Języki i metody programowania – Java
INF302W
Wykład 1 (część 1)
na podstawie
https://docs.oracle.com/javase/tutorial/

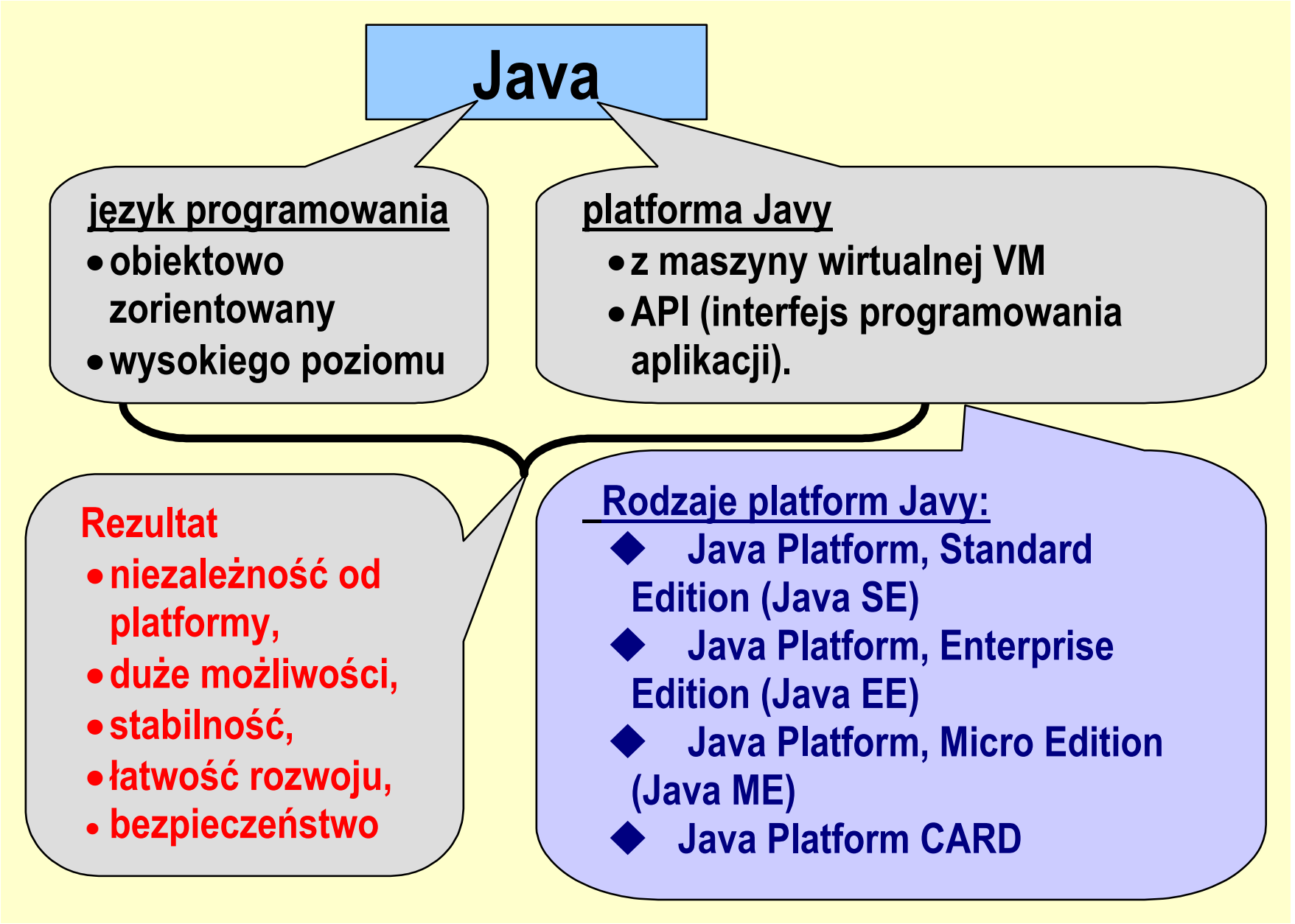
Autor

Dr inż. Zofia Kruczkiewicz

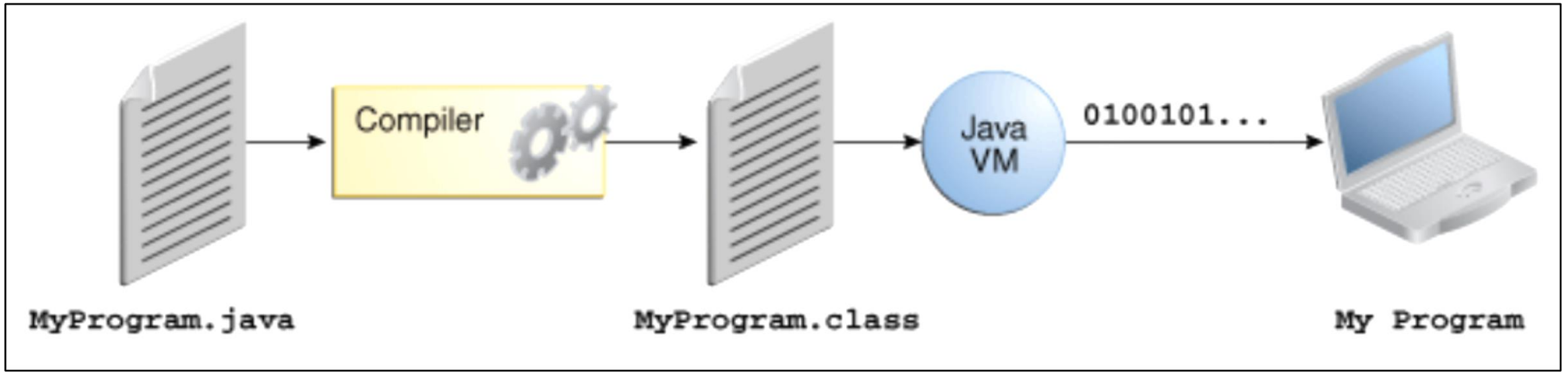
Struktura wykładu

1. **Porównanie języków Java i C++, proste programy typu aplikacja- budowa klasy, składowe statyczne i niestatyczne, rola metody main; pisanie programu z użyciem jedynie metody main - podstawowe typy danych, operatory arytmetyczne, klasa System.**
2. **Identyfikacja danych reprezentowanych przez klasy podczas opracowania koncepcji prostego programu obiektowego. Tworzenie programów z użyciem jednej i wielu klas: budowa klasy, konstruktory, metody, zastosowanie składowych statycznych i niestatycznych, operator new, odwołanie do obiektów-operator kropka, wywołanie metod, przeciążenie metod (1- część).**

