



**WYŻSZA SZKOŁA BANKOWA**  
**Warszawa**

# Programowanie aplikacji w Java

**Maciej Gowin**

**Zjazd 1 - dzień 1**

# Linki

## Opis

<https://maciejgowin.github.io/wsb-java/>

## Kod źródłowy przykładów oraz zadań

<https://github.com/MaciejGowin/wsb-programowanie-aplikacji-java>

# Architektura komputera

- procesor: wykonywanie operacji
- pamięć SSD/dysk twardy: pamięć długotrwała
- pamięć RAM: pamięć krótkotrwała

Instrukcje programu są wgrywane do pamięci komputera w postaci **kodu maszynowego**. Kod maszynowy definiuje instrukcje zrozumiałe dla procesora. Jest on definiowany przy pomocy ciągu liczb.

# Kod maszynowy

[illegible]

Ciąg 8-bitowych rozkazów dla procesora. Zapis szesnastkowy możemy przekształcić w zapis binarny.

# Kod maszynowy

Kod szesnastkowy	Kod binarny
6C	01101100
36	00110110
00	00000000

Kod maszynowy to instrukcje niskopozycyjne i operują na rejestrach procesora. Są one zależne od typu/architektury procesora (x86, AMD).

# Język asemblera

- język niskiego poziomu
- zrozumiały dla człowieka
- operacje odpowiadają faktycznym rozkazom procesora
- używa mnemonik
- zależy od architektury procesora, ale i używanego asemblera, jednak zwykle autorzy asemblerów dla danego procesora trzymają się oznaczeń danych przez producenta

# Asembler

- dokonuje tłumaczenia języka asemblera na język maszynowy w procesie **aseemblacji**
- zależy od platformy, ale też od systemu operacyjnego

Język asemblera jest często zależny od samego asemblera używanego do jego tłumaczenia na kod maszynowy.

# Przykładowy kod języka asemblera

```
;do rejestru RAX wpisz wartość natychmiastową 02h
mov rax, 02h
;do rejestru RCX wpisz wartość z rejestru RAX
mov rcx, rax
;do zmiennej o nazwie var1 wpisz wartość z rejestru RCX
mov var1, rcx
;odłóż wartość z rejestru RAX na stos
push rax
;zdejmij wartość ze stosu i umieść w zmiennej o nazwie var2
pop var2
```



# Hardware a software

**Hardware** - fizyczne komponenty wymagane do poprawnego działania systemu komputerowego.

**Software** - zestaw instrukcji, danych oraz programów używanych do wykonywania konkretnych zadań przy użyciu systemu komputerowego.

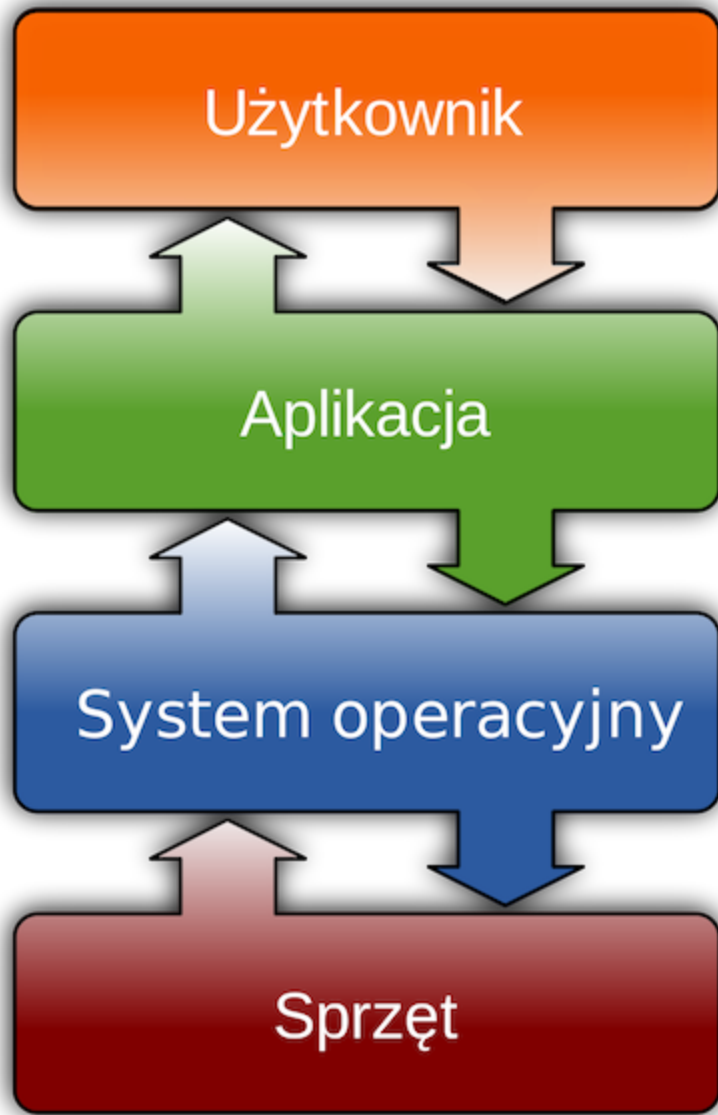
# System operacyjny

System odpowiedzialny za zarządzanie systemem komputerowym, w tym jego:

- zasobami
- procesem
- pamięcią operacyjną
- plikami
- wejściem/wyjściem
- danymi

System operacyjny stanowi interfejs pomiędzy aplikacjami a sprzętem.

# System operacyjny



# Języki wysokopoziomowe

- programowanie w języku assemblera jest uciążliwe
- powstały języki wysokiego poziomu, które mają na celu ułatwienie zrozumienia kodu przez człowieka
- zwiększają poziom abstrakcji ukrywając szczegóły sprzętowe
- przykłady języków wysokiego poziomu: C, C++, Java, Go, Scala, Kotlin, PHP, Ruby

# Kompilacja

Kod języka wysokiego poziomu nie jest zrozumiały przez komputer. Wynika z tego, że kod źródłowy języka wysokiego poziomu musi zostać przetłumaczony na język maszynowy lub kod asemblera.

