

Programowanie w języku Java

MGR INŻ. SZYMON GUZIK

Kontakt:
e-mail: szguzik@wsb.gda.pl



Odpisuję po godzinie 18-tej lub z samego rana (ok. 6-tej)

Zasady zaliczenia przedmiotu:

- obecność na zajęciach (100% obecności podnosi ocenę)
- prezentacja projektu na ostatnich zajęciach (max 5-10 min.) (obowiązkowa) (własne tematy / projekty)
- test
 - 60% - 3
 - 70% - 3.5
 - 80% - 4
 - 90% - 4.5
 - 100% - 5

Obecność (max 1,2 nieobecności) + prezentacja + test 80% = ocena bardzo dobry

Obecność (max 1,2 nieobecności) + prezentacja + test 70% = dobry

Obecność (max 1,2 nieobecności) + prezentacja + test 60% = dostateczny

Przerwa (15 min.)



Po przeprowadzeniu 1 h 30m zajęć

Po kawę i do toalety można wyjść w każdym momencie ☺

Laboratorium

Co będziemy robić ?

- Porozmawiamy o kodowaniu
- Trochę definicji
- Opanujemy podstawowe zasady kodowania
- Omówimy oraz przeanalizujemy
 - Instrukcje warunkowe
 - Pętle warunkowe
- Logika matematyczna
- Operatory logiczne
- Poznamy
 - Metody
 - Tablica jednowymiarowa
 - Typy danych
 - Znaki specjalne
- Przejdziemy proces instalacji środowiska Java
- Środowiska IDE
- Uruchomimy pierwszy program
- Ustalimy tematy projektów

Laboratorium

Co będziemy robić ?

- Zainstalujemy oprogramowanie IntelliJ
- Uruchamianie programu w konsoli
- System.out.print("")
- System.out.format("")
- Komentarze
- Zmienne
- Zadania
- Stałe
- Liczba PI
- Typy danych – ponownie
- Potęgowanie
- Wprowadzanie danych przez użytkownika
- Zadanie
- Klasa osłonowa

Laboratorium

Co będziemy robić ?

- Konwersja zmiennych na inny typ danych
- Trójargumentowy operator
- Switch
- Pętle (przypomnienie)
 - While
 - Do While
 - For
- Zagnieżdżanie pętli
- Instrukcje skoku
- Zadanie
- Tablice
 - Ogólnie (przypomnienie)
 - For Each
 - Tablice wielowymiarowe

Laboratorium

Co będziemy robić ?

- Funkcje
- Funkcje – argumenty funkcji
- Funkcje - zwracanie wartości z funkcji
- Przeciążanie nazw funkcji
- OOP
- OOP – implementacja
- Konstruktory
- Static
- Non-static
- Hermetyzacja
- Dziedziczenie
 - Dziedziczenie – konstruktory
 - Dziedziczenie – nadpisywanie
 - Dziedziczenie – metody abstrakcyjne
- Interfejsy
- Modyfikatory dostępu
- Polimorfizm
- Enum

Porozmawiajmy o kodowaniu

- ▶ Czy ktoś z Was już kodował ?
- ▶ Jakie programy pisaliście ?
- ▶ Czym są algorytmy ?
- ▶ Czym są obiekty ?
- ▶ Co to są schematy blokowe ?

Trochę definicji

„**Algorytm** - skończony ciąg jasno zdefiniowanych czynności koniecznych do wykonania pewnego rodzaju zadań, sposób postępowania prowadzący do rozwiązania problemu”

[WIKIPEDIA]

Potocznie ujmując – „Przepis” na program.

Trochę definicji

„**Obiekt** - to wycinek rzeczywistości (wyodrębniony element)posiadający własne cechy (właściwości) oraz mogący wykonywać pewne akcje (metody). Obiektem może wszystko w świecie rzeczywistym, np. samochody, ludzie, domy, komputery”

[WIKIPEDIA, agh.edu.pl]

Trochę definicji

„Klasa – Klasa jest to zestaw cech oraz możliwych akcji na podstawie, której został zbudowany obiekt.”

[agh.edu.pl]



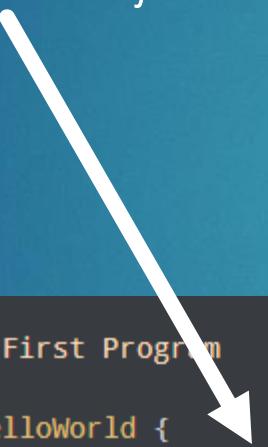
```
// Your First Program

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

Trochę definicji

*„**Metoda** – jest to funkcja wewnętrz klasy*

[WIKIPEDIA, agh.edu.pl]



```
// Your First Program

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

Podstawowe zasady kodowania

- ▶ Nazwa klas powinna wskazywać na jej funkcjonalność
- ▶ Nazwa metod, funkcji powinna określić jej zadanie
- ▶ Metody oraz klasy powinny odpowiadać za pojedynczą funkcjonalność
- ▶ Nazwy zmiennych określają ich zadanie
- ▶ Analogicznie nazwy stałych
- ▶ Używamy wzorców projektowych
- ▶ Kod powinien być elastyczny
- ▶ Stosujemy zasadę DRY (Don't Repeat Yourself)
- ▶ System kontroli wersji (np. GIT)

Instrukcje warunkowe

Instrukcja warunkowa to po prostu rozgałęzienie w działaniu programu (ang. **if** = jeżeli). W zależności od tego, czy warunek zawarty w instrukcji jest prawdziwy lub fałszywy, wykonane zostają inne instrukcje. Klauzula **else** jest opcjonalna, to znaczy nie musi koniecznie wystąpić – zależy to od nas i rozpatrywanego problemu. Składnia w pseudokodzie:

[MIROSŁAW ZELENT]

```
int temperature = 38;  
if (temperature < 36) {  
    System.out.println("Jesteś osłabiony?");  
}  
else if (temperature < 37) {  
    System.out.println("Wszystko w normie!");  
}  
else if (temperature < 38) {  
    System.out.println("Jesteś przeziębiony?");  
}  
else {  
    System.out.println("Masz co najmniej 38 stopni! Biegiem do lekarza!");  
}
```

Pętle warunkowe

Pętle (ang. loop) to konstrukcje językowe służące do zdefiniowania szeregu instrukcji, które będą powtarzane wielokrotnie. Poznajmy najbardziej podstawowe rodzaje pętli: for, while

Pętla For

[MIROSŁAW ZELENT]

```
for (int number = 0; number <= 10; number++) {  
    System.out.println(number);  
}
```

Pętle warunkowe

Pętle (ang. loop) to konstrukcje językowe służące do zdefiniowania szeregu instrukcji, które będą powtarzane wielokrotnie. Poznajmy najbardziej podstawowe rodzaje pętli: for, while

Pętla While

[MIROSŁAW ZELENT]

```
int number = 0;  
  
while (number < 10) {  
  
    System.out.println(number);  
  
    number++;  
}
```

Pętle warunkowe

Pętle (ang. loop) to konstrukcje językowe służące do zdefiniowania szeregu instrukcji, które będą powtarzane wielokrotnie. Poznajmy najbardziej podstawowe rodzaje pętli: for, while

[MIROSŁAW ZELENT]

Pętla Nieskończoności

```
while(true) {  
    System.out.println(1);  
}
```

Logika matematyczna

Decyzje pętli oraz instrukcji podejmowane są na podstawie logiki matematycznej

Negacja

p	$\neg p$
0	1
1	0

Koniunkcja

p	q	$p \wedge q$
0	0	0
0	1	0
1	0	0
1	1	1

Logika matematyczna

Decyzje pętli oraz instrukcji podejmowane są na podstawie logiki matematycznej

Alternatywa

p	q	$p \vee q$
0	0	0
0	1	1
1	0	1
1	1	1

Implikacja

p	q	$p \Rightarrow q$
0	0	1
0	1	1
1	0	0
1	1	1

Logika matematyczna

Decyzje pętli oraz instrukcji podejmowane są na podstawie logiki matematycznej

Równoważność

p	q	$p \Leftrightarrow q$
0	0	1
0	1	0
1	0	0
1	1	1

Alternatywa wykluczająca

p	q	$p \vee q$
0	0	0
0	1	1
1	0	1
1	1	0

Operatory logiczne

- ▶ == równe,
- ▶ < mniejsze,
- ▶ <= mniejsze bądź równe,
- ▶ > większe,
- ▶ >= większe bądź równe

```
int x = 1;  
int y = 1;  
int z = 2;  
  
x == y; // true  
x == z; // false  
x < y; // false  
x < z; // true  
x <= y; // true  
// itd.
```

Operatory logiczne

Proste operacje logiczne możesz ze sobą łączyć przy pomocy dodatkowych operatorów:

- ▶ **&&** logiczne **i**,
- ▶ **||** logiczne **lub**.

Pomocna przy tym może być następująca tabela. Pokazuje ona podstawowe operacje logiczne

Operacja	Wynik
prawda i prawda	prawda
prawda i fałsz	fałsz
fałsz i prawda	fałsz
fałsz i fałsz	fałsz
prawda lub prawda	prawda
prawda lub fałsz	prawda
fałsz lub prawda	prawda
fałsz lub fałsz	fałsz

```
int x = 1;  
int y = 1;  
int z = 2;  
  
x == y && z > y; // true && true => true  
x <= y && z <= x; // true && false => false  
x == y || z > y; // true || true => true  
x <= y || z <= x; // true || false => true
```

Logika matematyczna – zadanie – 15 minut

Zadanie polega na stworzeniu pętli w pseudokodzie dla wymyślonych przez grupę przypadków na spełnienie

- ▶ Koniunkcji
- ▶ Implikacji
- ▶ Alternatywy
- ▶ Negacji
- ▶ Równoważności

symbol logiczny	nazwa zdania złożonego
\wedge	koniunkcja
\vee	alternatywa
\neg lub \sim	zaprzeczenie (negacja)
\Rightarrow	implikacja
\Leftrightarrow	równoważność

p	q	$p \wedge q$	$\neg(p \wedge q)$	$\neg p$	$\neg q$	$(\neg p \vee \neg q)$	$\neg(p \wedge q) \Leftrightarrow (\neg p \vee \neg q)$
1	1	1	0	0	0	0	1
1	0	0	1	0	1	1	1
0	1	0	1	1	0	1	1
0	0	0	1	1	1	1	1

p	q	$p \wedge q$
1	1	1
1	0	0
0	1	0
0	0	0

p	q	$p \vee q$
1	1	1
1	0	1
0	1	1
0	0	0

p	$\neg p$
1	0
0	1

p	q	$p \Leftrightarrow q$
1	1	1
1	0	0
0	1	0
0	0	1

Metody

Metoda – podprogram składowy klasy, którego zadaniem jest działanie na rzecz określonych elementów danej klasy lub klas z nią spokrewnionych

[Wikipedia]

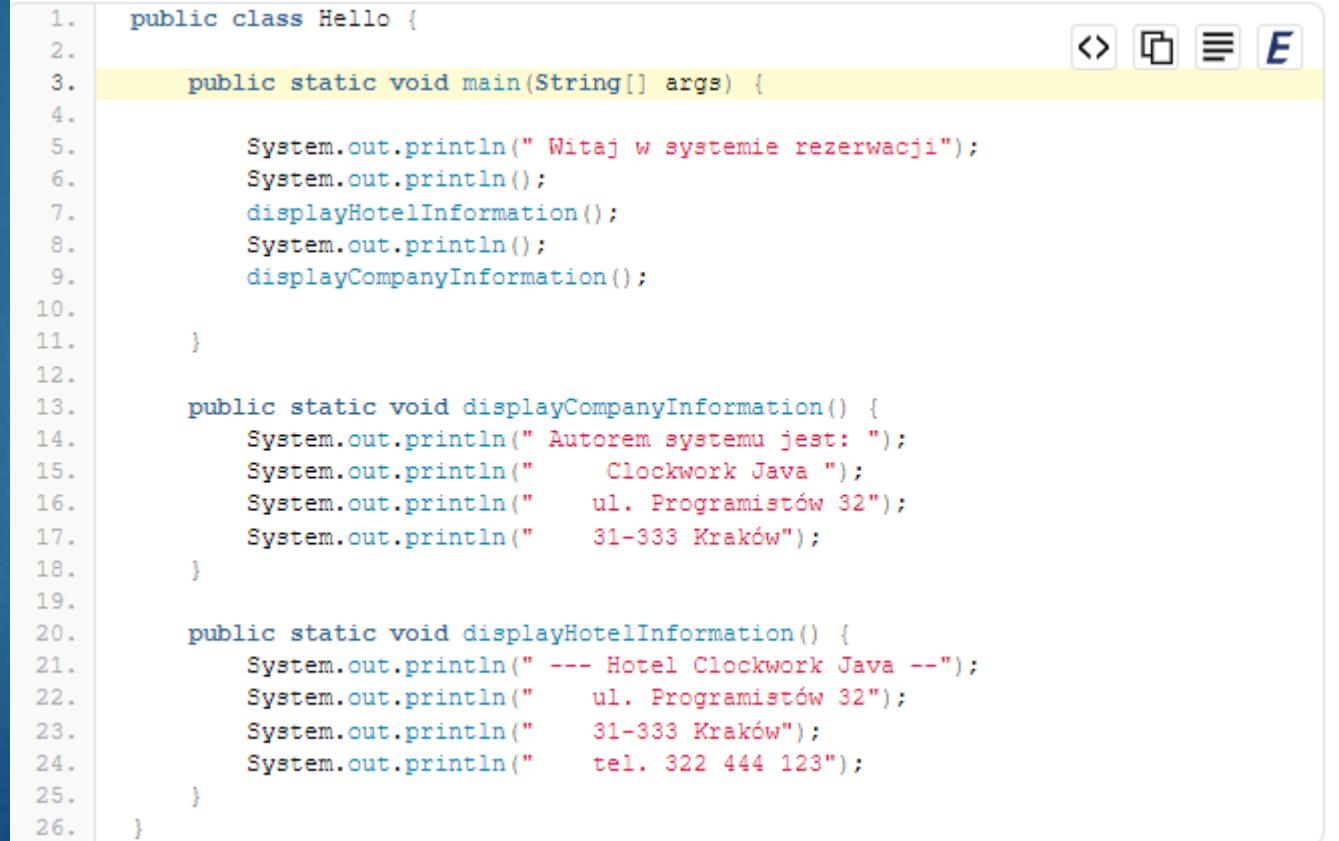
```
boolean isBig(int someNumber) {  
    return someNumber > 100;  
}
```

```
public boolean metodaA() {  
    System.out.println("metodaA");  
    return true;  
}  
  
public boolean metodaB() {  
    System.out.println("metodaB");  
    return false;  
}
```

