcemy wypisać imię i nazwisko użyjmy nazwy np. `daneOsobowe`

➔ **Można rozpocząć zmienną podkreśleniem np.**

```
int _cos_ff;
```

➔ **Znak podkreślenia** (_) zarezerwowany jest i powinniśmy go używać przy tworzeniu stałych.

```
final double LICZBA_PI = 3,14;
```

STAŁE - to zmienne finalne, nie jesteśmy w stanie zmienić podanej w nich wartości.

NIE WOLNO :

- używać jako etykiety keywordów, czyli słów, kluczy które są już zarezerwowane np. public
- zaczynać nazwy zmiennej od liczb
- korzystać ze spacji w nazwie zmiennej

Rzutowanie - zmiana typu zmiennej z jednego na drugi.

```
int a = 5 , b = 2;
double c = 12, d = 15;

system.out.println(a/b)
```

W powyższym przykładzie dzielimy przez siebie dwie liczby całkowite, więc nasz wynik również będzie liczbą całkowitą - w tym przypadku 2.

Jeżeli chcemy, aby pojawiła się liczba z ułamkiem, czyli liczba zmiennoprzecinkowa musimy wykonać **rzutowanie** - czyli, jedną z podanych zmiennych zmienić na typ double. Nie ma znaczenia którą zmienną będziemy rzutować - wynik będzie ten sam.

```
system.out.println((double)a/b);
```

otrzymany wynik to 2.5.

Jeżeli chcemy podzielić

```
int wynik1 = a/d
```

czyli liczbę całkowitą (a) przez liczbę zmiennoprzecinkową (d) po odpaleniu pojawi się błąd, ponieważ kompilator chce nas ostrzec, że dzielimy całkowitą przez ułamek i przypisujemy to pod typ integer.

Jeżeli chcemy aby nasz wynik został wywołany - musimy pokazać kompilatorowi, że rozumiemy jaka operacja zostanie wywołana. A więc wykonujemy rzutowanie (d) do int`a i od tego momentu możemy wywołać kod. Jeżeli zapiszemy do inta stracimy ułamek i otrzymamy wynik 0.

```
int wynik1 = a/(int)d;
```

W innym przypadku np. dzielenia liczby całkowitej przez liczbę z ułamkiem w typie double

```
double wynik2 = a / d;
```

po wywołaniu

```
system.out.println(wynik2);
```

kod zostanie wywołany, ponieważ gdy mamy przynajmniej jedną liczbę, która jest zmiennoprzecinkową przy dzieleniu to nasz kod zadziała poprawnie.

Jeżeli będziemy chcieli podzielić liczby stałe np. ⅕

```
system.out.println(1/5);
```

to wynik jest równy 0. Aby wynik był poprawny musimy wykonać rzutowanie bądź zastosować formę 1.0 / 5

```
system.out.println(1.0/5);
```

Operatory - to znaki służące do operowania na zmiennych.

OPERATOR	ROLA OPERATORA
+	dodawanie liczb, łączenie stringów
-	odejmowanie
*****	mnożenie
/	dzielenie
=	przypisanie wartości
+=	a = a + 2 to inaczej a+=2
-=	a = a - 7 to inaczej a -= 7
/=	a = a / 4 to inaczej a /= 4
***=**	a = a * 3 to inaczej a *= 3
%	reszta z dzielenia (dzielenie modulo)
++	INKREMENTACJA (powiększ o 1)
--	DEKREMENTACJA (zmniejsz o 1)
y++	POST inkrementacja
y--	POST dekrementacja
++y	PRE inkrementacja
--y	PRE dekrementacja

Przykłady wykorzystania operatorów arytmetycznych :

- **= przypisanie wartości**

```
int a = 3
```

- **% - reszta z dzielenia (dzielenie modulo)**

```
int a = 1 % 3;
```

wynik to 1, ponieważ gdy podzielimy 1/3 to mamy ⅓ więc wynikiem jest część z ułamka która pozostała w liczniku

```
int a = 3 % 3
```

(brak reszty - wynik 0)

- **++ INKREMENTACJA (powiększ o 1)**

```
int a = 1;
a++;
System.out.println(a);
```