**Kompilacja** jest procesem transformacji kodu źródłowego języka wysokiego poziomu na kod języka niskiego poziomu.

# Kompilacja na przykładzie języka C

Przykładem języka wysokiego poziomu jest język C, jeden z najszerzej używanych języków.

Kod źródłowy *main.c*

```
#include <stdio.h>
int main() {
    printf("Hello, World!\n");
    return 0;
}
```

# Kompilacja na przykładzie języka C

Kompilacja do kodu wykonywalnego *a.out*

```
gcc main.c
```

Uruchomienie

```
gowinm:~/> ./a.out  
Hello, World!
```

# Kompilacja na przykładzie języka C

Z kodu źródłowego napisanego w języku C podczas **kompilacji** powstaje kod zrozumiały dla komputera, czyli kod asemblera, który w tym przypadku jest zależny od platformy.

Warto zauważyć, że:

- kod skompilowany dla procesora Intel nie będzie działał na procesorze AMD
- kod skompilowany na platformie UNIX nie będzie działał na platformie Windows

W takim przypadku mówimy, że skompilowany plik wynikowy **nie jest przenośny**.



# Kompilacja na przykładzie języka C

```
#include <stdlib.h>
int sub(int x, int y){
    return 2*x+y;
}
```

```
int main(int argc, char ** argv){
    int a;
    a = atoi(argv[1]);
    return sub(argc,a);
}
```

```
.text:00000000 _sub:    push    ebp
.text:00000001         mov     ebp, esp
.text:00000003         mov     eax, [ebp+8]
.text:00000006         mov     ecx, [ebp+0Ch]
.text:00000009         lea     eax, [ecx+eax*2]
.text:0000000C         pop     ebp
.text:0000000D         retn
.text:00000010 _main:   push    ebp
.text:00000011         mov     ebp, esp
.text:00000013         push   ecx
.text:00000014         mov     eax, [ebp+0Ch]
.text:00000017         mov     ecx, [eax+4]
.text:0000001A         push   ecx
.text:0000001B         call    dword ptr ds:__imp__atoi
.text:00000021         add     esp, 4
.text:00000024         mov     [ebp-4], eax
.text:00000027         mov     edx, [ebp-4]
.text:0000002A         push   edx
.text:0000002B         mov     eax, [ebp+8]
.text:0000002E         push   eax
.text:0000002F         call    _sub
.text:00000034         add     esp, 8
.text:00000037         mov     esp, ebp
.text:00000039         pop     ebp
.text:0000003A         retn
```

# Dlaczego kod wykonywalny jest zależny od platformy?

1. Plik wykonywalny zawiera szereg instrukcji, które wskazują, jak powinien on zostać załadowany do pamięci.
2. Każdy z systemów operacyjnych ma swój własny format plików wykonywalnych odpowiadający możliwościom danego systemu operacyjnego.
3. Każdy z plików wykonywalnych może wymagać wczytania bibliotek specyficznych dla systemu.
4. Dodatkowo każdy z procesorów dostarcza zestaw instrukcji, które jest w stanie obsłużyć.

# Sieć komputerowa

Systemy komputerowe nie są osobnymi bytami i istotą ich działania jest komunikacja. Połączenie to jest realizowane za pomocą sieci komputerowej.

Główne zadania sieci komputerowej:

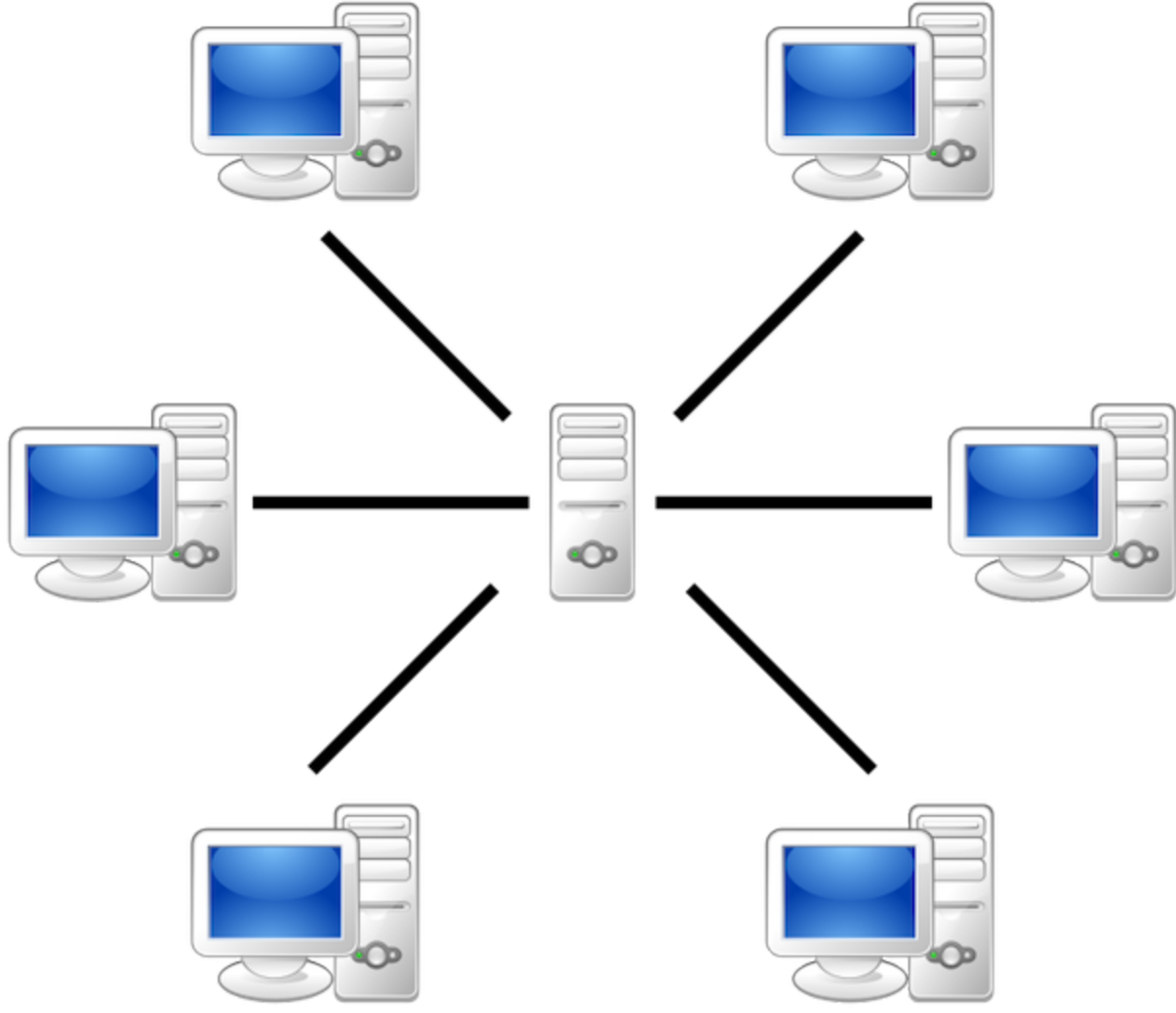
- przesył danych
- udostępnianie zasobów