Metody

Metoda – podprogram składowy klasy, którego zadaniem jest działanie na rzecz określonych elementów danej klasy lub klas z nią spokrewnionych

[Wikipedia]



```
1. public class Hello {
2.
3.     public static void main(String[] args) {
4.
5.         System.out.println(" Witaj w systemie rezerwacji");
6.         System.out.println();
7.         displayHotelInformation();
8.         System.out.println();
9.         displayCompanyInformation();
10.
11.    }
12.
13.    public static void displayCompanyInformation() {
14.        System.out.println(" Autorem systemu jest: ");
15.        System.out.println("      Clockwork Java ");
16.        System.out.println("      ul. Programistów 32");
17.        System.out.println("      31-333 Kraków");
18.    }
19.
20.    public static void displayHotelInformation() {
21.        System.out.println(" --- Hotel Clockwork Java ---");
22.        System.out.println("      ul. Programistów 32");
23.        System.out.println("      31-333 Kraków");
24.        System.out.println("      tel. 322 444 123");
25.    }
26. }
```

Tablica - jednowymiarowa

Tablica – miejsce do przechowywania kolekcji danych – często wykorzystywana do przechowywania zmiennych tego samego typu

```
String[] daysOfWeek = new String[7];
daysOfWeek[0] = "poniedziałek";
daysOfWeek[1] = "wtorek";
daysOfWeek[2] = "środa";
daysOfWeek[3] = "czwartek";
daysOfWeek[4] = "piątek";
daysOfWeek[5] = "sobota";
daysOfWeek[6] = "niedziela";
```

Raz przypisany obiekt w tablicy możemy nadpisać.

```
daysOfWeek[0] = "Monday";
```

```
System.out.println(daysOfWeek[6]);
```

Typy danych

Typy danych – format przechowywania danych

Lista typów

- ▶ Byte
- ▶ Short
- ▶ Int
- ▶ Long
- ▶ Float
- ▶ Double
- ▶ Char
- ▶ Boolean
- ▶ String

Typy danych – zadanie 15 min.

Zadanie – proszę odszukać wszystkie typy danych w Internecie.

Dla każdego przypadku stworzyć tablicę 10 elementową.

Wyniki zadania wysłać na adres e-mail: szguzik@wsb.gda.pl

- ▶ Byte
- ▶ Long
- ▶ Char
- ▶ Short
- ▶ Float
- ▶ Boolean
- ▶ Int
- ▶ Double
- ▶ String

Znaki specjalne

Znak	Nazwa	Opis
{}	Nawias klamrowy	
()	Zwykły nawias	
[]	Nawias kwadratowy	
//	Podwójny ukośnik	
""	Cudzysłów	
;	Średnik	

Znaki specjalne

Znak	Nazwa	Opis
{}	Nawias klamrowy	Oznacza blok obejmujący instrukcje
()	Zwykły nawias	Używany razem z metodami
[]	Nawias kwadratowy	Oznacza tablicę
//	Podwójny ukośnik	Poprzedza komentarz jednowierszowy
""	Cudzysłów	Obejmuje łańcuch znaków (string)
;	Średnik	Oznacza koniec instrukcji

Proces instalacji środowiska Java

- ▶ Pobranie oprogramowania Java
 - ▶ <https://www.java.com/>
- ▶ Wybór wersji

Windows i Które pobieranie wybrać?		
 Windows Online Rozmiar pliku: 2.16 MB	Instrukcje	Po zainstalowaniu oprogramowania Java może być konieczne zamknięcie i ponowne uruchomienie przeglądarki w celu włączenia w niej oprogramowania Java.
 Windows Offline Rozmiar pliku: 73.40 MB	Instrukcje	
 Windows Offline (wersja 64-bitowa) Rozmiar pliku: 84.49 MB	Instrukcje	

Jeżeli są zamiennie używane przeglądarki w wersji 32-bitowej i 64-bitowej, należy - aby w każdej z nich korzystać z oprogramowania Java - zainstalować obie wersje (32-bitową i 64-bitową). » [FAQ — Java w wersji 64-bitowej dla systemu Windows](#)

Mac OS X i FAQ — Mac		
 Mac OS X (wersja 10.7.3 i nowsze) Rozmiar pliku: 85.77 MB	Instrukcje	Po zainstalowaniu oprogramowania Java może być konieczne zamknięcie i ponowne uruchomienie przeglądarki w celu włączenia w niej oprogramowania Java.

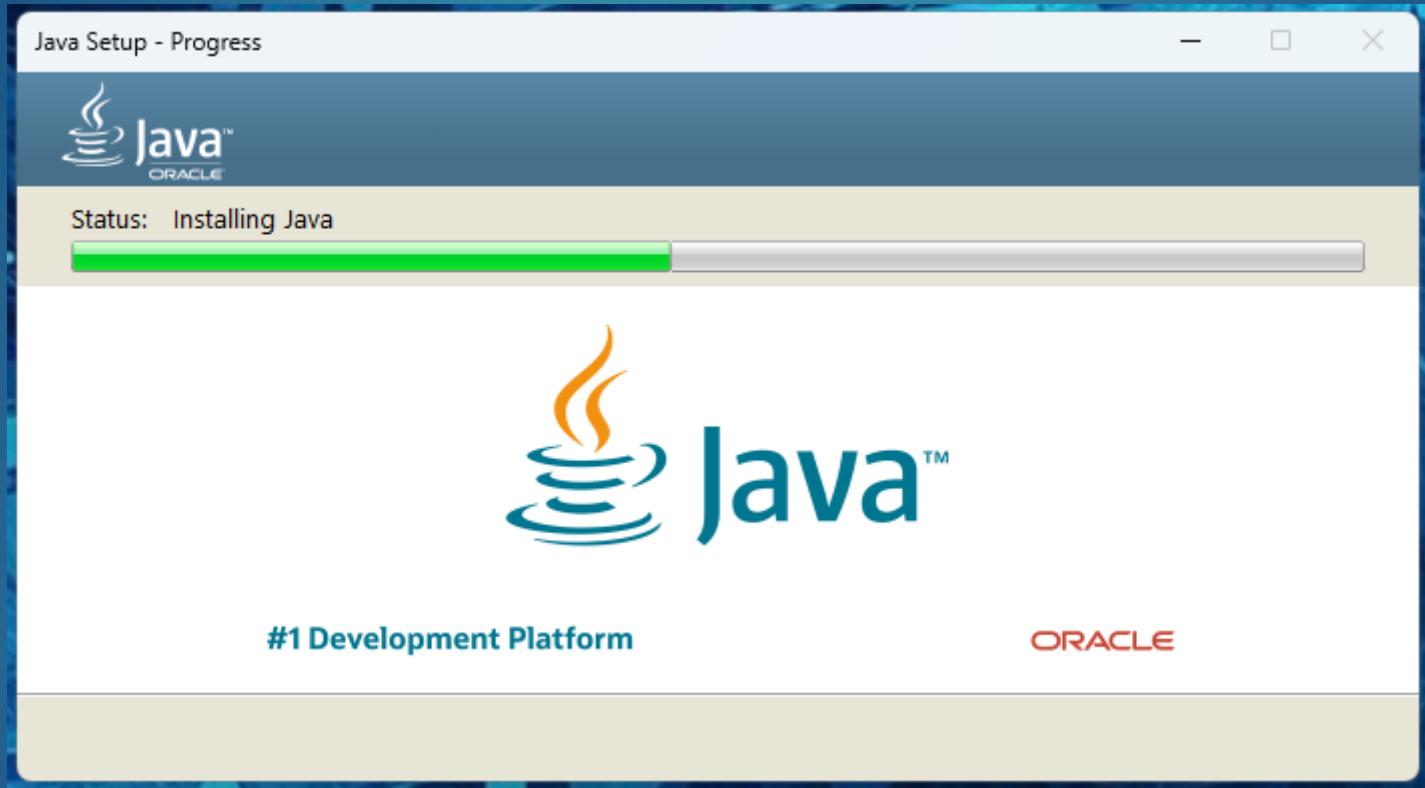
* Oracle Java (wersja 7 i nowsze) wymaga komputera Mac opartego na Intelu z systemem Mac OS X 10.7.3 (Lion) lub wersji nowszej oraz uprawnień administratora do przeprowadzenia instalacji. » [Więcej informacji](#)

Linux			
 Linux RPM	Rozmiar pliku: 63.82 MB	Instrukcje	Po zainstalowaniu oprogramowania Java trzeba włączyć w przeglądarce jego obsługę.
 Linux	Rozmiar pliku: 94.94 MB	Instrukcje	
 Linux x64	Rozmiar pliku: 92.01 MB	Instrukcje	
 Linux x64 RPM	Rozmiar pliku: 61.27 MB	Instrukcje	