Od tego momentu a zostanie powiększone o 1

- **-- DEKREMENTACJA (zmniejsz o 1)**

```
int a = 2;
a--;
System.out.println(a);
```

Od tego momentu a zostanie pomniejszone o 1

- **a++ POST inkrementacja** , czyli wykona dodawanie, gdy wypiszesz na ekran wartość, która znajdowała się pod daną zmienną

```
a = 5,
System.out.println(a++);
System.out.println(a);
```

- **a-- POST dekrementacja,** czyli wykona odejmowanie, gdy wypiszesz na ekran wartość, która znajdowała się pod daną zmienną

```
a = 5,
System.out.println(a--);
System.out.println(a);
```

- **++a PRE inkrementacja** - pierw powiększy podaną zmienną, następnie wypisze jej wartość

```
System.out.println(++a);
```

- **--a PRE dekrementacja -** pierw pomniejszy podaną zmienną, następnie wypisze jej wartość

```
System.out.println(--a);
```

Operatory bitowe - operują na bitach, podstawowych jednostkach informacji przechowywanych w komputerze za pomocą impulsów elektrycznych (01010)

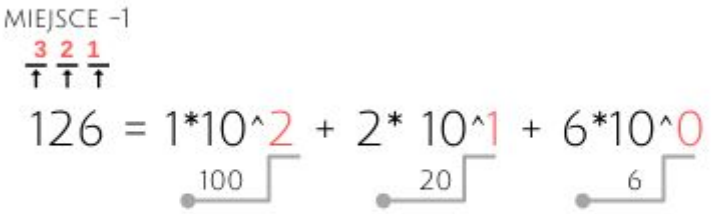
OPERATORY BITOWE

0 - false
1 - true

OPERATORY BITOWE	ROLA OPERATORA
&	iloczyn bitowy
|	suma bitowa
^	XOR e**X**clusive**OR**
x << 1	przesunięcie w lewo o 1
x >> 2	przesunięcie w prawo o 2
~	negacja bitowa

Wyróżniamy

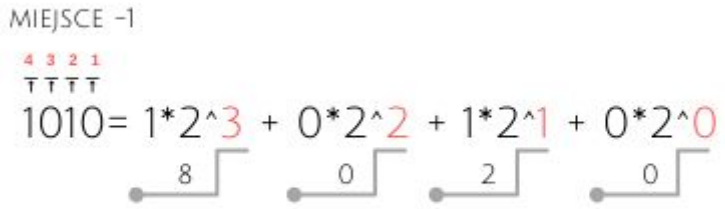
- ❏ system dziesiętny (od 0 do 9)



Mnożymy pierwszą liczbę (1) przez system dzięsietny, czyli 10 następnie podnosimy go do potęgi (podajemy miejsce liczby). Przy podaniu **miejsca** odejmujemy 1 czyli jeśli liczba występuje na pozycji nr 3 odejmujemy od niej 1 w tym momencie jej pozycja to 2. Kolejne liczby obliczamy identycznie zgodnie z regułą.

- ❏ system dwójkowy (od 0 do 2)

Przeliczamy liczbę 1010 systemem dwójkowym, aby sprawdzić jaka to liczba w systemie dziesiętnym



8 + 0 + 2 + 0 = 10

W systemie dwójkowym zapis **1010** to w postaci dziesiętnej **liczba 10.**

Aby ułatwić sobie sposób przeliczania liczb na system dziesiętny

miejsce	4	3	2	1
miejsce -1	**3**	**2**	**1**	**0**
	1	0	1	0

- ❏ Jeżeli występuje 0 - pomijamy tą wartość
- ❏ Miejsce każdej liczby 1 symbolizuje liczbę dwa (2) podniesioną do potęgi miejsca w którym się znajduję -1 czyli np.

1 0 **1** 0 to inaczej

2 ^ 3 = 8
0 ^ 2 = 0
2 ^ 1 = 2
0 ^ 0 = 0

8 + 2 = 10

Ostatnia liczba symbolizuje czy dana wartość jest parzysta bądź nie
np. 1010

- jeśli na końcu występuje **1 - nieparzysta**
- jeśli na końcu występuje **0 - parzysta**

Operacje bitowe :

★ **iloczyn bitowy -** operator ten działa jak koniunkcja, różnicą jest to że nie operuje na prawdzie i fałszu lecz na zerach i jedynkach. Koniunkcja jest prawdziwa wtedy, gdy oba warunki są spełnione.