# Komunikacja client-server

Przykładem komunikacji pomiędzy komputerami przy użyciu sieci komputerowej (w tym przypadku sieci Internet) jest udostępnianie zasobów, jakimi są strony internetowej.

Jest to przykład komunikacji typu **client-server**, w której:

- nasz komputer wraz z przeglądarką pełni rolę klienta (service requestor)
- dostawca strony internetowej pełni rolę serwera (service provider)

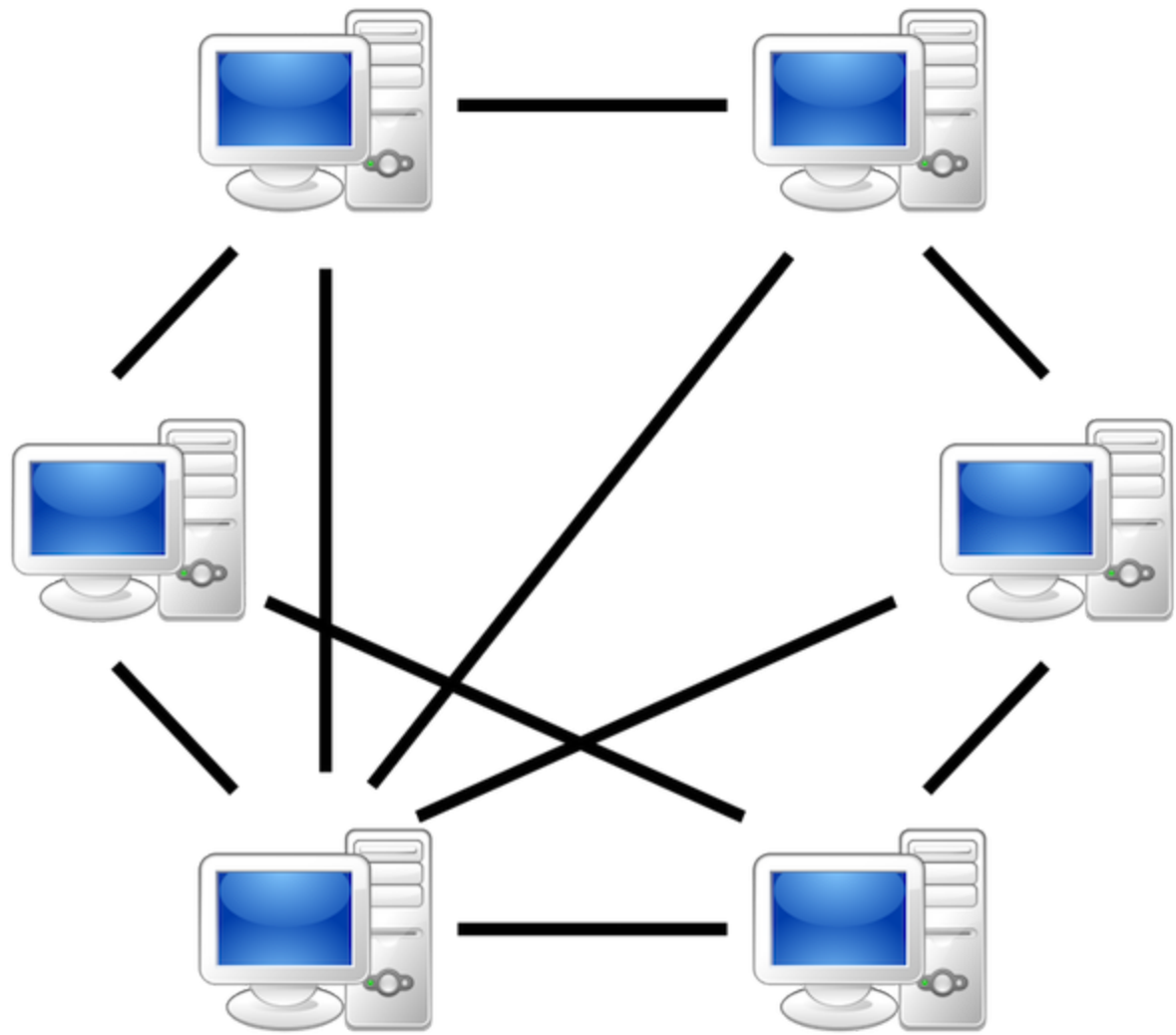


# Komunikacja peer-to-peer (P2P)

Przykładem komunikacji pomiędzy komputerami przy użyciu sieci komputerowej (w tym przypadku sieci Internet) jest współdzielenie i wymiana plików przy użyciu protokołu BitTorrent.

Jest to przykład komunikacji typu **peer-to-peer**, w której każdy z komputerów pełni funkcję:

- klienta pobierającego dane
- serwera udostępniającego dane



# Aplikacje desktopowe a webowe

**Aplikacje desktopowe** - aplikacje uruchomione lokalnie na systemie operacyjnym urządzenia. W naszym przykładzie komunikacji client-serwer przykładem aplikacji desktopowej jest przeglądarka internetowa.