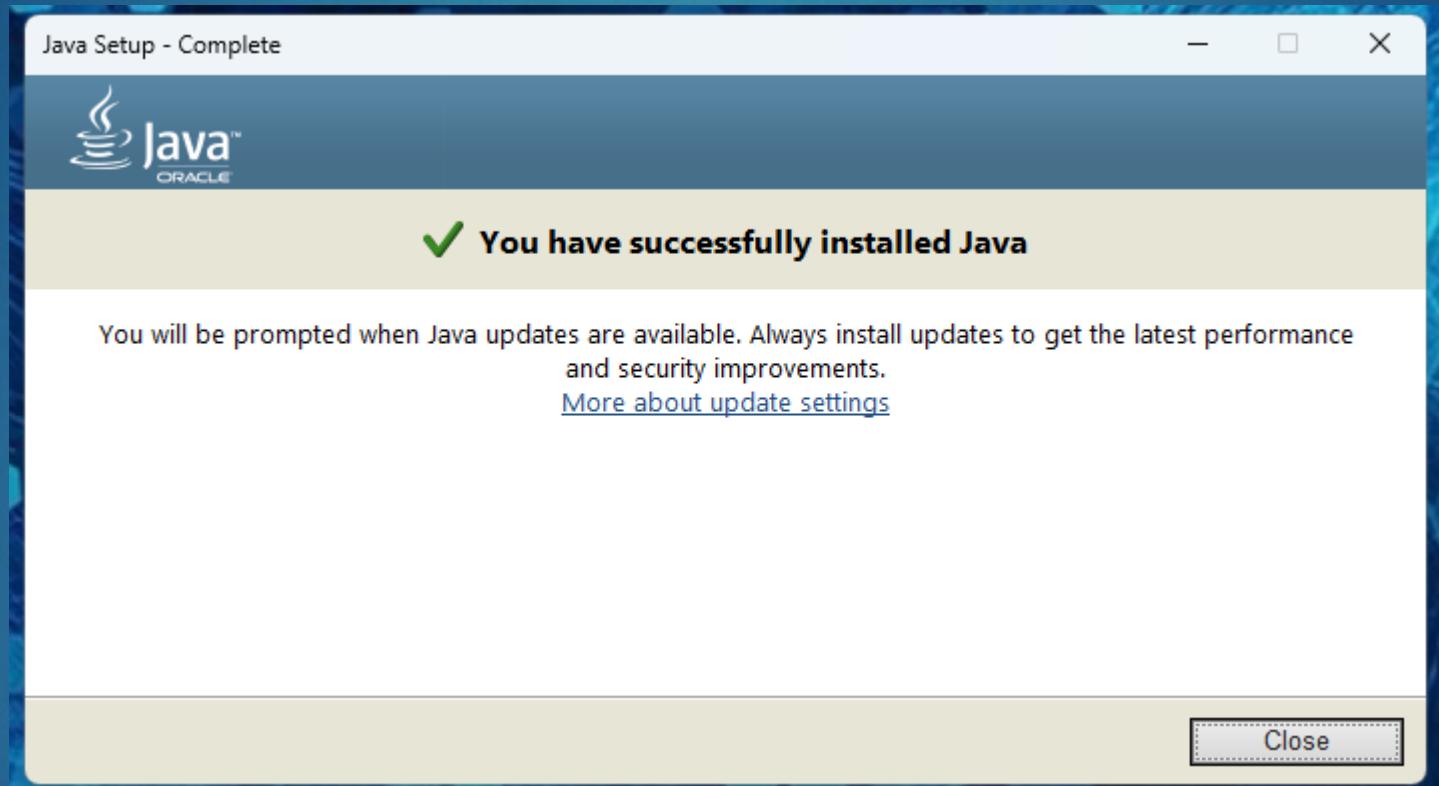
Proces instalacji środowiska Java



Proces instalacji środowiska Java

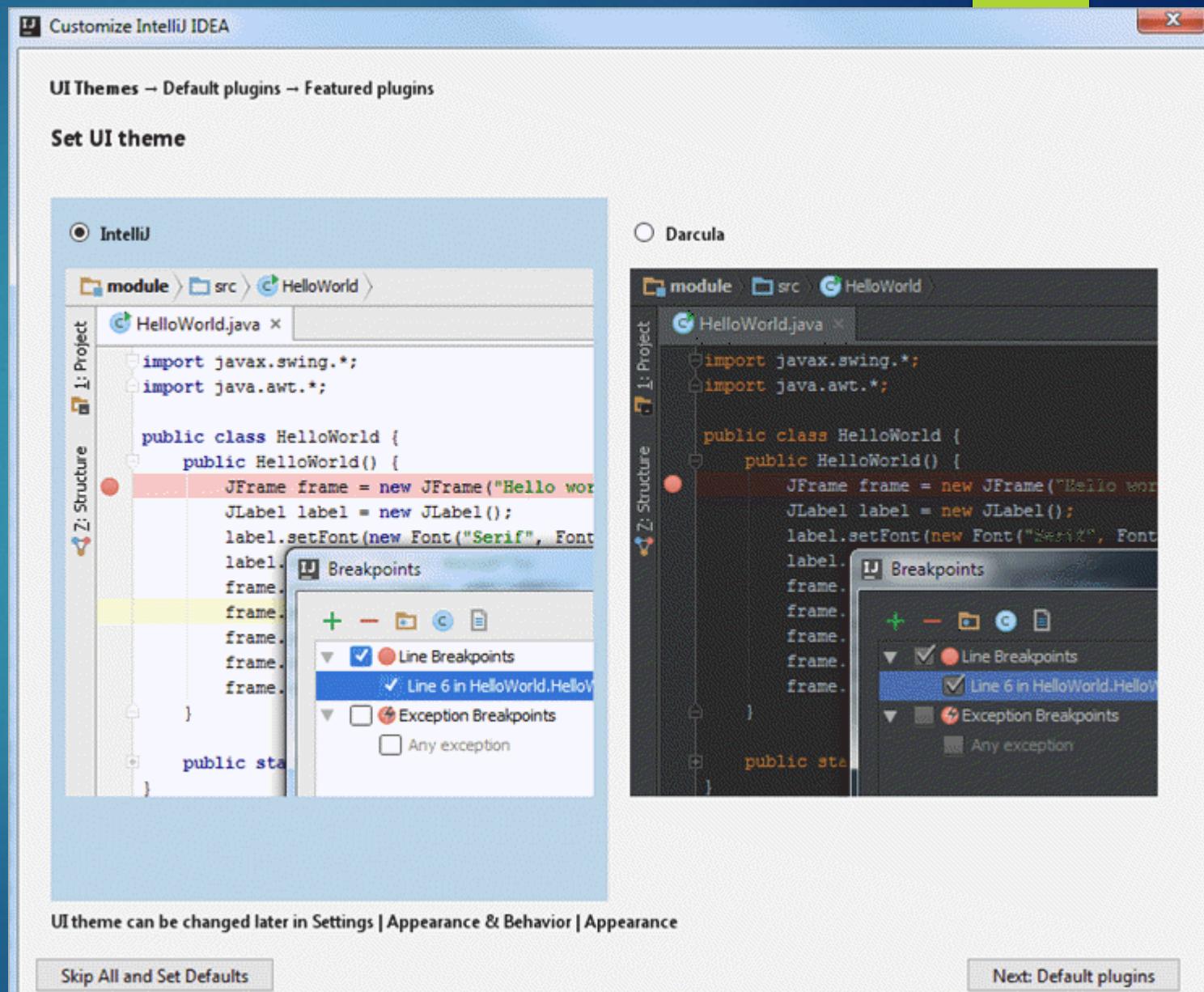


Proces instalacji środowiska Java



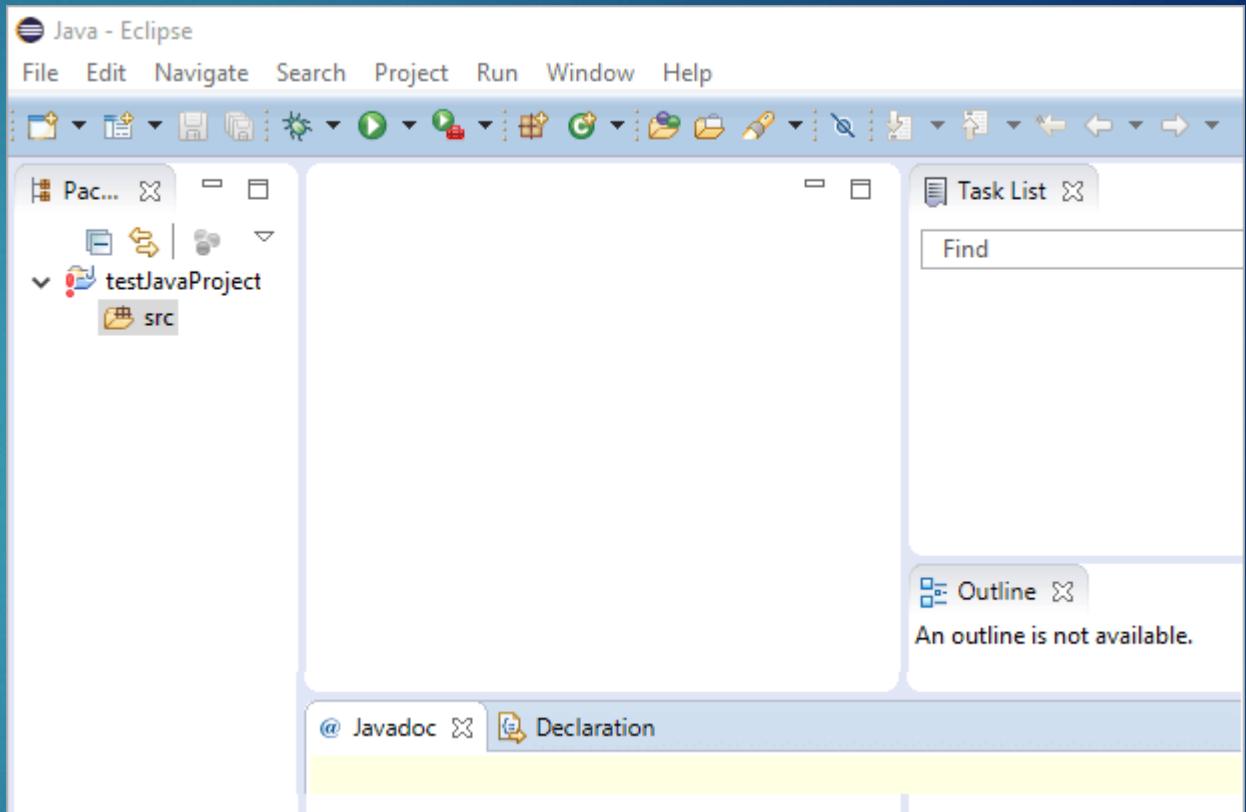
Środowiska IDE

► IntelliJ IDEA



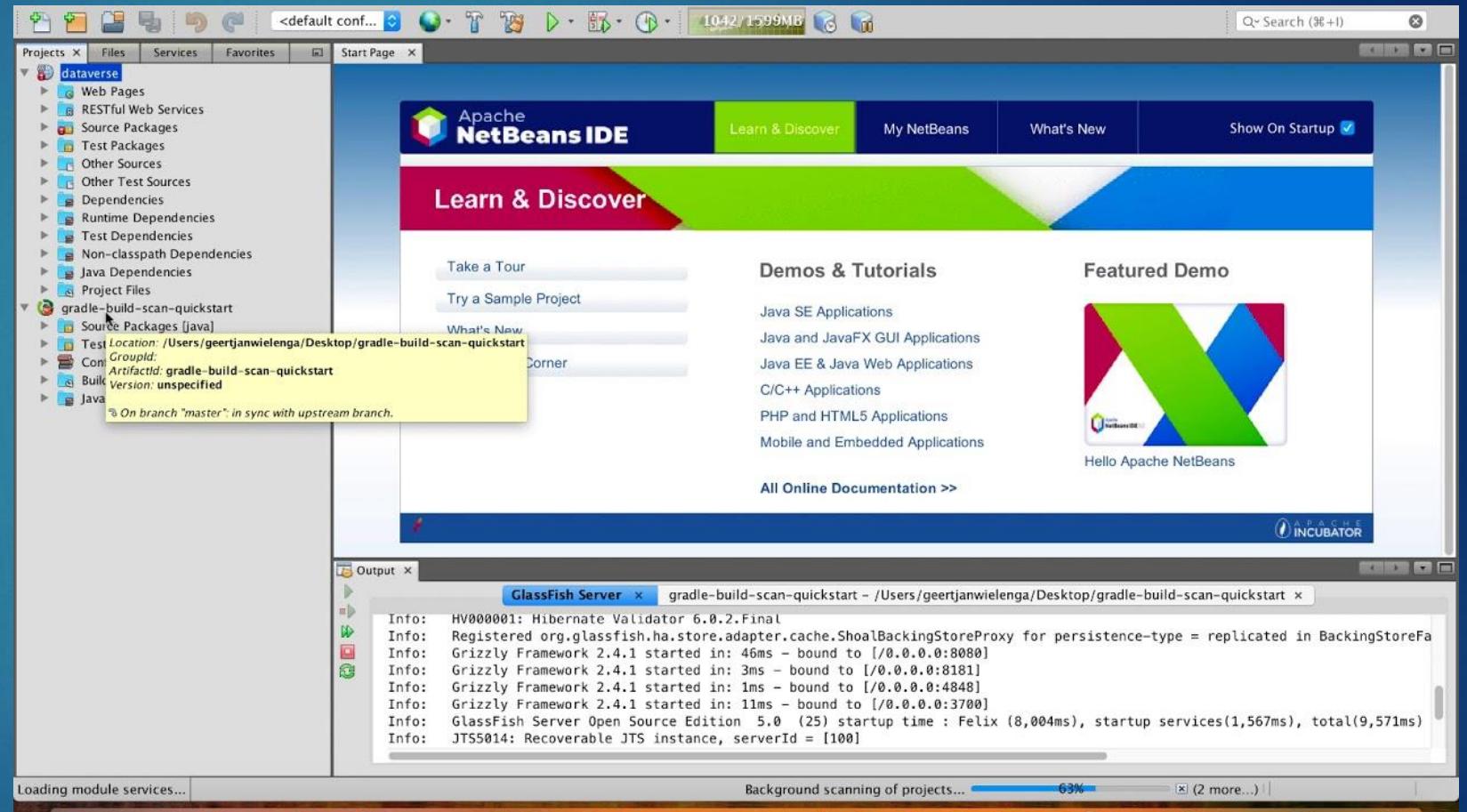
Środowiska IDE

► Eclipse IDE

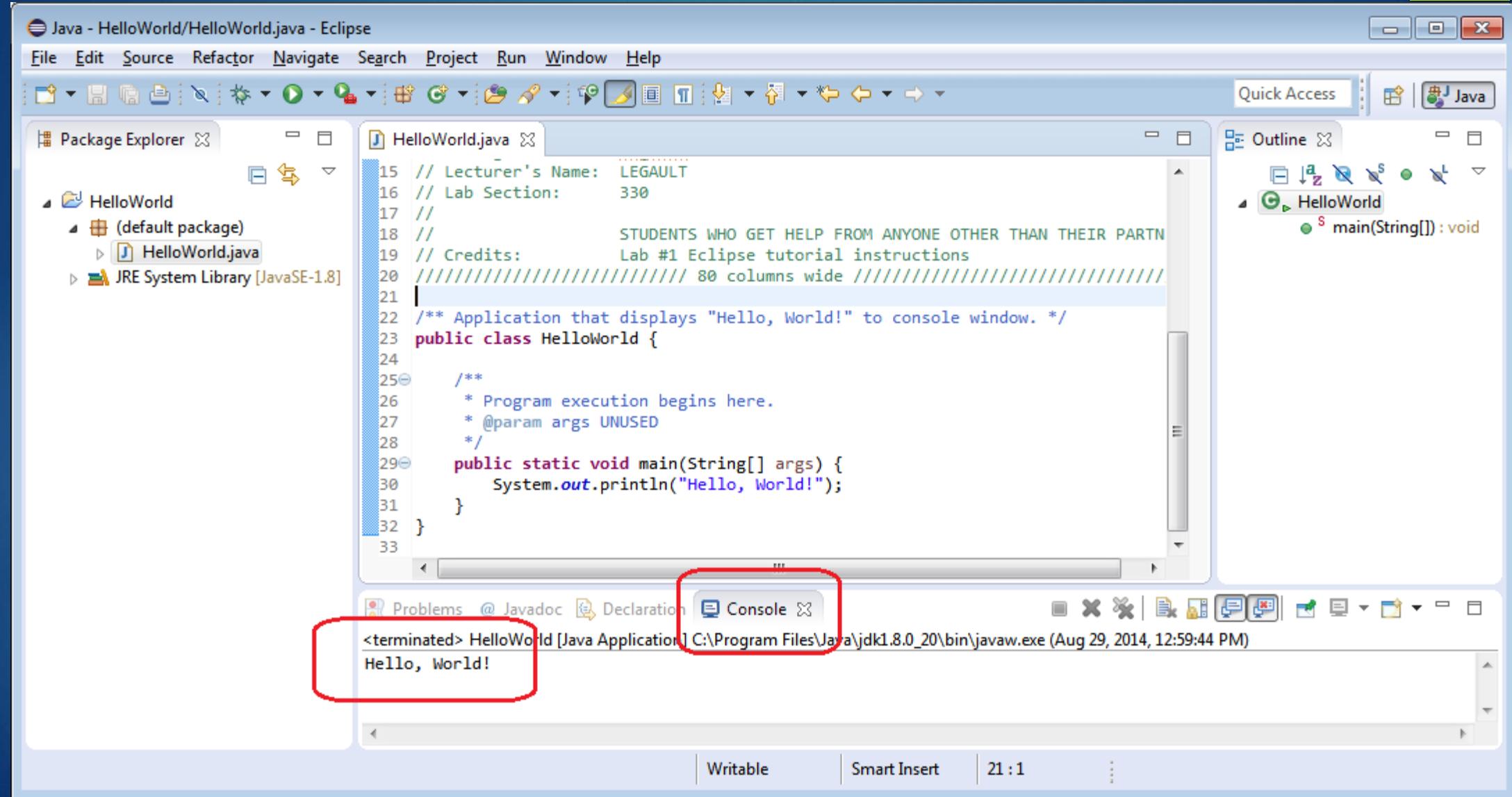


Środowiska IDE

► NetBeans



Uruchomienie pierwszego programu



Tematy projektów

- ▶ Tematy proszę wysłać na adres szguzik@wsb.gda.pl

Instalacja środowiska IntelliJ

jetbrains.com/idea/download/#section=windows

JET BRAINS

Tools Languages Solutions Support Company Store

IntelliJ IDEA

What's New Features Learn Buy Download



Version: 2020.1.1
Build: 201.7223.91
30 April 2020

Release notes

Download IntelliJ IDEA

Windows Mac Linux

Ultimate

For web and enterprise development

Community

For JVM and Android development

Download .exe ▾

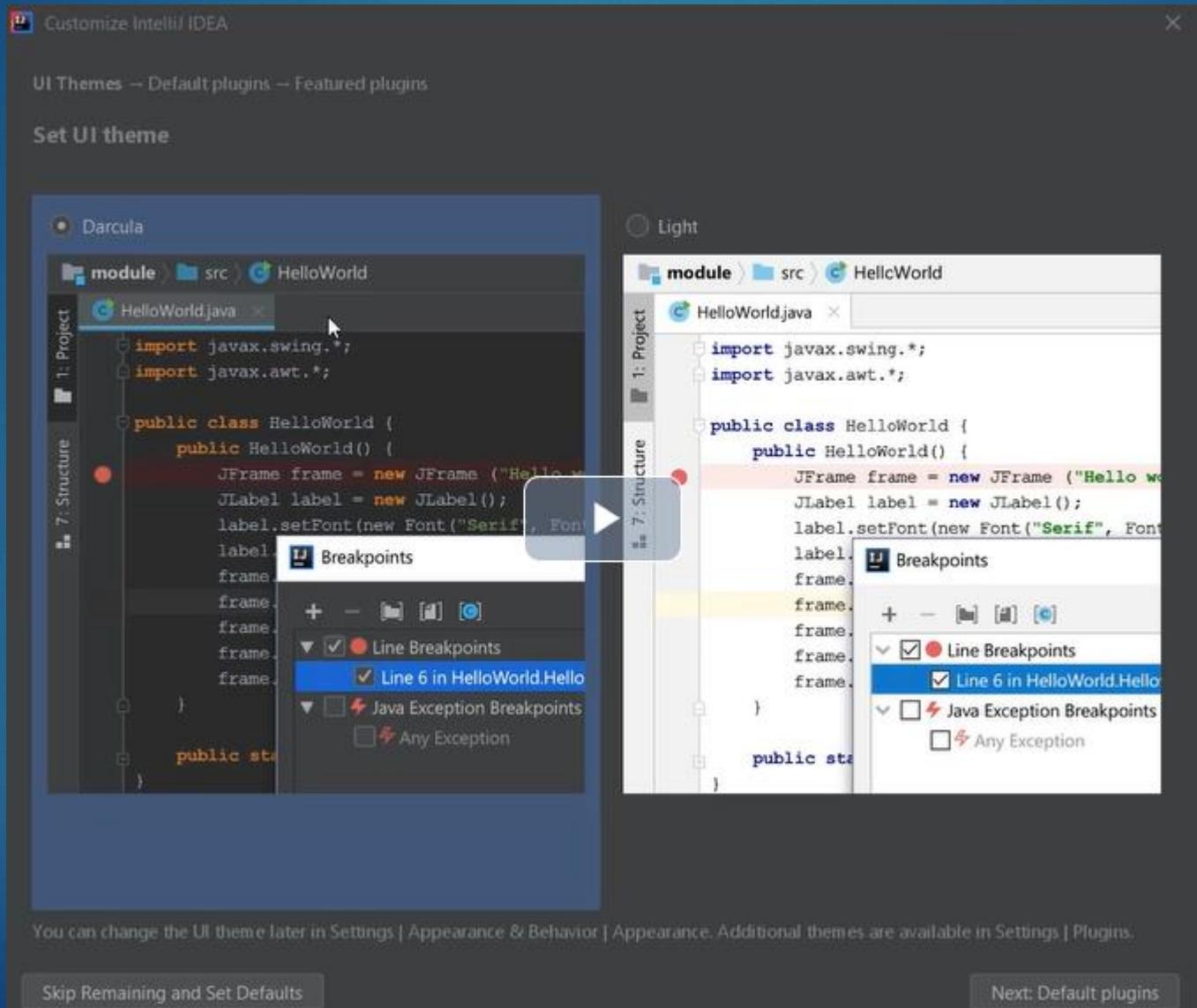
Free trial

Download .exe ▾

Free, open-source