1 1 1 0 // 14
1 0 1 1 // 11
---
1 0 1 0
8 + 2 = 10

```
System.out.println(14 & 11);
```

Otrzymany wynik to 10.

★ **suma bitowa -** działa jak alternatywa. Alternatywa jest fałszywa tylko wtedy, gdy oba argumenty są fałszywe.

1 0 1 0 // 10
1 0 1 1 // 11
---
1 0 1 1
8 + 2 + 1 = 11

```
System.out.println(10 | 11);
```

Otrzymany wynik to 11.

★ **XOR eXclusive OR (albo) -** gdy obie wartości są prawdziwe lub fałszywe to wynik jest fałszywy.

1 1 - 0 wartość fałszywa
0 0 - 0 wartość fałszywa
1 0 - 1 wartość prawdziwa
0 1 - 1 wartość prawdziwa

1 0 1 0 // 10
1 0 1 1 // 11
---
0 0 0 1 // 1

```
System.out.println(10 ^ 11);
```

Otrzymany wynik to 1.

XOR gdy ma podane te same wartości to zawsze zwróci 0, np.

```
System.out.println(5 ^ 5);
```

Natomiast, gdy już wartości będą się różnić zwróci liczby różne od 0.

★ Przesunięcie w prawo

Gdy napiszemy np.

0 00 000 1010 i przesuniemy w prawo o 1 otrzymamy :
0 00 00 0101

4 +1 = 5

```
System.out.println(10 >> 1);
```

Wynik otrzymany to 5, nastąpiła **operacja dzielenia** przez 2.

★ Przesunięcie w lewo

```
System.out.println(10 << 1);
```

Przemnożyliśmy 10 * 2
Nasz wynik to 20.

`1 -` liczba (*2) która mówi do której potęgi mamy podnieść liczbę, która znajduje się przed strzałkami

★ Operator negacji - zmienia wartość

1 0 1 0 **zmienia na**
0 1 0 1

```
System.out.println(~10);
```

Otrzymany wynik to -11

-

Operatory logiczne - operują na logicznych wartościach, czyli **true** - prawda i **false** - fałsz

<u>Podstawowe operatory:</u>

- ❏ **!** - negacja (zaprzeczenie)
- ❏ **!(true)** - zmieni prawdę na **false** (fałsz) np.

```
int  a = 5,
     b = 5,
     c = 7;
if (!(a==b))
   System.out.println("tak")
```

- ❏ **!(false)** - zmieni fałsz na **true (**prawda)
- ❏ **&&** tłumaczone jako `i` to tzw. **koniunkcja,** czyli połączenie dwóch warunków w jednym miejscu i sprawdzenie czy oba warunki zaszły jednocześnie. Koniunkcja jest prawdziwa wtedy gdy oba wyrażenia są prawdziwe - tylko wtedy zostanie wywołana instrukcja.

  Koniunkcja działa według schematu :

  true true - true
  true false - false
  false true - false
  false false - false

  Przykład koniunkcji

```
int a = 5,
    b = 7,
    c = 7;
if (a < b && c == 7)
   System.out.println("tak")
```

- || **Alternatywa -** tłumaczona jako **“lub”**. Alternatywa jest prawdziwa, gdy co najmniej jeden z warunków jest spełniony.

Alternatywa działa według schematu :

true true - true
true false - true
false true - true
false false - false

Alternatywa jest fałszywa tylko wtedy, gdy oba wyrażenia są fałszywe.

Przykład alternatywy

```
int a = 5,
b = 7,
c = 7;
if (a > b || c == 7)
System.out.println("tak")
```

W powyższym przykładzie **jeden z warunków został spełniony**, ponieważ c jest równe 7. Nasza alternatywa jest prawdziwa.