**Aplikacje webowe** - aplikacje uruchomione na serwerze, do których dostęp odbywa się przy użyciu sieci komputerowej. W naszym przykładzie aplikacją webową będzie oprogramowanie udostępniające zawartość strony internetowej uruchomione na serwerze.



# Porównanie systemów frontend (FE) i backend (BE)

Przeglądarka wykonuje zapytanie do serwera. Serwer przetwarzający zapytanie pełni rolę systemu backendowego (BE). W odpowiedzi zostaje zwrócona zawartość strony internetowej (w formacie HTML) oraz dodatkowe zasoby takie jak: obrazy i pliki JavaScript. Pliki JavaScript zawierają instrukcje zrozumiałe dla przeglądarki i tworzą system frontendowy (FE), który steruje warstwą prezentacji strony internetowej.

- System frontend (FE) - pełni rolę prezentacji danych

- System backend (BE) - pełni rolę dostawcy danych

# Języki programowania: kompilowane a interpretowane

Nie każdy język programowania wysokiego poziomu jest językiem kompilowanym. Istnieje grupa języków, których kod źródłowy jest wczytywany, interpretowany i wykonywany przez interpreter języka.

Główną zaletą języków interpretowalnych jest ich przenośność pomiędzy systemami operacyjnymi, ponieważ to sam interpreter jest zależny od platformy. W niektórych przypadkach do wad tych systemów zaliczamy gorszą wydajność działania.

Przykłady: PHP, JavaScript, Ruby, Bash

Pojęcia kompilatora oraz interpretera nie wykluczają się wzajemnie. Przykładem mogą być interpretery dokonujące dynamicznej kompilacji kodu (JIT).

# Wprowadzenie do języka Java

- wysokopoziomowy język obiektowy
- stworzony w 1995 przez Jamesa Goslinga (Sun Microsystems, obecnie pod skrzydłami Oracle Corporation)
- został zaprojektowany tak, aby składnią przypominał język C/C++
- zakłada podejście "Write Once, Run Anywhere (WORA)"
- głównym założeniem języka Java jest przenośność: raz napisany i skompilowany program powinien działać, na każdej platformie w podobny sposób
- wersje Java: 1.0, 1.1, 1.2, 1.3, 1.4, 5.0, 5, 6 ... 17
- wszystkie wersje języka Java są kompatybilne: kod napisany w starszej wersji może być skompilowany przy użyciu nowszej wersji
- tylko niektóre wersje wspierają LTS (Long Time Support): Java 8, Java 11, Java 17

# Java i przenośność

Przenośność (portability) została osiągnięta poprzez kompilację **kodu źródłowego Java** do pośredniej reprezentacji, którą określamy mianem **Java bytecode** zamiast do zależnego od architektury kodu maszynowego.

Instrukcje Java bytecode są analogiczne do instrukcji kodu źródłowego, ale wykonywane są na maszynie wirtualnej Java (JVM, Java Virtual Machine), która jest uruchomiona i stworzona dla konkretnej platformy.

# Java i przenośność



# Wirtualna maszyna Java (JVM), JRE, JDK.

## **JVM (Java Virtual Machine)**

Abstrakcyjna maszyna uruchomiona na urządzeniu pozwalająca na wykonywanie programów Java.

## **JRE (Java Runtime Environment)**

Oprogramowanie, które dostarcza wbudowane biblioteki klas Java, wirtualną maszynę (JVM) oraz dodatkowe komponenty wymagane do uruchamiania aplikacji Java.

## Java Runtime Environment (JRE)

JVM

Biblioteki klas

# Wirtualna maszyna Java (JVM), JRE, JDK.

## JDK (Java Development Kit)

Zbiór narzędzi programistycznych służących to rozwijania aplikacji w języku Java. Wraz z JDK dostarczane jest JRE oraz zbiór narzędzi programistycznych.



## Java Development Kit (JDK)

### Java Runtime Environment (JRE)

JVM

Biblioteki klas

Kompilatory

Debugery

Narzędzia

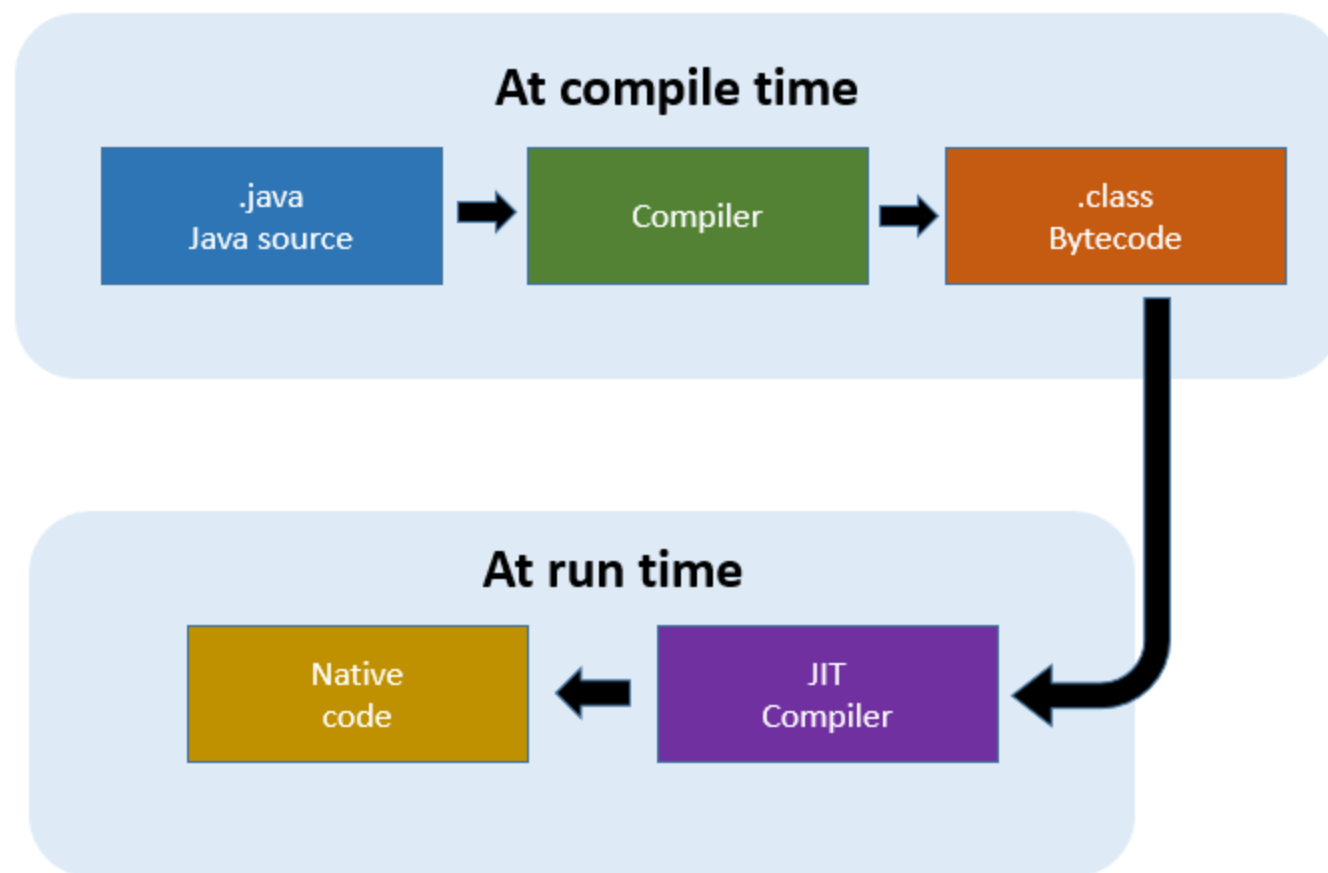
Dokumentacja (JavaDoc)