A screenshot of the IntelliJ IDEA download page on the JetBrains website. The page has a dark header with the Jet Brains logo and navigation links. Below the header, there's a large 'IntelliJ IDEA' title and a 'Download' button. On the left, there's a logo for the 2020.1.1 version and some release details. The main section is titled 'Download IntelliJ IDEA' and shows two options: 'Ultimate' (for web and enterprise development) and 'Community' (for JVM and Android development). Each option has a 'Download' button followed by a dropdown menu for file formats (.exe or .tar.gz). Below each download link is a link for a 'Free trial' or 'Free, open-source' version.

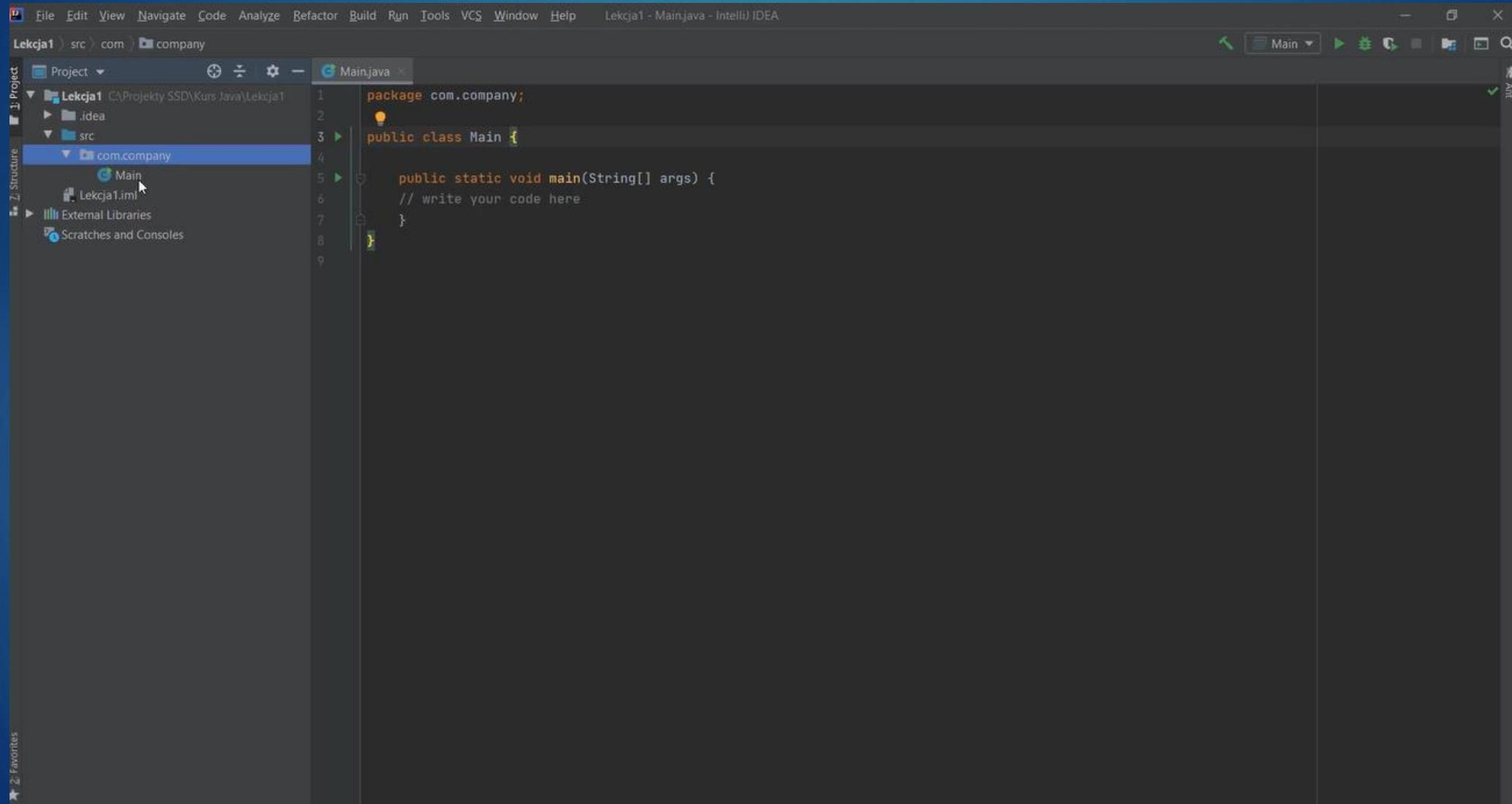
Instalacja środowiska IntelliJ



Instalacja środowiska IntelliJ



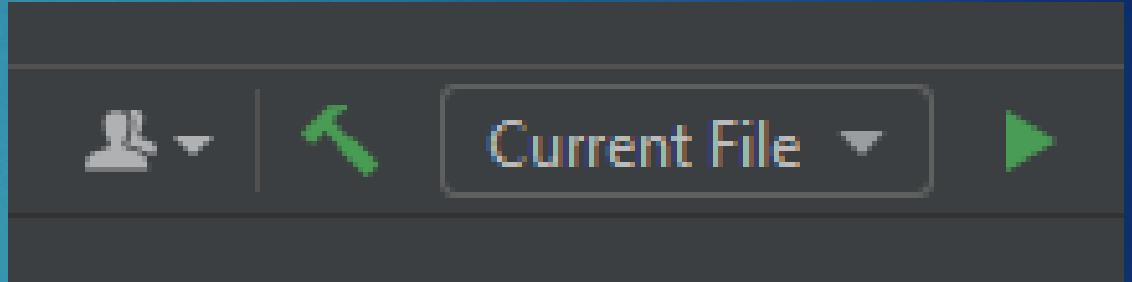
Instalacja środowiska IntelliJ



Uruchamianie programu w konsoli

```
$ javac HelloArg.java
```

```
$ java Main  
Hello World!
```



System.out.print("")

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

System.out.format("")

```
public static void main(String[] args) {  
    System.out.format("Hello %s", args[0]);  
}
```

Tablica stringów !!!