Gdy wykonywaliśmy porównanie np.

```
int a = 5,
    b = 5;
if (a == b)
    system.out.println(“tak”)
```

wykonana była jedna instrukcja, czyli warunek if. Natomiast, gdy chcielibyśmy wprowadzić np. zmienną c

```
int a = 5,
    b = 5;
    c = 7;
```

Operatory relacyjne (porównania) - to tzw. operator do warunkowania, służy do sprawdzania np. czy podane zmienne są np. równe, większe mniejsze itp.

Wyróżniamy operatory:

- **> większe od**
- **< mniejsze od**
- **>= większe bądź równe**
- **<= mniejsze bądź równe**
- **== równe**
- **!= nierówne**

Przykładowe zastosowanie operatorów

```
int a = 5,
    b = 7;

System.out.println(a > b);

System.out.println(a < b);

System.out.println(a != b);
```

Boolean czyli wartość, która może przechowywać tylko **prawdę (true)** lub **fałsz (false),** korzystamy z niej, gdy wykonujemy jakieś porównania

```
boolean isTrue = 4 == 4;
System.out.println(isTrue);
```

w powyższym przykładzie 4 jej równe 4,czyli została zwrócona nam prawda (true)

```
boolean isTrue = 4 == 8;
System.out.println(isTrue);
```

Podana liczba 4 nie jest równa 8, więc wynik zwróci nam fałsz (false)

Z **boolean** korzystamy też podczas **tworzenia warunków.**

if (z. ang) - jeśli

```
boolean isTrue = 4 == 8;
if (4 == 8)
   System.out.println(isTrue);
```

instrukcja ta nie zostanie wywołana, ponieważ 4 nie jest równe 8.

Jeżeli mielibyśmy przykład 4 == 4 instrukcja ta zostałaby wywołana, ponieważ zostałby spełniony warunek (if)

switch (przełącznik) - pozwala przełączać pomiędzy różnymi stanami wartość, która została do niego przesłana.

```
int a = 150;
switch (a)
{
 case 50:
        System.out.println("a jest równe 50");
        break;
 case 100:
        System.out.println("a jest równe 100");
        break;
 default:
     System.out.println("a nie jest równe 50 ani 100 a jest równe" + a);
}
```

case - przypadek
break - to całkowite wyjście z przełącznika

Instrukcje warunkowe - to forma sprawdzenia czy podana instrukcja jest prawdziwa lub fałszywa.

Typ boolean - to wartość , która może przechowywać tylko prawdę (true) lub fałsz(false).

Jeżeli wyrażenie będzie prawdziwe mamy możliwość dzięki warunkowaniu wykonać podaną instrukcję. Jeżeli będzie fałszywe nasza instrukcja nie zostanie wykonana.

Schemat tworzenia instrukcji warunkowych:

```
if (WYRAŻENIE)
   instrukcja;
else if (WYRAŻENIE)
        instrukcja;
else
    instrukcja;
```

Przykład instrukcji warunkowej :

- **if - jeżeli**

```
int a = 6,
    b = 6;
if (a == b)
   System.out.println("Wartości są równe");
```

Instrukcja została wykonana, ponieważ warunek (if) został spełniony.

```
int a = 5,
    b = 6;
if (a > b)
   System.out.println("Wartości są równe");
```

Instrukcja ta nie została wykonana, ponieważ warunek (a > b) jest fałszywy.

- **else** możemy połączyć z **if** tzw. **else if** (w innym wypadku jeżeli)

```
int a = 5,
    b = 5;
if (a > b)
   System.out.println("a > b");
else if (a < b)
   System.out.println("a < b");
```

- **else** w całkowicie innym wypadku..

```
int a = 5,
    b = 5;
if (a > b)
   System.out.println("a > b");
else if (a < b)
   System.out.println("a < b");
else
   System.out.println("a == b");
```

Pod instrukcją warunkową, możemy napisać jedną instrukcję. Jeżeli chcemy wykorzystać ich więcej i traktować je jako jedną instrukcję to musimy skorzystać z nawiasów { }, które określają jej zasięg, grupę.

```
int a = 2,
    b = 5;
if (a > b)
   System.out.println("a > b");
else if (a < b)
{
   System.out.println("a < b");
   System.out.println("Program działa");
}
else
   System.out.println("a == b");
```