# JIT oraz Wydajność Java Bytecode

Użycie kodu pośredniego w postaci Java bytecode wprowadza konieczność interpretacji do kodu maszynowego danej platformy. Sprawia to, że programy takie prawie zawsze działają wolniej od natywnych kodów wykonywalnych.

# JIT oraz Wydajność Java Bytecode

Just-in-time (JIT) kompiluje Java bytecode do kodu maszynowego w trakcie działania programu. Usprawnia to działanie oraz poprawia wydajność.

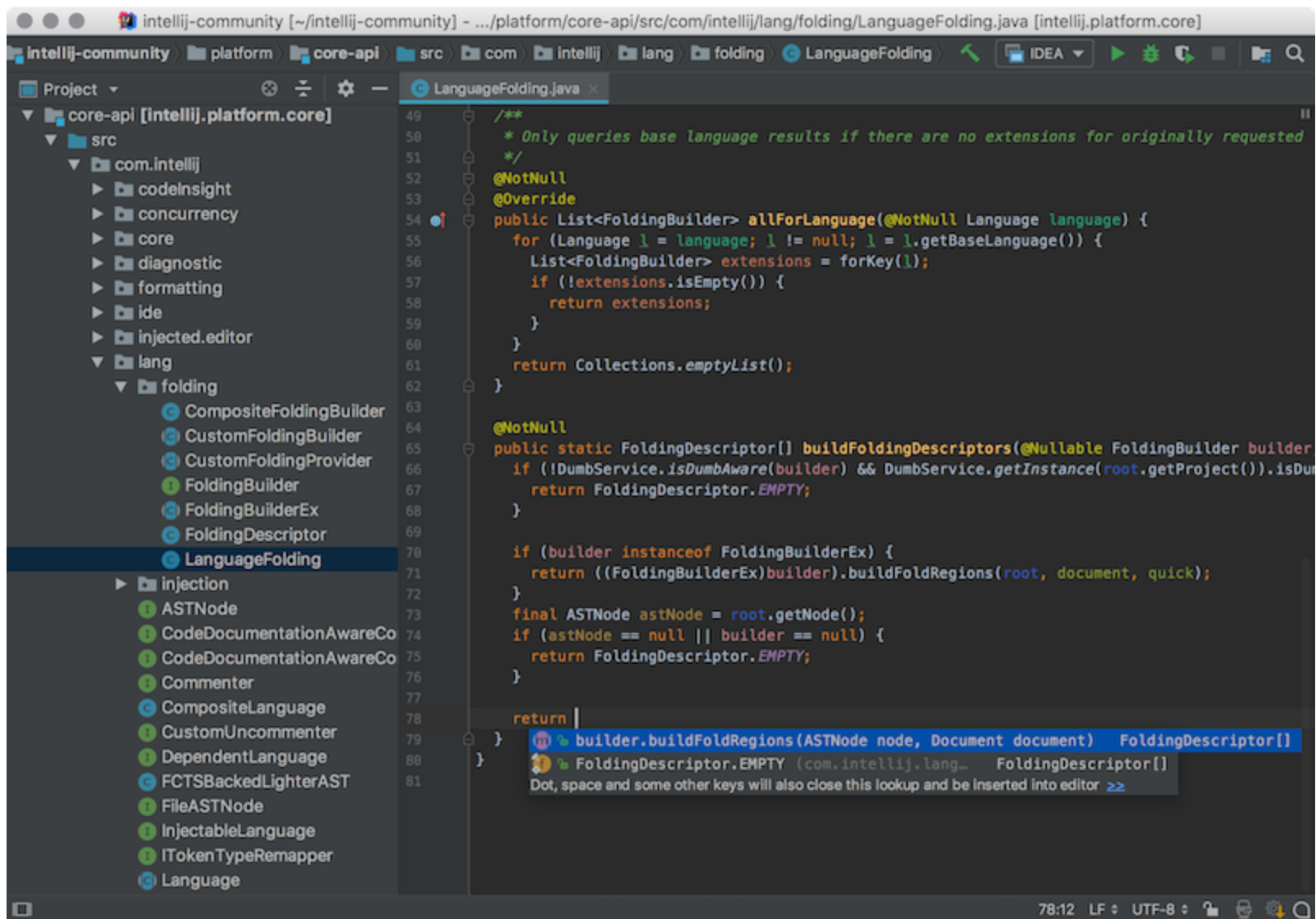


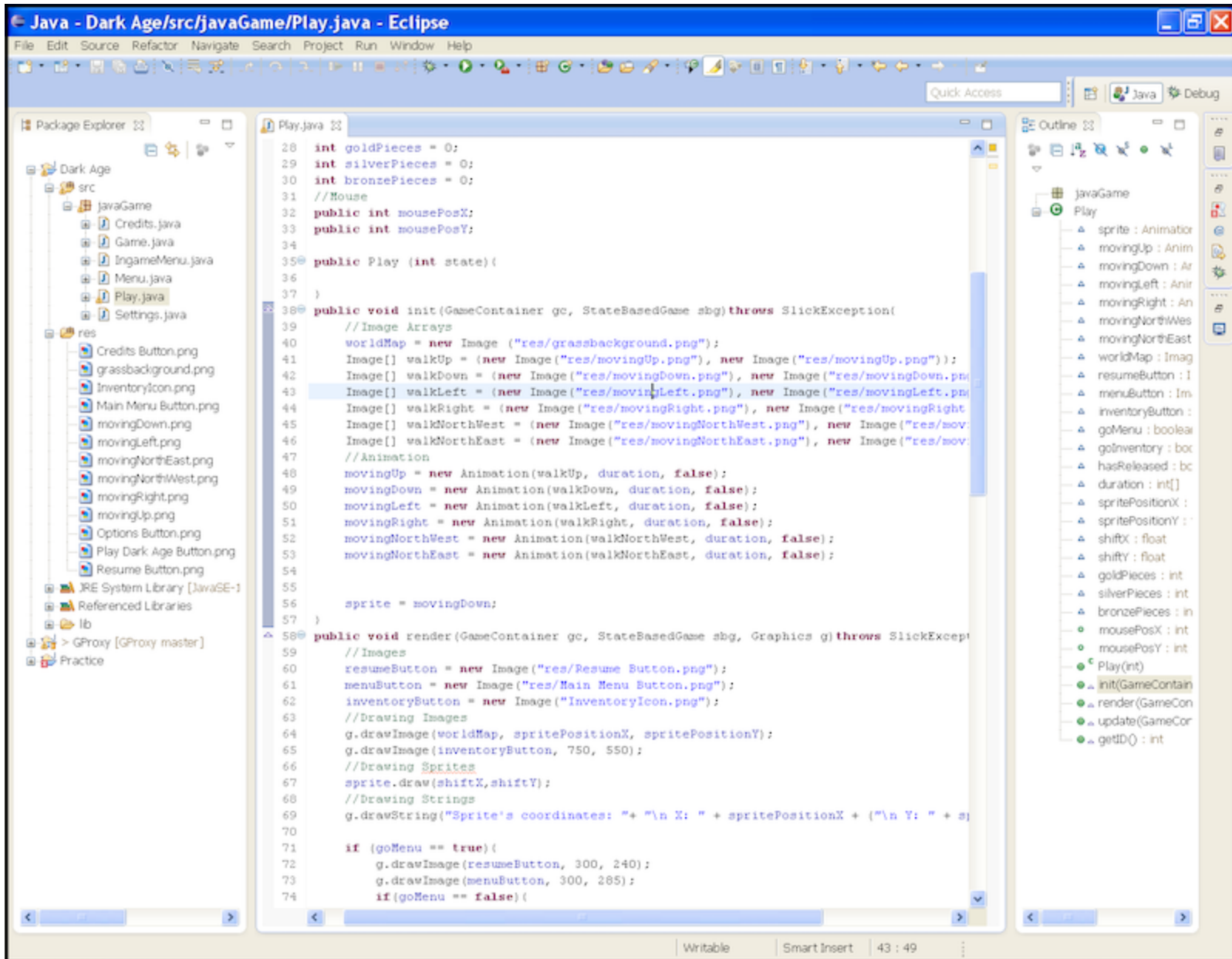
# Środowisko programistyczne IDE (IntelliJ, Eclipse)

Środowisko IDE (Integrated Development environment) jest oprogramowaniem, które dostarcza narzędzi programistycznych przyspieszających proces tworzenia oprogramowania.

Do głównych modułów IDE należą:

- edytor kodu źródłowego
- narzędzia automatyzujące proces budowania i uruchamiania aplikacji
- debugger





# Programowanie: przykład 01

Pierwszy program oraz kompilacja.

Plik: Main.java

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

```
javac Main.java  
java Main
```

# Język Java: metoda main

- Każda aplikacja musi posiadać definicję klasy, której nazwa jest zgodna z nazwą pliku.
- Definicja metody **main** musi znajdować się w środku klasy.