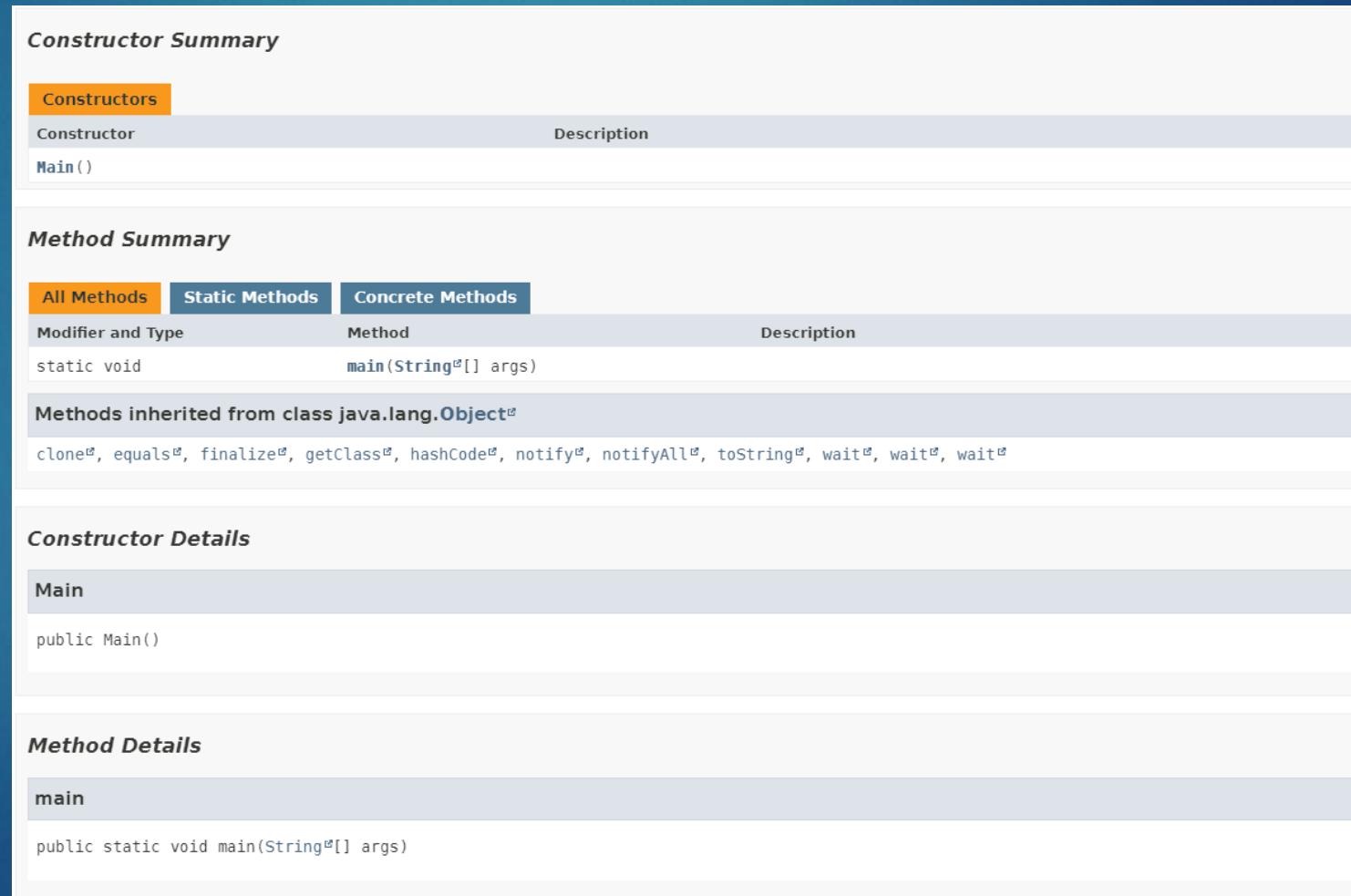
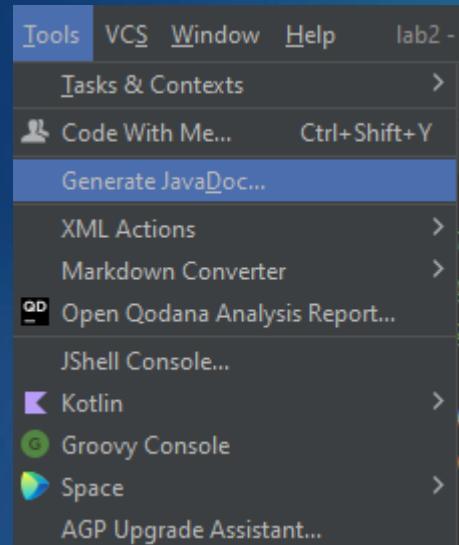
Komentarze

```
System.out.println("Hello World!"); // komentarz jednolinijkowy
```

```
/**  
 * Komentarz  
 * wielolinijkowy  
 */
```

```
1  /**  
2   * Komentarz  
3   * wielolinijkowy  
4  */  
5  ► public class Main {  
6  ►   public static void main(String[] args) {  
7  
8      System.out.println("Hello World!"); // komentarz jednolinijkowy  
9    }  
10 }
```

Komentarze – pomagają w generowaniu dokumentacji



The image shows three generated JavaDoc pages:

- Constructor Summary**: Shows a table with one entry: `Main()`.
- Method Summary**: Shows tabs for `All Methods`, `Static Methods` (selected), and `Concrete Methods`. It lists the `main` method under `Modifier and Type` as `static void` and under `Method` as `main(String[] args)`. It also lists methods inherited from `java.lang.Object` like `clone`, `equals`, etc.
- Constructor Details**: Shows the `Main` constructor with its definition: `public Main()`.
- Method Details**: Shows the `main` method with its definition: `public static void main(String[] args)`.

Zmienne

```
/*
public class Main {
    String zmienka;
    String zmienka2 = "WSB";
    public static void main(String[] args) {
        String uczelnia = "Uczelnia";
        System.out.println(uczelnia);
    }
}
```

Zadania

Utwórz klasę HelloArg. Następnie wewnątrz klasy stwórz metodę main, której argumentem będzie tablica stringów. Wyświetl na ekranie tekst "Hello World"

Zadania

Utwórz klasę HelloArg. Następnie wewnątrz klasy stwórz metodę main, której argumentem będzie tablica stringów.

Wyświetl na ekranie tekst "Hello", który zostanie połączony (konkatenacja) z przekazanym argumentem.

Przykład rozwiązania na ekranie to "Hello Szymon" gdzie "Szymon" jest przekazanym argumentem

Zadania

Utwórz klasę HelloVar. Następnie wewnątrz klasy stwórz metodę main, której argumentem będzie tablica stringów.

Wyświetl na ekranie tekst "Hello", który zostanie połączony (konkatenacja) ze zmiennej typu string. Do zdeklarowanej zmiennej przypisz ciąg znaków w zadeklarowanym typie.

Przykład rozwiązania na ekranie to "Hello Java !" gdzie "Szymon" jest przekazanym argumentem

Stałe

```
public static void main(String[] args) {  
    final String zmienna = "Java";  
}
```



Zadania

Utwórz klasę HelloVar. Następnie wewnątrz klasy stwórz metodę main, której argumentem będzie tablica stringów.

Wyświetl na ekranie tekst "Hello", który zostanie połączony (konkatenacja) ze stałą typu string. Do zdeklarowanej stałej przypisz ciąg znaków w zadeklarowanym typie.

Przykład rozwiązania na ekranie to "Hello Java !" gdzie "Szymon" jest przekazanym argumentem

Zadania

Utwórz klasę HelloVar. Następnie wewnątrz klasy stwórz metodę main, której argumentem będzie tablica stringów.

Wyświetl na ekranie tekst "Hello", który zostanie połączony (konkatenacja) ze stałą typu string. Do zdeklarowanej stałej przypisz ciąg znaków w zadeklarowanym typie. Po wypisaniu na ekran nadpisz stałą oraz wypisz ją ponownie.

Przykład rozwiązania na ekranie to "Hello Java !" gdzie "Szymon" jest przekazanym argumentem

Zadania

Utwórz klasę HelloVar. Następnie wewnątrz klasy stwórz metodę main, której argumentem będzie tablica stringów.

Wyświetl na ekranie tekst "Hello", który zostanie połączony (konkatenacja) ze stałą typu string. Do zdeklarowanej stałej przypisz ciąg znaków w zadeklarowanym typie. Po wypisaniu na ekran nadpisz stałą oraz wypisz ją ponownie.

Przykład rozwiązania na ekranie to "Hello Java,!" gdzie "Szymon" jest przekazanym argumentem

Liczba PI

```
package com.company;

public class Main {

    public static void main(String[] args) {
        final String JĘZYK = "Java";

        System.out.format("Hello PI: %f", Math.PI);
    }
}
```

Hello PI: 3,141593|

Typy danych ponownie

```
public static void main(String[] args) {  
    /* LICZBOWE - CAŁKOWITE */  
    byte zmByte = Byte.MAX_VALUE;  
    short zmShort = Short.MAX_VALUE;  
    int zmInt = Integer.MAX_VALUE;  
    long zmLong = Long.MAX_VALUE;  
    zmLong = 2356789L;
```

Typy danych ponownie

```
/* LICZBOWE - ZMIENNOPRZECINKOWE */  
float zmFloat = 25.525F;  
double zmDouble = 50.782D;
```

Typy danych ponownie

```
/* LOGICZNE */  
boolean zmBool = false;    I
```

```
/* LOGICZNE */  
boolean zmBool = true;
```

Typy danych ponownie

```
/* ZNAKOWE */
char zmChar = '9';
```

```
/* ZNAKOWE */
char zmChar = 'A';
```

Typy danych ponownie

```
/* ZNAKOWE */
char zmChar = '9';
```

```
/* ZNAKOWE */
char zmChar = 'A';
```

```
String zmString = "Tekst";
```

Zadanie

Napisz metodę przedstawiającą podstawowe operacje arytmetyczne z domyślnie przypisanymi logikami. Pamiętaj o deklaracji typu zmiennej. Każde działanie wypisz na ekran

Zadanie - rozwiążanie

```
public class Main {  
  
    public static void main(String[] args) {  
        int dodawanie = 2 + 2;  
        int odejmowanie = 3 - 2;  
        int mnozenie = 2 * 3;  
        int dzielenie = 6 / 2;  
        int reszta = 8 % 3;  
  
        System.out.format("Wynik dodawania: %d\n", dodawanie);  
        System.out.format("Wynik odejmowania: %d\n", odejmowanie);  
        System.out.format("Wynik mnożenia: %d\n", mnozenie);  
        System.out.format("Wynik dzielenia: %d\n", dzielenie);  
        System.out.format("Reszta z dzielenia: %d\n", reszta);  
    }  
}
```

Potęgowanie

```
System.out.println(Math.pow(2, 3));
```

Wprowadzanie danych przez użytkownika

```
package com.company;

import java.util.Scanner;

public class WprowadzDane {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String a;
        a = scanner.nextLine();
        System.out.print("Wprowadzono " + a);
    }
}
```

Zadanie

Używając Scanner napisz prostą metodę na sumę dwóch liczb.
Poczytaj o **klasach osłonowych** i wykonaj powyższe zadanie

Zadanie

```
package com.company;

import java.util.Scanner;

public class Dodawanie {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int a, b;
        System.out.print("Podaj a ");
        a = Integer.parseInt(scanner.nextLine());
        System.out.print("Podaj b ");
        b = Integer.parseInt(scanner.nextLine());

        System.out.print("Suma " + (a + b));
    }
}
```

Klasa osłonowa

Klasy osłonowe

Ponieważ w Javie **typy proste**, czyli int, double, float, czy boolean nie są typami obiektowymi, przez co nie można ich na nich wywoływać żadnych metod. Na szczęście istnieją **klasy osłonowe**, które dają nam sporo możliwości - przykładowo w łatwy sposób zmienić reprezentację liczb w systemie dziesiętnym na dwójkowy, czy szesnastkowy, lub zamienić je na Stringi. Jest to szczególnie przydatne, gdy przykładowo odczytujemy jakieś dane w postaci Stringów od użytkownika, bądź z plików, a chcielibyśmy uzyskać niektóre dane w formacie liczbowym.