Jeżeli chcemy sprawdzić, czy dana liczba jest parzysta korzystamy z instrukcji warunkowych oraz dzielenia modulo % np.

```
int x = 5;

if (x % 2 == 0)
   System.out.println("Liczba jest parzysta");
else
   System.out.println("Liczba jest nieparzysta");
```

aby skrócić i ułatwić sobie pracę możemy również zapisać powyższy kod dużo krócej, za pomocą **wyrażenia warunkowego** np.

```
String czyParzysta = x % 2 == 0 ? "parzysta" : "nieparzysta";
System.out.println(czyParzysta)
```

Sposób tworzenia wyrażenia warunkowego:

wyrażenie ? co ma się stać jeśli wyrażenie jest prawdziwe : co ma się stać, gdy wyrażenie jest fałszywe ;

Tablica - to kontener dla zmiennych ułożonych obok siebie. Używamy jej, gdy chcemy stworzyć więcej zmiennych, które są tego samego typu.

Sposób tworzenia tablicy

int[ ] tab;

int - typ zmiennych
[ ] - kwadratowe nawiasy - informują, że to tablica
tab - nazwa tablicy

tab = new int [5];

tab - odwołujemy się do nazwy tablicy
new - rezerwujemy miejsce w tablicy za pomocą słowa new w podanym typie
int [5] podajemy ile mamy zarezerwowanego miejsca dla tego typu

Aby odwołać się do miejsc w tablicy korzystamy z indeksów.

```
System.out.println(tab[1]);
```

piszemy **nazwę tablicy** oraz indeks do którego chcemy się odwołać.

tab = new int [5];

`tab[0]` `tab [1]` `tab[2]` `tab[3]` **`tab[4]`**

Ostatnia tablica zawsze ma indeks o jeden mniejszy niż **nr tablicy,** ponieważ **numerowanie indeksów zaczynamy od 0.**
Do pierwszej tablicy odwołujemy się za pomocą indeksu 0.

Możemy również stworzyć tablicę w inny sposób np.

- ❏ w jednej linii

```
int[] tab = new int
deklaracja  przypisanie w pamięci miejsca
```

- ❏ stworzyć i przypisać odrazu wartości tablicy

```
int[] tab2 = {4, 14, 2, 12}
```

- **Aby odwołać się np. do pierwszej liczby w tablicy**

```
System.out.println(tab2[0]);
```

- **Aby pobrać długość tablicy**

```
System.out.println(tab2.length);
```

- **Aby odwołać się do ostatniego elementu tablicy**

```
System.out.println(tab2[tab2.length-1]);
```

Tablica wielowymiarowa - składa się z wierszy i kolumn. Tworzymy ją za pomocą dwóch [ ] [ ] klamer kwadratowych. Tablicę numerujemy od **0**. Pierwszy argument to **ilość wierszy**, drugi to **ilość kolumn** np.

```
int[][] tab = new int [4][3];
tab[0][1] = 25;
```

Odwołanie do wiersza 0, kolumny 1 to:

```
System.out.println(tab[0][1]);
```

Tworzenie tabeli wielowymiarowej

Kolumna: 1 2 3

```
-------------------------------------------------------
tab[0][0] | tab [0][1] | tab[0][2] |
-------------------------------------------------------
-------------------------------------------------------
tab[1][0] | tab [1][1] | tab[1][2] |
-------------------------------------------------------
-------------------------------------------------------
tab[2][0] | tab [2][1] | tab[2][2] |
-------------------------------------------------------
-------------------------------------------------------
tab[3][0] | tab [3][1] | tab[3][2] |
-------------------------------------------------------
```

```
int[][]tab2=
{
  {4, 15, 17},
  {2, 154, 170},
  {41, 125, 174}
};
```

Aby odwołać się np. do 1-szej kolumny i 1-szego wiersza

```
System.out.println(tab2[0][0]);
```

ENHANCED FOR - to ulepszona pętla for. Możemy jej używać do tablic. Do stworzenia tej pętli potrzebujemy określić typ tablicy, nazwę zmiennej oraz nazwę tablicy do której się odwołujemy.