- Podczas uruchomienia wyszukiwana jest metoda **main**, która jest punktem wejściowym każdego programu.
- Każdy plik może zawierać tylko jedną klasę publiczną, ale też więcej klas niepublicznych.



# Programowanie: zadanie 01

1. Stwórz program wypisujący na ekran tekst `Programuję w Java` . Użyj do tego notatnika. Uruchom program przy pomocy linii poleceń.
2. Stwórz pierwszy projekt w IntelliJ IDEA o nazwie `zadanie-01` . Przenieś stworzony wcześniej plik do IDE. Uruchom program przy pomocy IDE.

# Język Java: zmienne

## Definicja zmiennej

Zmienna reprezentuje przestrzeń w pamięci przechowującą wartość. Jest ona identyfikowana przez unikatową nazwę.

```
int age = 35;
```

W powyższym przykładzie do zmiennej typu `int` została przypisana (`=`) wartość `35`. W Javie możliwe jest zdefiniowanie zmiennej oraz przypisanie wartości (lub jej nadpisanie) na późniejszym etapie.

```
int age;  
age = 35;  
age = 40;
```

# Język Java: zmienne

## Definicja zmiennej

Istotne jest to, że nie możemy zdefiniować dwóch zmiennych o tej samej nazwie, lecz innych typach w danym zasięgu.

```
int age = 35;  
char age;
```

# Język Java: zmienne

## Zasady

Istnieje szereg zasad, którymi rządzą się zmienne, ich nazewnictwo oraz sposób użycia.