Klasy osłonowe to odpowiednio:

- Boolean - boolean
- Character - char
- Integer - int
- Double - double
- Float - float
- Byte - byte
- Short - short
- Long - long

Klasa osłonowa

Niektóre z metod są statyczne, dzięki czemu dostęp do nich jest ułatwiony i można je wywołać w schematyczny sposób:

```
NazwaKlasy.nazwaMetody(parametr);
```

Przykładowo:

```
Integer.toString(5);
```

Wykorzystanie klas osłonowych nietrudno sobie wyobrazić. Jeśli chcemy odczytać dane od użytkownika przy pomocy buforowanego strumienia i wiemy, że będą tam dane różnego typu to należy użyć klasy osłonowej, aby zamienić ją z typu String na typ liczbowy należy użyć **parsera**, czyli metody `parseInt()`, `parseDouble()` itp.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Test{
    public static void main(String[] args) throws NumberFormatException, IOException{
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Podaj imię: ");
        String imie = br.readLine();

        System.out.println("Podaj wiek: ");
        int wiek = Integer.parseInt(br.readLine());
    }
}
```

Kopiuj do schowka

Klasa osłonowa

W przypadku metod typu parse należy pamiętać o zgłaszanym przez nie wyjątku typu NumberFormatException. Jest on zgłaszany w przypadku kiedy użytkownik wprowadzi przykładowo jakieś znaki alfanumeryczne w momencie pytania o wiek. Możemy go oczywiście złychać w bloku try catch i poprosić o ponowne wprowadzenie danych, co wydaje się być rozsądny posunięciem.

Jeszcze ciekawsze możliwości daje nam klasa **Character**. Najpopularniejsze jej metody to:

- boolean isDigit() - sprawdza, czy podany argument w postaci znaku jest liczbą
- char toLowerCase(char) / toUpperCase(char) - zwracają podany znak odpowiednio w postaci małej, lub wielkiej litery
- boolean isWhiteSpace(char) - sprawdza, czy podany znak jest białym znakiem

Klasa osłonowa

Zobaczmy przykładowe działanie:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Test{
    public static void main(String[] args) throws NumberFormatException, IOException{
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Podaj cyfrę od 0 do 9: ");
        String str = br.readLine();
        int liczba = 0;
        if(str.length()==1 && Character.isDigit(str.charAt(0))){
            System.out.println("Hurra podałeś cyfrę");
            liczba = Integer.parseInt(str);
        }
        else{
            System.out.println("niestety podałeś nieprawidłowe dane");
        }
    }
}
```

Kopiuj do schowka

Konwersja / rzutowanie zmiennych na inny typ danych

Rzutowanie (jeżeli dotyczy liczb)

Nawias jest zbędny

Mniejszy typ danych na większy

```
/* Rzutowanie niejawne */  
short zmShort = 125;  
int zmInt;  
zmInt = (int)zmShort;
```

```
zmInt = zmShort;
```

Większy typ danych na mniejszy

```
/* Rzutowanie jawne */  
float zmFloat = 3.14F;  
long zmLong;  
zmLong = (long)zmFloat;
```

**Nawias jest wymagany!!!
– możliwa utrata części danych**

Konwersja / rzutowanie zmiennych na inny typ danych

Konwersja danych (Liczba na string)

```
short zmShort = 125;  
int zmInt;  
zmInt = (int)zmShort;
```

Przekonwertowanie z typu liczbowego
(przez przeciążenie)

```
String tekst = String.valueOf(zmInt);
```

Przekonwertowanie z typu liczbowego
(przez klasę osłonową)

```
tekst = Long.toString(zmInt);
```

```
Float.parseFloat(zmInt)
```

Konwersja / rzutowanie zmiennych na inny typ danych

Konwersja danych (String na liczbę)

```
String tekst2 = "123.5";
```

```
String tekst2 = "123,5";
```

Przekonwertowanie z typu liczbowego
(przez klasę osłonową)

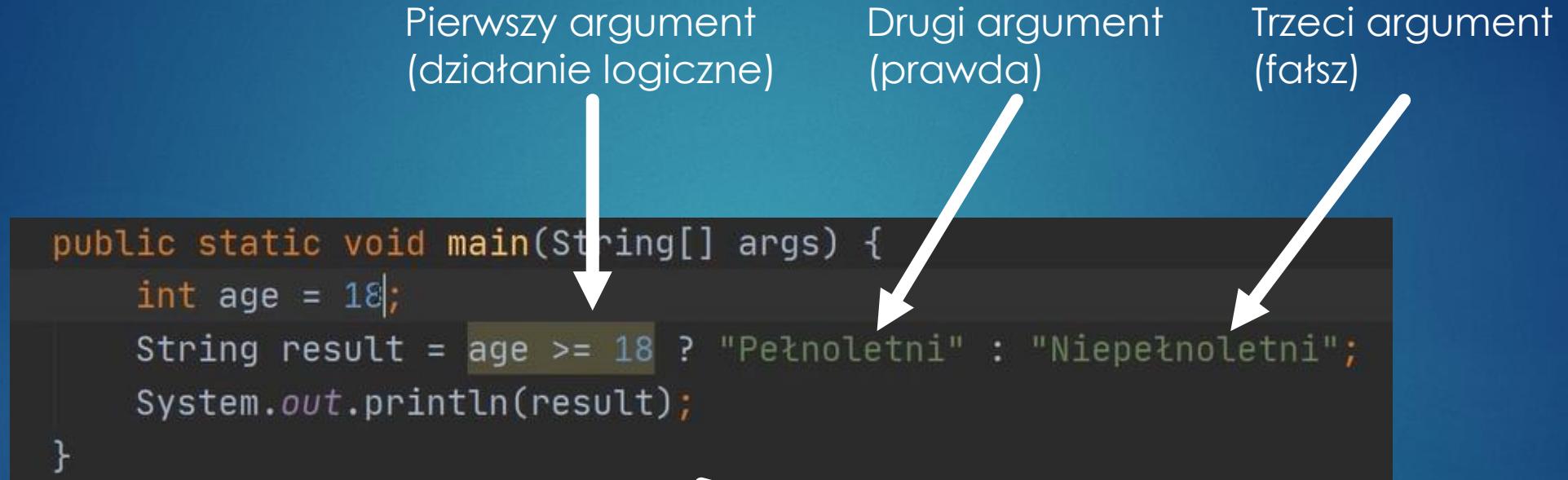
```
float zmFloat2 = Float.parseFloat(tekst2);  
System.out.println(zmFloat2 + 100);
```

223.5

```
float zmFloat2 = Float.parseFloat(tekst2);  
System.out.println(zmFloat2 + 100);
```

```
Exception in thread "main" java.lang.NumberFormatException: For input string: "123,5"  
at sun.misc.FloatingDecimal.readJavaFormatString(FloatingDecimal.java:2043)  
at sun.misc.FloatingDecimal.parseFloat(FloatingDecimal.java:122)  
at java.lang.Float.parseFloat(Float.java:451)  
at com.company.Main.main(Main.java:26)
```

Trójargumentowy operator



Pełnoletni

Switch (instrukcja sterująca)

```
String namedDay = "Poniedziałek";  
  
switch (namedDay) {  
    case "Poniedziałek":  
        System.out.println("1 dzień tygodnia");  
        break;  
    case "Wtorek":  
        System.out.println("2 dzień tygodnia");  
        break;  
    default:  
        System.out.println("Inny dzień tygodnia");  
}
```

Funkcja dopasowująca (porównująca)

```
switch (day) {  
    case 1:  
        System.out.println("Poniedziałek");  
        break;  
    case 2:  
        System.out.println("Wtorek");  
        break;  
    case 3:  
        System.out.println("Środa");  
        break;  
    case 6:  
    case 7:  
        System.out.println("Weekend");  
        break;  
    default:  
        System.out.println("Inny dzień");  
}
```

Pętla - while

Pętla dopuszcza nie wykonanie ani jednego przebiegu w momencie, kiedy od razu warunek jest fałszywy

```
public class Main {  
  
    public static void main(String[] args) {  
  
        int i = 0;  
  
        while (i < 10) {  
            System.out.println(i);  
            i++;|  
        }  
    }  
}
```

Pętla - do while

Pętla wykona się co najmniej
Jeden raz

```
public static void main(String[] args) {  
  
    int i = 0;  
  
    do {  
        System.out.println(i);  
        i++;  
    } while (i < 10);
```

Średnik na końcu pętli

Pętla - For

Pętla dopuszcza nie wykonanie ani jednego przebiegu w momencie, kiedy od razu warunek jest fałszywy

// PSEUDOKOD //
np: for (int i = 10; i < 10; i++)
{
 sout(i) // pseudokod
}

```
public class Main {  
  
    public static void main(String[] args) {  
  
        for (int i = 0 ; i < 10 ; i++) {  
            System.out.println(i);  
        }  
    }  
}
```

Zagnieżdżanie pętli

```
public static void main(String[] args) {  
  
    for (int i = 1; i <= 10; i++) {  
        for (int j = 1; j <= 10; j++){  
            System.out.println(i);  
            System.out.println(j);  
        }  
    }  
}
```

Zagnieżdżanie pętli - zadanie

Proszę napisać tabliczkę mnożenia – całą zawartość wypisać na ekranie

$$1 * 1 = 1$$

Zagnieżdżanie pętli – zadanie - rozwiązywanie

Proszę napisać tabliczkę mnożenia – całą zawartość wypisać na ekranie

```
public static void main(String[] args) {  
  
    for (int i = 1; i <= 10; i++) {  
        for (int j = 1; j <= 10; j++){  
            System.out.println(Integer.toString(i) + " * " + Integer.toString(j) + " = " + (i * j));  
        }  
    }  
}
```

1 * 9 = 9
1 * 10 = 10
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16

5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
6 * 1 = 6
6 * 2 = 12

10 * 3 = 30
10 * 4 = 40
10 * 5 = 50
10 * 6 = 60
10 * 7 = 70
10 * 8 = 80
10 * 9 = 90
10 * 10 = 100

Instrukcje skoku

```
public static void main(String[] args) {  
  
    int i = 0;  
  
    while (true) {  
        i++;  
        System.out.println(i);  
    }  
}
```

```
public static void main(String[] args) {  
  
    int i = 0;  
  
    while (true) {  
        i++;  
        System.out.println(i);  
        if (i >= 10) {  
            break;  
        }  
    }  
}
```

**Słówko skoku
nakazujące przerwanie
pętli – w tym przypadku
pętli nieskończonej**

Instrukcje skoku

```
if (i % 2 == 1) {  
    continue;  
}
```

Słówko skoku
nakazujące przerwanie
pętli – w tym przypadku
pętli dla liczb
nieparzystych

Instrukcje - zadanie

Proszę napisać program, który wypisze

- Liczby parzyste od 1 do 10
- Wykorzysta instrukcję skoku
- LOGIKA MA ZOSTAĆ NAPISANA WEWNĄTRZ FUNKCJI NIESKOŃCZONOŚCI

Instrukcje – zadanie - rozwiązywanie

Proszę napisać program, który wypisze

- Liczby parzyste od 1 do 10
- Wykorzysta instrukcję skoku
- LOGIKA MA ZOSTAĆ NAPISANA WEWNĄTRZ FUNKCJI NIESKOŃCZONOŚCI

```
while (true) {  
    i++;  
    if (i % 2 == 1) {  
        continue;  
    }  
    System.out.println(i);  
    if (i >= 10) {  
        break;  
    }  
}
```



```
2  
4  
6  
8  
10
```

Zadanie

- Komputer losuje liczbę z przedziału 1 - 10
- Gracz zgaduje wylosowaną liczbę
- Jeżeli gracz nie odgadł liczby to:
 - komputer podaje wskazówkę, czy wylosowana liczba jest mniejsza czy większa od podanej
 - gracz wraca do punktu 2
- Jeżeli gracz odgadł liczbę to:
 - komputer wyświetla za którym razem zgadł

Zadanie - rozwiążanie

- Komputer losuje liczbę z przedziału 1 - 10
- Gracz zgaduje wylosowaną liczbę
- Jeżeli gracz nie odgadł liczby to:
 - komputer podaje wskazówkę, czy wylosowana liczba jest mniejsza czy większa od podanej
 - gracz wraca do punktu 2
- Jeżeli gracz odgadł liczbę to:
 - komputer wyświetla za którym razem zgadł

```
package com.company;

import java.util.Random;
import java.util.Scanner;

public static void main(String[] args) {
    Random rnd = new Random();
    int rndNumber = rnd.nextInt( bound: 10 ) + 1;
    Scanner scanner = new Scanner(System.in);
    int i = 0;
    int userNumber;
    do {
        System.out.print("Zgadnij liczbę z przedziału 1 - 10: ");
        userNumber = scanner.nextInt();
        i++;
        if (userNumber < rndNumber) {
            System.out.println("Twoja liczba jest za mała");
        } else if (userNumber > rndNumber) {
            System.out.println("Twoja liczba jest zbyt duża");
        }
    } while (userNumber != rndNumber);
    System.out.println("BRAWO! Odgadłeś za " + i + " razem!");
}
```