1. Charakterystyka technologii Java



1 (cd). Język Java – proces tworzenia programu

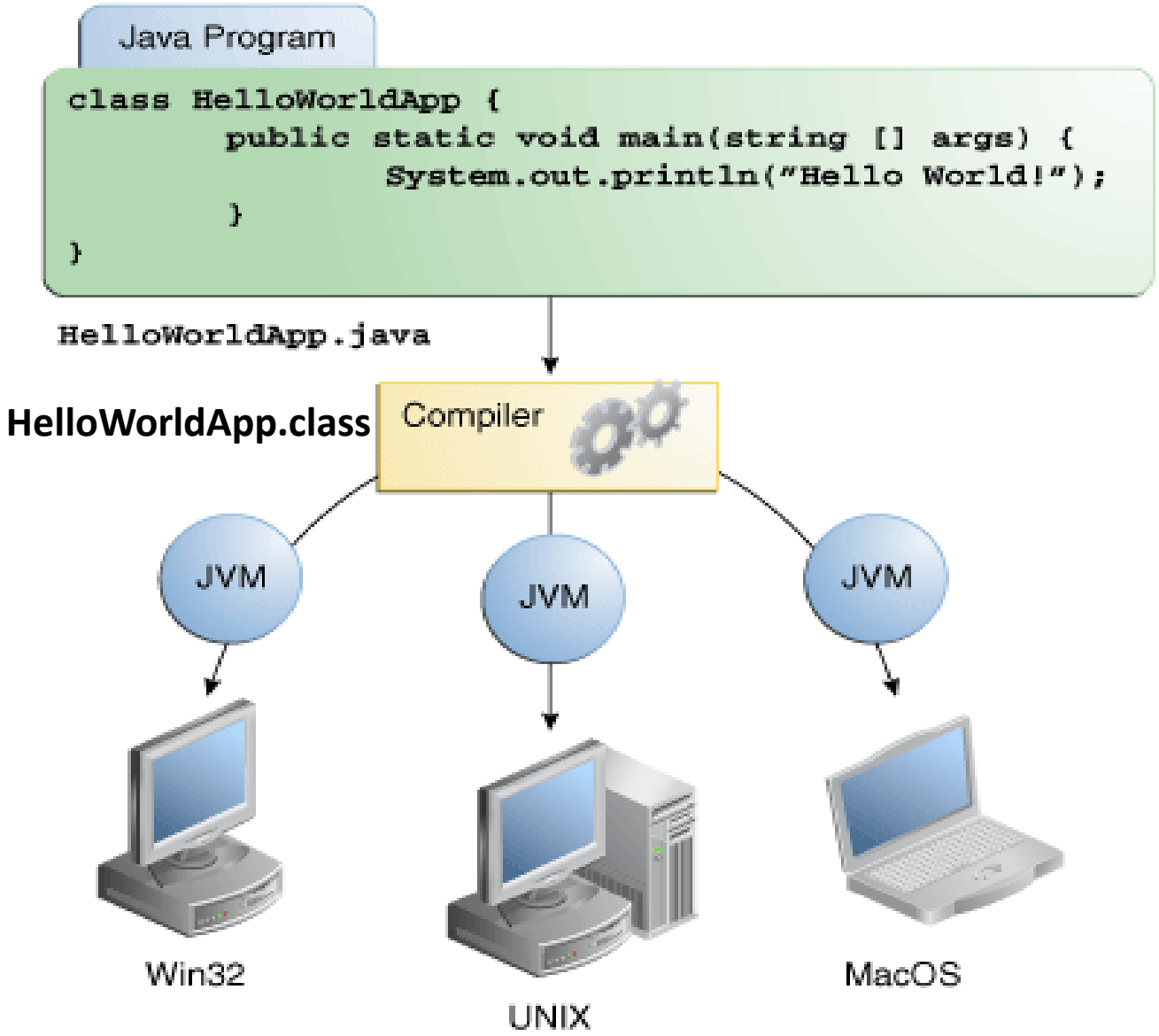