- Język Java rozróżnia duże i wielkie litery (case sensitive). Dlatego też zmienne `height` oraz `HEIGHT` to dwie różne zmienne.
- Zmienna musi zaczynać się od litery, podkreślenia ( `_` ) lub dolara ( `$` ). Dozwolone jest używanie liczb po pierwszym znaku: `height1` , `he1ght` .
- Zmienne nie mogą zawierać spacji: `int my height` .
- Zmienne nie mogą używać słów kluczowych takich jak: `abstract` , `static` , `final` .

# Język Java: zmienne

## Dobre praktyki

- Istnieją dwa główne podejścia nazewnicze pozwalające na łatwiejsze oddzielenie nazw zmiennych składających się z więcej niż jednego słowa, przy czym camel case jest typowy dla języka Java:
  - camel case: `firstName`
  - snake case: `first_name`
- Dla zmiennych jednowyrazowych nie zmieniamy wielkości liter.
- Zmienne powinny zawierać w sobie podstawową informację o ich znaczeniu, unikamy zmiennych typu `i`, które utrudniają czytanie kodu.

# Język Java: typy prymitywne

Typ danych, jak sama nazwa wskazuje, definiuje typ, który może zostać przypisany do zmiennej. Warto wspomnieć, że Java jest językiem silnie typowanym i musi on być zawsze zdefiniowany.

Istnieją dwie grupy typów: prymitywne oraz typy obiektowe. Wyróżniamy 8 typów prymitywnych.

# Język Java: typy prymitywne

## boolean

Definiuje logiczną wartość prawda ( `true` ) lub fałsz ( `false` ).

```
boolean saved = true;
```

## byte

Definiuje wartość całkowitą z przedziału -128, 127 (8 bitów).

```
byte move = 121;
```

# Język Java: typy prymitywne

## short

Definiuje wartość całkowitą z przedziału -32768 to 32767 (16 bitów).

```
short move = 121;
```

## int

Definiuje wartość całkowitą z przedziału  $-2^{31}$  to  $2^{31}-1$  (32 bitów).

```
int move = 123455678;
```



# Język Java: typy prymitywne

## long

Definiuje wartość całkowitą z przedziału  $-2^{63}$  to  $2^{63}-1$  (64 bitów).

```
long move = 123455678L;
```

## double

Definiuje liczbę zmiennoprzecinkową podwójnej precyzji (64 bitów).

```
double move = 42.3;
```

# Język Java: typy prymitywne

## float

Definiuje liczbę zmiennoprzecinkową pojedynczej precyzji (32 bitów).

```
float move = 42.3F;
```

## char

Definiuje literę Unicode (16 bitów)

```
char letter1 = '\u0069';  
char letter2 = 'i';
```

# Język Java: literały

Literały pozwalają na zdefiniowanie z góry ustalonej wartości. Używamy ich do zainicjalizowania zmiennej lub w wyrażeniach.

```
int left = 9;  
int right = left * 9;
```

# Język Java: typ obiektowy String

Przykładem klasy oraz obiektu jest typ wbudowany String. Nie jest on typem prymitywnym, ponieważ jest on reprezentowany przez klasę `java.lang.String`. Zmienna typu String możemy zainicjalizować również przy pomocy literału:

```
String name = "Maciej";
```

# Język Java: operatory

Operatory definiują operacje na zmiennych oraz literałach.

## Arytmetyczne

Definiują operacje arytmetyczne na zmiennych oraz literałach. Zaliczamy do nich:

- dodawanie: +
- odejmowanie: −
- mnożenie: \*
- dzielenie: /
- resztę z dzielenia (module): %

```
int square = 8*8;  
int reminder = 8 % 3;  
int longCalculation = (2 + 3) * (4 + 5);  
int combination = (3 + reminder) * square + longCalculation;
```

## Programowanie: zadanie 02

Stwórz projekt `zadanie-02`. Dodaj klasę `Converter` wraz z metodą główną `main`, w której zdefiniowane będą dwie zmienne: `priceInEur` oraz `eurToPlnRatio`. Oblicz końcową wartość towaru w PLN oraz przypisz do zmiennej `priceInPln`. Wynik wypisz na ekran.

```
Cena w EUR: 5  
Przelicznik: 4.25  
Cena w PLN: 21.25
```

# Język Java: operatory

## Przypisania

Pozwalają na przypisanie wartości do zmiennej.

```
int a = 5;  
a += 75;  
a -= 20;  
a *= 6;  
a /= 120;
```

# Język Java: operatory

## Porównania

Porównują wartość dwóch operandów (zmiennych lub literałów) zwracając wartość logiczną ( `true` albo `false` ).

```
int price = 1000;  
boolean isExpensive = price > 500;
```



# Język Java: operatory

## Porównania

Operator	Opis
==	równy
!=	nie równy
<	mniejszy
>	większy
<=	mniejszy lub równy
>=	większy lub równy

# Język Java: operatory

## Logiczne

Sprawdzają, czy wyrażenie jest prawdą lub fałszem. Operują na zmiennych lub wyrażeniach zwracających typ boolean.

```
boolean isRed = true;  
boolean isCar = false;  
boolean isRedCar = isRed && isCar;  
boolean isNotRed = !isRed;
```

# Język Java: operatory

## Logiczne

Operator	Opis
&&	oraz
	lub
!	negacja

# Język Java: operatory

## Logiczne

Do obliczania wartości używamy algebry Boole'a.

Operacja	Wartość
false && false	false
true && false	false
false && true	false
true && true	true

# Język Java: operatory

## Logiczne

Do obliczania wartości używamy algebry Boole'a.

Operacja	Wartość
false    false	false
true    false	true
false    true	true
true    true	true

# Język Java: operatory

## Logiczne

Do obliczania wartości używamy algebry Boole'a.

Operacja	Wartość
!true	false
!false	true

# Język Java: operatory

## Jednoargumentowe

Operują na jednym argumencie.

```
int six = 6;  
int minusSix = -six;  
int order = 5;  
++order;
```

# Język Java: operatory

## Jednoargumentowe

Operują na jednym argumencie.