Tablice

```
int[] array =
```

Wartości:	5	9	7	1	2
Index:	0	1	2	3	4

```
array[0] - 5
```

```
array[4] = 2
```

```
Długość (.length) = 5
```

Tablice

```
int[] arr = new int[5];
arr[0] = 5;
arr[1] = 2;
arr[2] = 7;
arr[3] = 8;
arr[4] = 9;
```

```
String[] colors = {"red", "green", "blue"};
System.out.println(colors[0]);
```

Wypełnianie tablicy



Tablice – pętla For Each

```
package com.company;

public class Main {

    public static void main(String[] args) {
        String[] colors = {"red", "green", "blue", "black", "white"};
        System.out.println("For:");
        for (int i = 0; i < colors.length; i++) {
            System.out.println(colors[i]);
        }

        System.out.println("For each:");
        for (String item : colors) {
            System.out.println(item);
        }
    }
}
```



Tablice - wielowymiarowe

- Tablice prostokątne:

int[][] array =

Index:	0	1	2	3	4
0	2	4	5	1	6
1	1	3	7	8	9
2	1	8	9	2	3

- Tablice nieprostokątne: **wyszczerbione**

int[][] array =

Index:	0	1	2	3	4
0	2	4	5		
1	1	3	7	8	9
2	1	8			

Tablice - wielowymiarowe prostokątne

```
int[][] arr = new int[3][5];
arr[0][4] = 9;
for (int[] items : arr) {
    for (int item : items) {
        System.out.print(item);
    }
    System.out.println();
}
```



0	0	0	0	9
0	0	0	0	0
0	0	0	0	0

Tablice – wielowymiarowe wyszczerbione

```
public static void main(String[] args) {  
  
    int[][] arr = new int[3][];  
    arr[0] = new int[5];  
    arr[1] = new int[2];  
    arr[2] = new int[7];  
    arr[0][4] = 9;  
  
    for (int[] items : arr) {  
        for (int item : items) {  
            System.out.print(item);  
        }  
        System.out.println();  
    }  
}
```

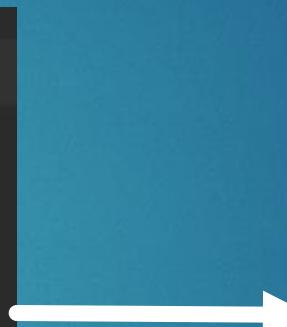


```
0 0 0 0 9  
  
0 0  
  
0 0 0 0 0 0 0
```

Tablice – wielowymiarowe wyszczerbione

```
int[][] arr2 = {{1, 4, 5}, {2, 5, 7, 8}, {2, 1}};

for (int[] items : arr2) {
    for (int item : items) {
        System.out.print(item);
    }
    System.out.println();
}
```



145
2578
21

Funkcje

Funkcje

```
[zwracany typ] nazwa_funkcji(typ argument1, typ argument2) {  
    int wynik = argument1 + argument2;  
    return wynik;  
}
```

Funkcje

```
x = nazwa_funkcji(5, 3);
```

```
y = nazwa_funkcji (3, 12);
```

Funkcje

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

Funkcje

```
public class Main {  
  
    public static void main(String[] args) {  
        hello();  
        hello();  
        hello();  
        hello();  
    }  
  
    public static void hello() {  
        System.out.println("Hello World!!!!");  
        return; //  
        System.out.println("I'm alive");  
    }  
}
```

Funkcje – argumenty funkcji

```
public class Main {  
  
    public static void main(String[] args) {  
        add( a: 2, b: 5 );  
    }  
  
    public static void add(int a, int b) {  
        System.out.println("Suma = " + (a + b));  
    }  
}
```

Funkcje – argumenty funkcji

```
int[] a = {10};  
change(a);  
System.out.println("a = " + a[0]);  
}  
  
@  
public static void change(int[] x) {  
    x[0] = 0;  
    x[0]--;  
    System.out.println("x = " + x[0]);  
}
```

Funkcje – przeciążenie nazw funkcji

Przeciążanie

- funkcja o tej samej nazwie
- różniąca się ilością argumentów lub typem danych jakie przyjmuje

6
10

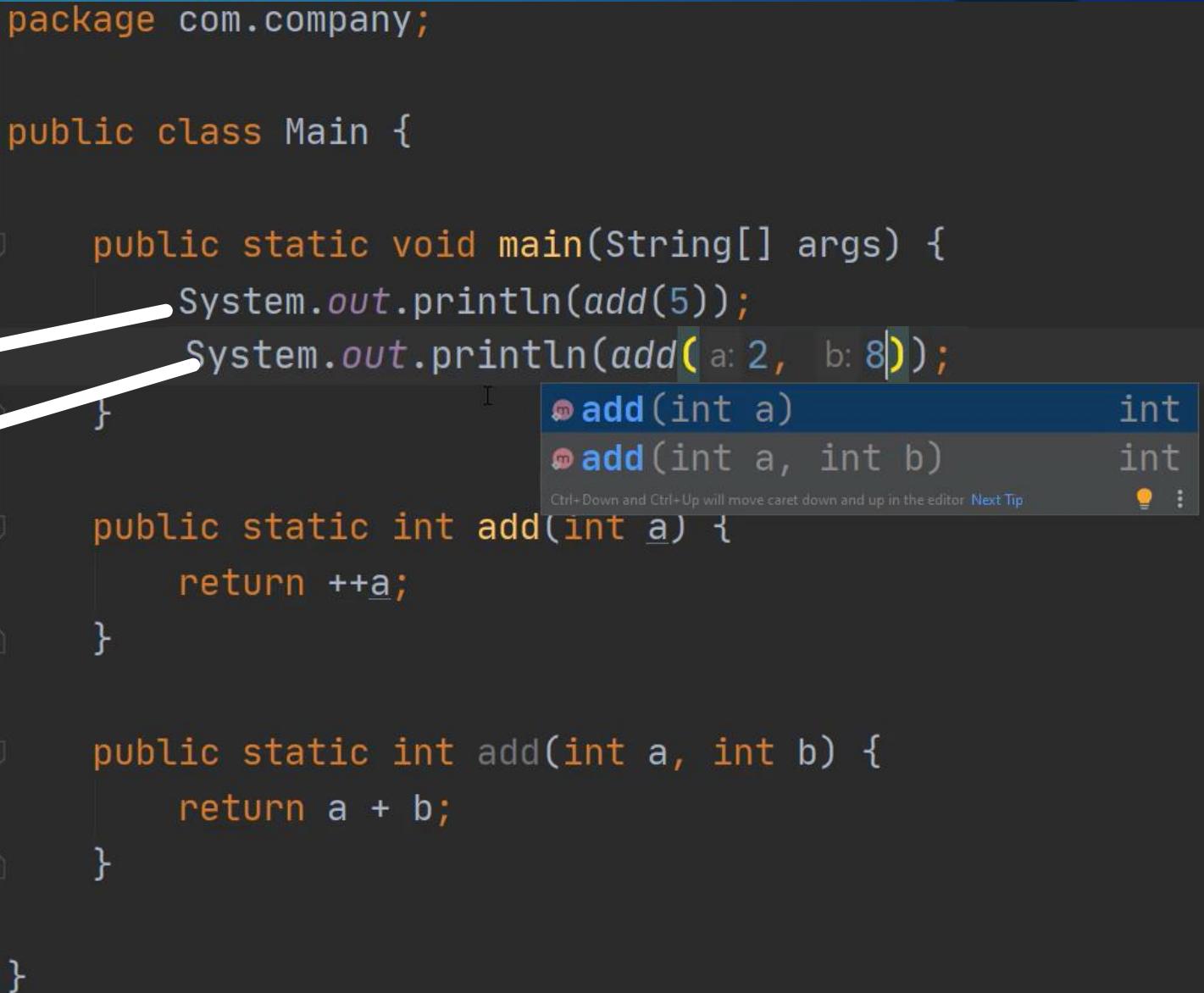
```
package com.company;

public class Main {

    public static void main(String[] args) {
        System.out.println(add(5));
        System.out.println(add( a: 2, b: 8));
    }

    public static int add(int a) {
        return ++a;
    }

    public static int add(int a, int b) {
        return a + b;
    }
}
```



Funkcje – przeciążenie nazw funkcji

Przeciążanie

- funkcja o tej samej nazwie
- różniąca się ilością argumentów lub typem danych jakie przyjmuje

6
4.0
10

```
public static void main(String[] args) {  
    System.out.println(add(5));  
    System.out.println(add(5.0F));  
    System.out.println(add( a: 2, b: 8));  
}  
  
public static float add(float a) {  
    return --a;  
}  
  
public static int add(int a) {  
    return ++a;  
}  
  
public static int add(int a, int b) {  
    return a + b;  
}
```

Funkcje – przeciążenie nazw funkcji

Przeciążanie

- funkcja o tej samej nazwie
- różniąca się ilością argumentów lub typem danych jakie przyjmuje

```
public static int add(float a) {  
    return (int)--a;  
}
```

Funkcje – przeciążenie nazw funkcji

Przeciążanie

- funkcja o tej samej nazwie
- różniąca się ilością argumentów lub typem danych jakie przyjmuje

Nie można przeciążyć funkcji, która zmienia tylko typ ZWRACANYCH danych – taka funkcja już istnieje

```
System.out.println(add(5.0F));
```

```
public static float add(float a) {  
    return (float)
```

```
}
```

'add(float)' is already defined in 'com.company.Main'

**Co mam wywołać ?
Bo nie wiem**

```
public static int add(float a) {  
    return (int)--a;  
}
```

OOP

Programowanie obiektowe

OOP - object-oriented programming

Obiekt = stan + zachowanie

- Stan = zmienne = pola / właściwości
- Zachowanie = funkcje = metody

OOP

```
class Film {  
    String nazwa;  
    int rok;  
    void informacje(){...}  
}
```

Obiekt - Film

```
Film avatar = new Film();  
avatar.nazwa = „Avatar”;  
avatar.rok = 2009;
```

```
Film titanic = new Film();  
titanic.nazwa = „Titanic”  
titanic.rok = 1997;  
titanic.informacje();
```

OOP – zadanie poczytać (20 min)

- Abstrakcja - (interfejsy)
- Hermetyzacja - (enkapsulacja)
- Polimorfizm
- Dziedziczenie

Odpowiedzi wysłać na szguzik@wsb.gda.pl

OOP - implementacja

```
package com.company;

public class Movie {
    String title = "Default";
    int year;
    public void info() {
        System.out.println("Nazwa: " + title + ", rok:");
    }
}
```

OOP - implementacja

```
public class Main {  
  
    public static void main(String[] args) {  
        Movie avatar = new Movie();  
        avatar.title = "Avatar";  
        avatar.year = 2009;  
  
        Movie titanic = new Movie();  
        titanic.title = "Titanic";  
        titanic.year = 1997;  
        titanic.info();  
    }  
}
```

Nazwa: Titanic, rok: 1997

Nazwa: Avatar, rok: 2009

OOP - implementacja

```
Movie movie1 = avatar;  
movie1.info();  
movie1.title = "Random";  
avatar.info();
```

Nazwa: Titanic, rok: 1997

Nazwa: Avatar, rok: 2009

Nazwa: Avatar, rok: 2009

Nazwa: Random, rok: 2009

Wszystkie obiekty na bazie klasy zachowują się jak typ złożony – zmienna typu referencyjnego

OOP - konstruktory

**Specjalna metoda wywoływana
w momencie instancjonowania obiektu**

OOP - konstruktory

```
public class Main {  
  
    public static void main(String[] args) {  
        Person p1 = new Person();  
    }  
}
```

konstruktor domyślny

OOP - konstruktory

```
package com.company;
```

```
public class Person {  
    String name;  
    int age;  
}
```

konstruktor domyślny

OOP - konstruktory

```
package com.company;  
  
public class Person {  
    Person(){  
        System.out.println("Konstruktor klasy");  
    }  
    String name;  
    int age;  
}
```

Przeciążenie konstruktora domyślnego

OOP - konstruktory

Konstruktor bezparametryowy

```
▶ public class Main {  
▶     ▶     public static void main(String[] args) {  
▶         Person p1 = new Person();  
▶     }  
▶ }
```

Konstruktor klasy

OOP - konstruktory

Konstruktor można przeciążać

OOP - konstruktory

```
package com.company;

public class Person {
    Person() {
        System.out.println("Konstruktor klasy");
    }
    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    String name;
    int age;
}
```



The code illustrates constructor overloading in Java. It defines two constructors for the `Person` class: one that takes no arguments and prints "Konstruktor klasy" to the console, and another that takes `String name` and `int age`. Inside the second constructor, two assignments are made: `this.name = name;` and `this.age = age;`. The annotations consist of a yellow bracket above the second constructor's body, a red bracket on the left side of the second constructor's definition, and two arrows pointing from the assignment statements to the corresponding parameters: a white arrow for `this.name` and a blue arrow for `this.age`.