który jest:

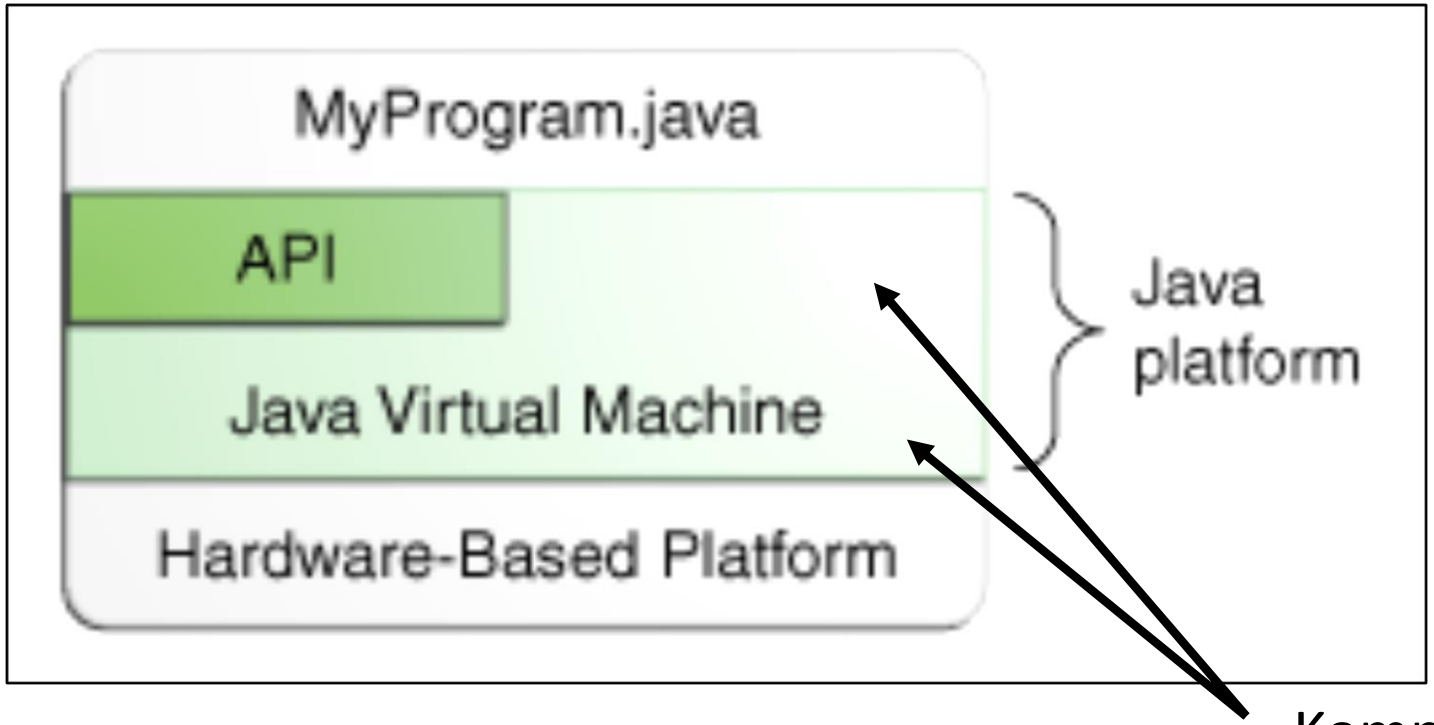
prosty
zorientowany obiektowo
rozproszony
wielowątkowy
dynamiczny