Operator	Opis
+	wartość dodatnia, raczej nieużywany ponieważ z definicji nie zmienia znaku
-	odwrócenie znaku wartości
++	inkrementacja, zwiększenie wartości o 1
--	dekrementacja, zmniejszenie wartości o 1
!	zaprzeczenie



## Programowanie: zadanie 03

Sprawdź różnicę pomiędzy wywołaniami `++x` oraz `x++`. Możesz posłużyć się następującymi poleceniami:

```
int x = 5;  
int y = ++x;  
int z = x++;
```

# Język Java: operatory

## Binarne

Operują na bitach. Operacje te nie należą do często używanych.

Operator	Opis
~	dopełnienie bitowe, zamienia wartość każdego bitu
<<, >>	przesunięcie w lewo, w prawo
>>>	bezznakowe przesunięcie w prawo
&, ^	bitowe AND, OR

# Język Java: operatory

## Warunkowy

Definiuje warunkowe zwrócenie wartości na podstawie wyniku wyrażenia warunkowego.

```
int age = 16;  
String title = (age >= 18) ? "Adult" : "Child";
```

# Język Java: operatory

## Konkatenacja

Operator `+` jest używany do łącznia literałów tekstowych w jeden ciąg znaków. Operację tą nazywamy konkatenacją.

```
String firstName = "Maciej";  
String lastName = "Gowin";  
String fullName = firstName + " " + lastName
```

## Programowanie: zadanie 04

Używając operacji modulo oraz operatora warunkowego sprawdź, czy liczba całkowita jest parzysta lub nieparzysta.

# Język Java: wyrażenia i bloki

W języku Java **wyrażeniem** (statement) nazywamy każdą wykonywalną jednostkę. Wyrażenia zakończone są znakiem średnika **;**.

```
int a = 6;  
int b = a + 5;
```

Każde z wyrażen składa się z jednej lub więcej **instrukcji** (expression).

```
a = 6  
a + 5
```

# Język Java: wyrażenia i bloki

Ważnym elementem języka są **bloki kodu**, które zawarte są pomiędzy `{` oraz `}`. Tworzą one grupę, przy czym każda może składać się z żadnego lub kilku wyrażień. Blok jest używany przy definicji każdej funkcji tak jak dla funkcji `main`.

```
public static void main(String[] args) {  
    System.out.println("Hello, World!");  
}
```

# Programowanie: przykład 02

Bloki definiują nie tylko zakres funkcji, ale też są używane w bardziej skomplikowanych instrukcjach warunkowych.

```
public class Main {  
    public static void main(String[] args) {  
        int age = 17;  
  
        if (age < 18) {  
            System.out.println("Warning!");  
        }  
  
        String type = (age < 18) ? "Child": "Adult";  
        System.out.printf("You are: %s\n", type);  
    }  
}
```



# Język Java: instrukcja warunkowa if...else

Jedną z głównych instrukcji prawie każdego języka jest instrukcja `if...then`. Jej zadaniem jest warunkowe wykonanie szeregu wyrażeń. Zasadę jej działania możemy opisać następująco:

```
IF (JEŻELI) warunek jest spełniony  
    THEN (WTEDY) wykonaj wyrażenie
```

Przykładowo:

```
if (age > 18) {  
    // execute if age is above 18  
}
```

# Język Java: instrukcja warunkowa if...else

Na przykładzie widzimy, że kod zostanie wykonany, tylko jeżeli warunek `age > 18` zwróci `true`. W przeciwnym wypadku kod zostanie pominięty. Rozszerzeniem tego warunku jest instrukcja `if...then...else`, która pozwala na wykonanie dodatkowej akcji w przypadku gdy warunek nie będzie spełniony i zwróci `false`.

```
IF (JEŻELI) warunek jest spełniony  
    THEN (WTEDY) wykonaj wyrażenie  
ELSE (W PRZECIWNYM RAZIE) wykonaj wyrażenie
```

Przykładowo:

```
if (age > 18) {  
    // execute if age is above 18  
} else {  
    // execute if age is not above 18  
}
```

# Język Java: instrukcja warunkowa if...else

Oczywiście nie jest to ograniczone tylko do jednego wyrażenia. Wykonane zostaną wszystkie wyrażenia, które zawarte są w bloku `{...}`.

Kolejnym rozszerzeniem jest instrukcja `if...else...if`, która pozwala na sprawdzenie kilku warunków. Co istotne są one sprawdzane kolejno. Przy spełnionym warunku wykonywane są wyrażenia do niego przypisane. Jeżeli jeden warunek zostanie spełniony, kolejny nie jest już sprawdzany.

```
IF (JEŻELI) warunek jest spełniony
    THEN (WTEDY) wykonaj wyrażenie
ELSE IF (W PRZECIWNYM RAZIE JEŻELI) warunek jest spełniony
    THEN (WTEDY) wykonaj wyrażenie
ELSE (W PRZECIWNYM RAZIE) wykonaj wyrażenie
```

# Język Java: instrukcja warunkowa if...else

Przykładowo:

```
if (age > 18) {  
    // execute if age is above 18  
}  
else if (age > 16) {  
    // execute if age is above 16  
} else {  
    // execute if none of above met  
}
```

# Programowanie: przykład 03

Sprawdzenie kilku warunków w instrukcji `if...else...if`.

```
public class Main {  
    public static void main(String[] args) {  
        int age = 19;  
  
        if (age > 18) {  
            System.out.println("You are an adult");  
        } else if (age > 16) {  
            System.out.println("You are a teenager");  
        } else {  
            System.out.println("You are a child");  
        }  
  
        System.out.println("You are a human");  
    }  
}
```

# Język Java: instrukcja warunkowa if...else

Co ważne, bloki oraz instrukcje mogą być zagnieżdżone.

```
if (age > 18) {  
    if (gender == "F") {  
        System.out.println("Adult female");  
    } else {  
        System.out.println("Adult male");  
    }  
}
```

# Programowanie: zadanie 05

Zdefiniuj zmienne `age` oraz `animal`. W zależności od wartości sprawdź, czy dane zwierzę jest:

- starym psem
- młodym psem
- starym kotem
- młodym kotem