OOP - konstruktory

```
package com.company;

public class Person {
    Person() {
        System.out.println("Konstruktor klasy");
    }
    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    String name;
    int age;
}
```

Adrian, 18

OOP – Static – non Static

```
package com.company;

public class Person {
    Person(String name, int age) {
        this.name = name;
        this.age = age;
        population++;
    }

    String name;
    int age;

    static int population = 0;
}

|
```

OOP – Static – non Static

```
package com.company;

public class Main {

    public static void main(String[] args) {
        Person p1 = new Person(name: "Adrian", age: 18);
        System.out.println(p1.popultion);
        Person p2 = new Person(name: "Bartek", age: 22);

        System.out.println(p2.popultion);
        System.out.println(Person.popultion);
    }
}
```

OOP – Static – non Static

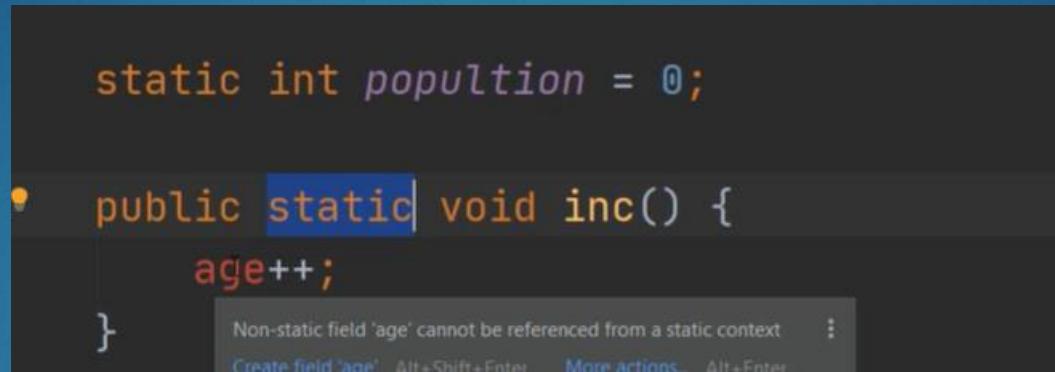
Instancja zmiennej

```
int age;
```

```
static int population = 0;
```

Jedna kopia zmiennej

OOP – Static – non Static



```
static int population = 0;

public static void inc() {
    age++;
}
```

A screenshot of an IDE showing a Java code snippet. The code defines a static integer field `population` and a static method `inc()`. Inside the `inc()` method, there is a line `age++;` where `age` is a non-static field. A tooltip appears over the `age` reference with the text: "Non-static field 'age' cannot be referenced from a static context". Below the tooltip are two options: "Create field 'age'" and "More actions".

Metody statyczne nie mogą mutować pól (elementów) niestatycznych

OOP – Static – non Static

```
String name;  
int age;  
  
static int popultion = 0;  
  
public void inc() {  
    popultion++;  
}
```

OOP – Static – non Static

```
public class Main {  
  
    public static void main(String[] args) {  
        Person p1 = new Person(name: "Adrian", age: 18);  
        System.out.println(p1.popultion);  
        Person p2 = new Person(name: "Bartek", age: 22);  
  
        System.out.println(p2.popultion);  
        Person.inc();  
        System.out.println(Person.popultion);  
    }  
}
```

1
2
3

Hermetyzacja

```
package com.company;

public class Person {
    String name;
    int age;
}
```

```
package com.company;

public class Person {
    private String name;
    int age;
}
```

Hermetyzacja

```
public String getName() {  
    return name;  
}  
  
public void setName(String name) {  
    this.name = name;  
}
```

Hermetyzacja

```
public void setName(String name) {  
    if(name.length() >= 3) {  
        name = name.substring(0, 1).toUpperCase() +  
name.substring(1).toLowerCase();  
        this.name = name;  
    }  
}
```

Warunek if oraz w przypadku spełnienia:

- Pobranie pierwszej litery i zamiana na „dużą” literę
- Pobranie kolejnych (przeciążenie) i zamiana na „małe” litery
- Przypisnie zmiennej name do zmiennej this.name (zmiennej prywatnej do zmiennej klasowej)

Hermetyzacja

```
public class Main {  
    public static void main(String[] args) {  
        Person person = new Person();  
        person.setName("aDrIaN");  
        System.out.println(person.getName());  
    }  
}
```

Adrian

Dziedziczenie

```
public class Animal {  
    String name;  
    int age;  
  
    public void eat() {  
        System.out.println("Tasty!");  
    }  
}
```

```
package com.company;  
  
public class Main {  
  
    public static void main(String[] args) {  
        Animal animal = new Animal();  
        Cat cat = new Cat();  
  
        cat.getVoice();  
        cat.eat();  
    }  
}
```

Zachodzi zawsze minimum na dwóch klasach

Dziedziczenie

```
public class Animal {  
    String name;  
    int age;  
  
    public void eat() {  
        System.out.println("Tasty!");  
    }  
}
```

Superclass

```
package com.company;  
  
public class Main {  
  
    public static void main(String[] args) {  
        Animal animal = new Animal();  
        Cat cat = new Cat();  
  
        cat.getVoice();  
        cat.eat();  
    }  
}
```

Subclass

Dziedziczenie

```
public class Animal {  
    String name;  
    int age;  
  
    public void eat() {  
        System.out.println("Tasty!");  
    }  
}
```

Superclass

```
package com.company;  
  
public class Main {  
  
    public static void main(String[] args) {  
        Animal animal = new Animal();  
        Cat cat = new Cat();  
  
        cat.getVoice();  
        cat.eat();  
    }  
}
```

Subclass

Dziedziczenie

```
public class Animal {  
    String name;  
    int age;  
  
    public void eat() {  
        System.out.println("Tasty!");  
    }  
}
```

```
public class Cat extends Animal {  
    String color;  
  
    public void getVoice() {  
        System.out.println("Meow");  
    }  
}
```

Superclass

Subclass

Dziedziczenie

```
public class Animal {  
    String name;  
    private int age;  
    public void eat() {  
        System.out.println("Tasty!");  
    }  
}
```

```
package com.company;  
  
public class Main {  
  
    public static void main(String[] args) {  
        Animal animal = new Animal();  
        Cat cat = new Cat();  
  
        cat.getVoice();  
        cat.eat();  
    }  
}
```

Superclass

Subclass

Nie jest widoczne
w Subclass

Dziedziczenie

```
public class Animal {  
    String name;  
    private int age;  
    public void eat() {  
        System.out.println("Tasty!");  
    }  
}
```

```
public class Cat extends Animal {  
    String color;  
  
    public void getVoice() {  
        System.out.println("Meow");  
    }  
}
```

Superclass

Subclass

Nie jest widoczne
w Subclass

Dziedziczenie

Jedna klasa może dziedziczyć po
innej ale też tylko

JEDNEJ klasie

Dziedziczenie

```
public class Tiger extends Cat {  
}
```

Klasa Tiger dziedziczy metody z klasy Cat

Klasa Cat dziedziczy metody z klasy Animal

Klasa Tiger ma wszystkie metody z obu klas

Dziedziczenie

```
public class Animal { ←  
    String name;  
    private int age;  
  
    public void eat() {  
        System.out.println("Tasty!");  
    }  
}
```

```
public class Cat extends Animal { ←  
    String color;  
  
    public void getVoice() {  
        System.out.println("Meow");  
    }  
}
```

```
public class Tiger extends Cat {  
}
```

Dziedziczenie

```
public void setName(String name) {  
    if(name.length() >= 3) {  
        name = name.substring(0, 1).toUpperCase() +  
name.substring(1).toLowerCase();  
        this.name = name;  
    } -----  
}
```

this dziedziczy po obiekcie w którym się znajduje

```
public class Cat extends Animal {  
    String color;  
  
    public void getVoice() {  
        System.out.println("Meow");  
        super.-----  
    }  
}
```

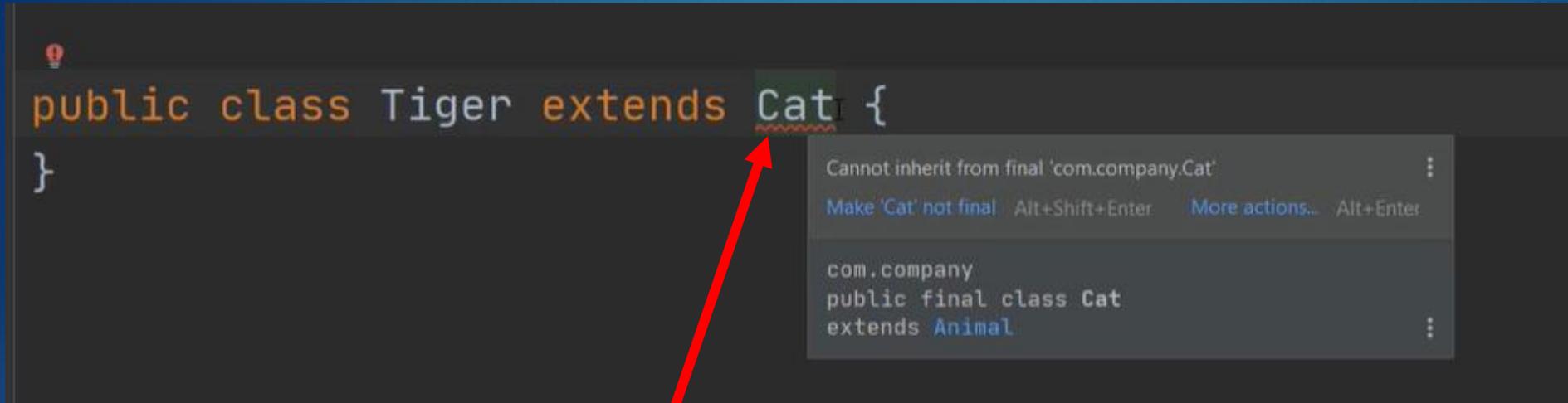
super dziedziczy po obiekcie superclass

Dziedziczenie

```
package com.company;  
.  
final public class Cat extends Animal {  
    String color;  
  
    public void getVoice() {  
        System.out.println("Meow");  
    }  
}
```

Po tej klasie dziedziczyć nie można ponieważ jest „oznaczone jako” final, które zabrania między innymi dziedziczenie po klasie

Dziedziczenie



The screenshot shows a Java code editor with the following code:

```
public class Tiger extends Cat { }
```

A tooltip is displayed over the word `Cat` in the `extends` clause, indicating that it cannot be inherited from a final class. The tooltip text is:

Cannot inherit from final 'com.company.Cat' ::
Make 'Cat' not final Alt+Shift+Enter More actions... Alt+Enter

Below the tooltip, the definition of the `Cat` class is shown:

com.company
public final class Cat
extends Animal ::

😊 Nie można 😊

Dziedziczenie - konstruktory

```
↓ public class Animal {  
    public Animal() {  
        System.out.println("Konstruktor klasy Animal");  
    }  
  
    String name;  
    int age;  
}
```

Konstruktor

Dziedziczenie - konstruktory

```
public class Cat extends Animal {  
    public Cat() {  
        System.out.println("Konstruktor klasy Cat");  
    }  
}
```

Konstruktor

Dziedziczenie - konstruktory

**Konstruktory są powoływanie wg. Hierarchii
Od najstarszego (superclass) do
najmłodszego (subclass)**



```
Konstruktor klasy Animal  
Konstruktor klasy Cat
```

Dziedziczenie - konstruktory

```
package com.company;

public class Animal {
    public Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }

    String name;
    int age;
}
```

```
package com.company;

public class Cat extends Animal {
    public Cat(String name, int age, String color) {
        super(name, age);
    }

    String color;
}
```

Super() – wywołanie konstruktora klasy „wstecz” (superclass)

Dziedziczenie - konstruktory

```
public class Cat extends Animal {  
    public Cat(String name, int age, String color) {  
        this.color = color;  
        super(name, age);  
    }  
}
```



**Konstruktor bazowy (superclass) musi
zostać wywołany jako pierwszy**

Dziedziczenie - konstruktory

```
public class Cat extends Animal {  
    public Cat(String name, int age, String color) {  
        super(name, age);  
        this.color = color;  
    }  
}
```



**Konstruktor bazowy (superclass) musi
zostać wywołany jako pierwszy**

Dziedziczenie – konstruktory - zastosowanie

```
public class Main {  
  
    public static void main(String[] args) {  
        Cat cat = new Cat(name: "Sisi", age: 4, color: "czarny");  
        System.out.println(cat.name);  
        System.out.println(cat.age);  
        System.out.println(cat.color);  
    }  
}
```

Sisi
4
czarny

Dziedziczenie – konstruktory - zastosowanie

Dziękuję za uwagę!