Przykład pętli ENHANCED FOR

```
for(String nazwaKursu: kursyProgramowania)
  {
      System.out.println(nazwaKursu);
  }
```

for (typ tablicy nazwa zmiennej, która będzie przechowywać wartości przy każdym przejściu pętli: nazwa tablicy po której chcemy przejść)

Jeżeli chcemy sprawdzić poszczególne przejście musimy stworzyć **zmienną** np.

```
int i = 0;

for(String nazwaKursu: kursyProgramowania)
{
     i++;
     if(i != 2)
        System.out.println(nazwaKursu);
}
```

Pętla for - to typ pętli, która sprawdza warunek na samym starcie. Średniki w niej zawarte pozwalają stworzyć dodatkowe miejsce na operacje, które tworzymy w jednej linii.

```
public static void main(String[] args) {
     String[] kursyProgramowania =
     {
         "Java",
         "Java Aplikacje",
         "Java Strumienie",
         "Java Aspekty Zaawansowane",
         "Java Android",
         "C#",
         "C# Tworzenie Aplikacji",
         "C# LINQ",
         "Pascal"
     };
```

Przykład pętli for

```
for(int i = 0; i < kursyProgramowania.length; i++)

    {
        System.out.println(kursyProgramowania[i]);

    }
```

Schemat tworzenia pętli for

```
for (inicjalizacja zmiennych; warunek pętli; co ma się stać po wykonaniu instrukcji)
```

ĆWICZENIE - suma zmiennych

```
int [] liczby = {1, 12, 41, 51, 12};
```

- Zastosujemy pętle for, ponieważ chcemy wykonać działania kilka razy.
- Tworzymy zmienną - neutralną (0) i nazywamy ją suma. Dzięki niej będziemy mogli przypisać do sumy poszczególne wartości liczb

```
int [] liczby = {1, 12, 41, 51, 12};

int suma = 0;

for (int i = 0; i < liczby.length; i++)
{
    suma = suma + liczby[i];
}

System.out.println(suma)
```

Możemy też zastosować ulepszoną pętle for, czyli pętle enhanced for

```
for(int liczba: liczby)
{
   suma += liczba;
}

System.out.println(suma)
```

W zmiennej “liczba” będzie przechowywana za każdym razem po każdej interakcji nowa wartość tej tablicy.

Pętla - służy do zapętlania, czyli powtarzania instrukcji nieprzerwanie aż do momentu kiedy warunek będzie spełniony.

Z pętli korzystamy, gdy:

- mamy dużą ilość danych
- powtarzamy daną czynność wielokrotnie
- instrukcja się powtarza

while - podczas, gdy

```
int i = 0;

while(i < 7)
{
  System.out.println(i);
}
```

Stworzyliśmy pętle nieskończoną, która będzie wykonywać się ciągle, ponieważ nasz warunek będzie zawsze prawdziwy.

Aby pętla zakończyła swoje działanie musimy zmienić warunek. Możemy to zrobić za pomocą operatora **i++ ,** czyli za każdym razem będzie dodawać do zmiennej **i** liczbę 1

```
int i = 0;
while(i < 7)
{
  System.out.println(i);
  i++;
}
```

Pętla zakończy swoje działanie i wypisze liczby od 0-6.

Podczas tworzenia programów należy pamiętać, że muszą być one uniwersalne, aby nie zmieniać za każdym razem naszego kodu. Warto wtedy odwołać się do tablicy i skorzystać z funkcji length - która zwróci nam aktualną długość tablicy.

```
int i = 0;
while(i < kursyProgramowania.length);
{
  System.out.println(kursyProgramowania[i]);

  i++;
}
```

W środku pętli można również zastosować instrukcje, warunki np.

```
int i = 0;
while(i < kursyProgramowania.length);
{
  if (i != 3)
     System.out.println(kursyProgramowania[i]);

  i++;
}
```

Pętla do while

```
i = 0

do
{
    System.out.println(kursyProgramowania[i]);

    i++;

} while(i < kursyProgramowania.length);
```

Stosujemy ją, gdy nie interesuje nas czy warunek na samym starcie musi być zostać spełniony.