o neutralnej architekturze
przenośny
o wysokiej wydajności
odporny na błędy
bezpieczny

1 (cd). JVM (Java Virtual Machine) umożliwia działanie tego samego skompilowanego programu na różnych systemach operacyjnych



1 (cd). Platforma Javy SE



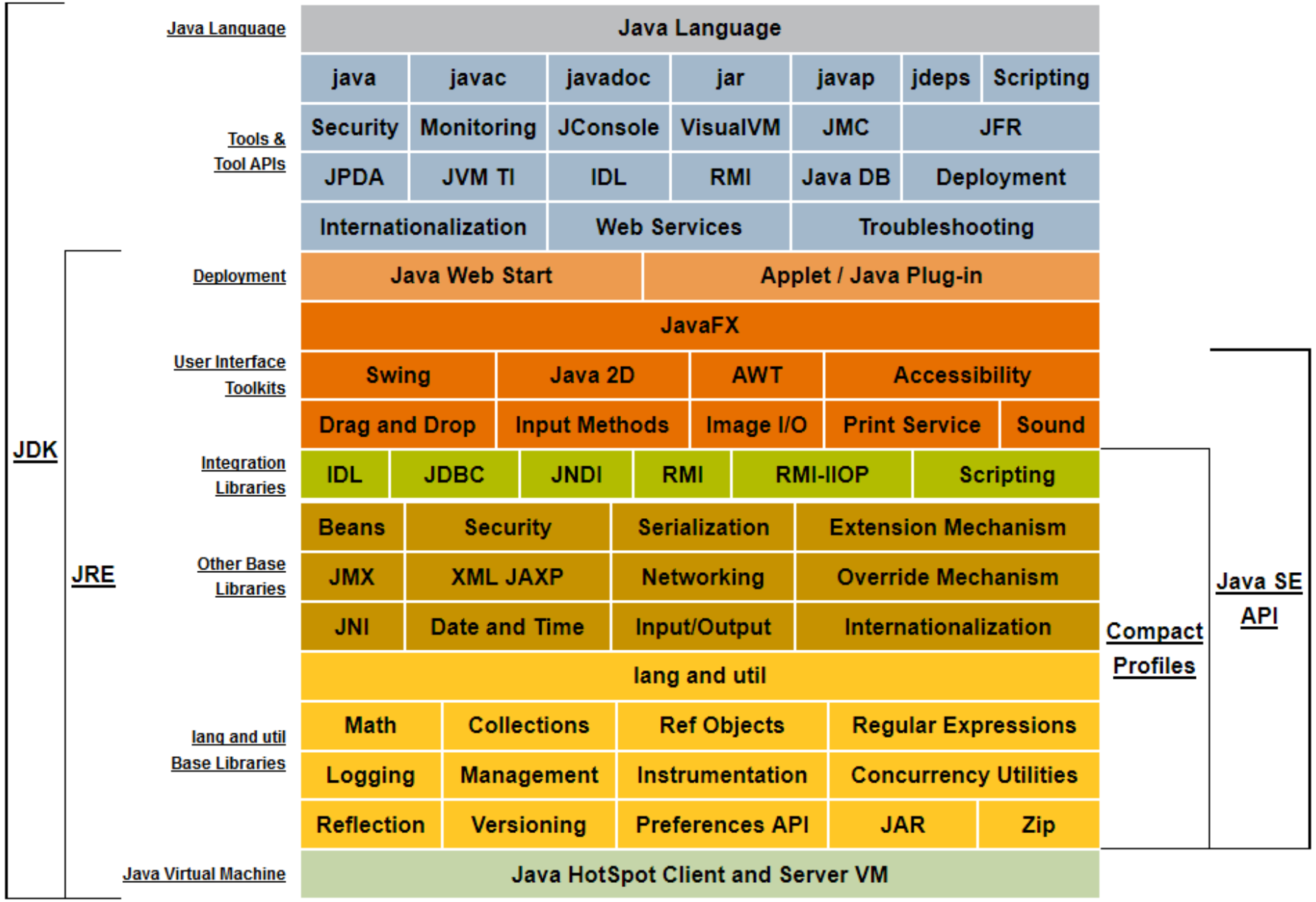
Komponenty platformy Java

An application programming interface (API),

Interfejs programistyczny aplikacji (API) to zbiór pakietów, klas i interfejsów jako interfejs oprogramowania ułatwiający interakcję JVM z kodem uruchamianych programów i ograniczają rozmiar tych programów.

Description of Java Conceptual Diagram

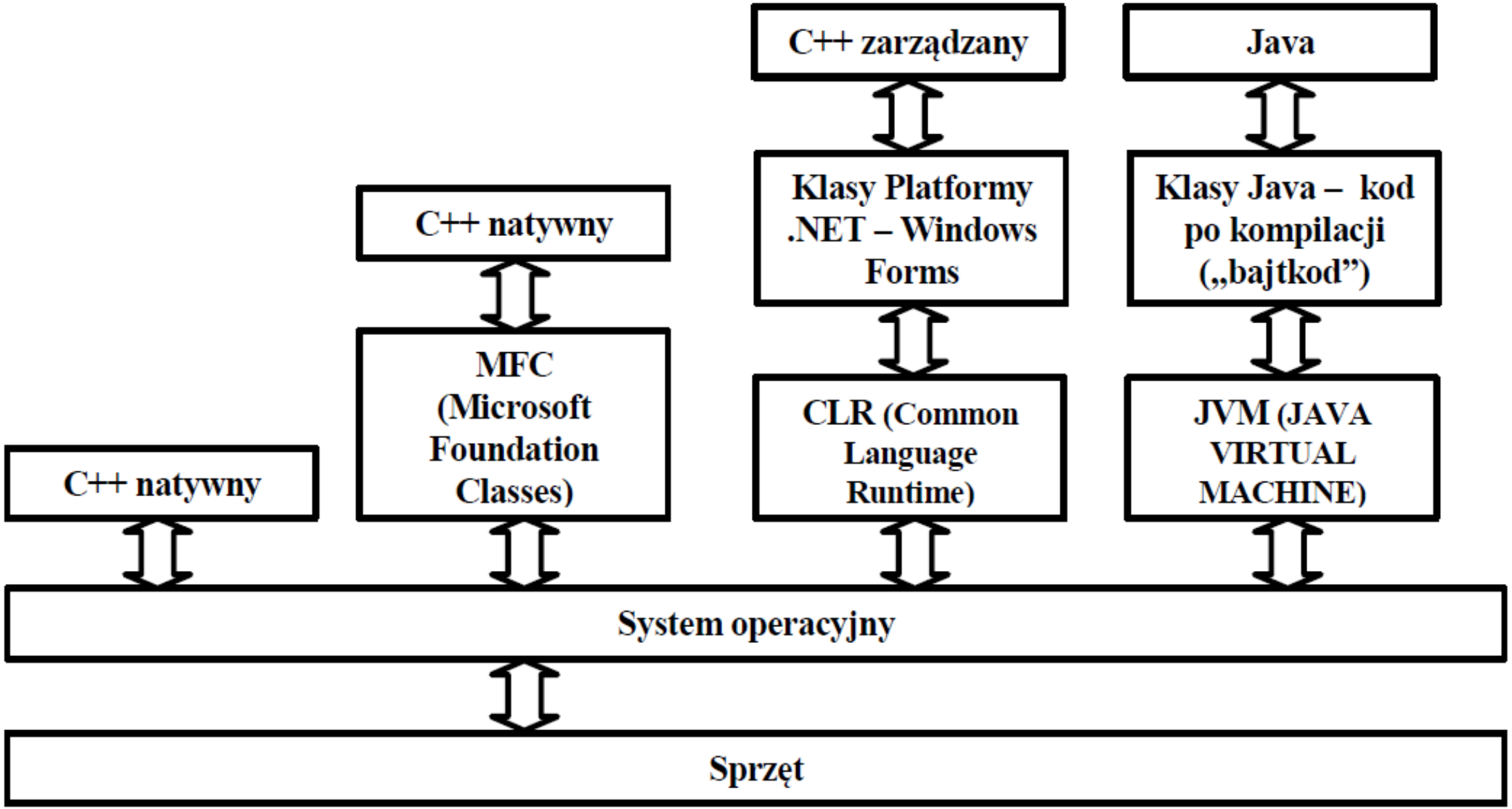
https://docs.oracle.com/javase/8/docs/



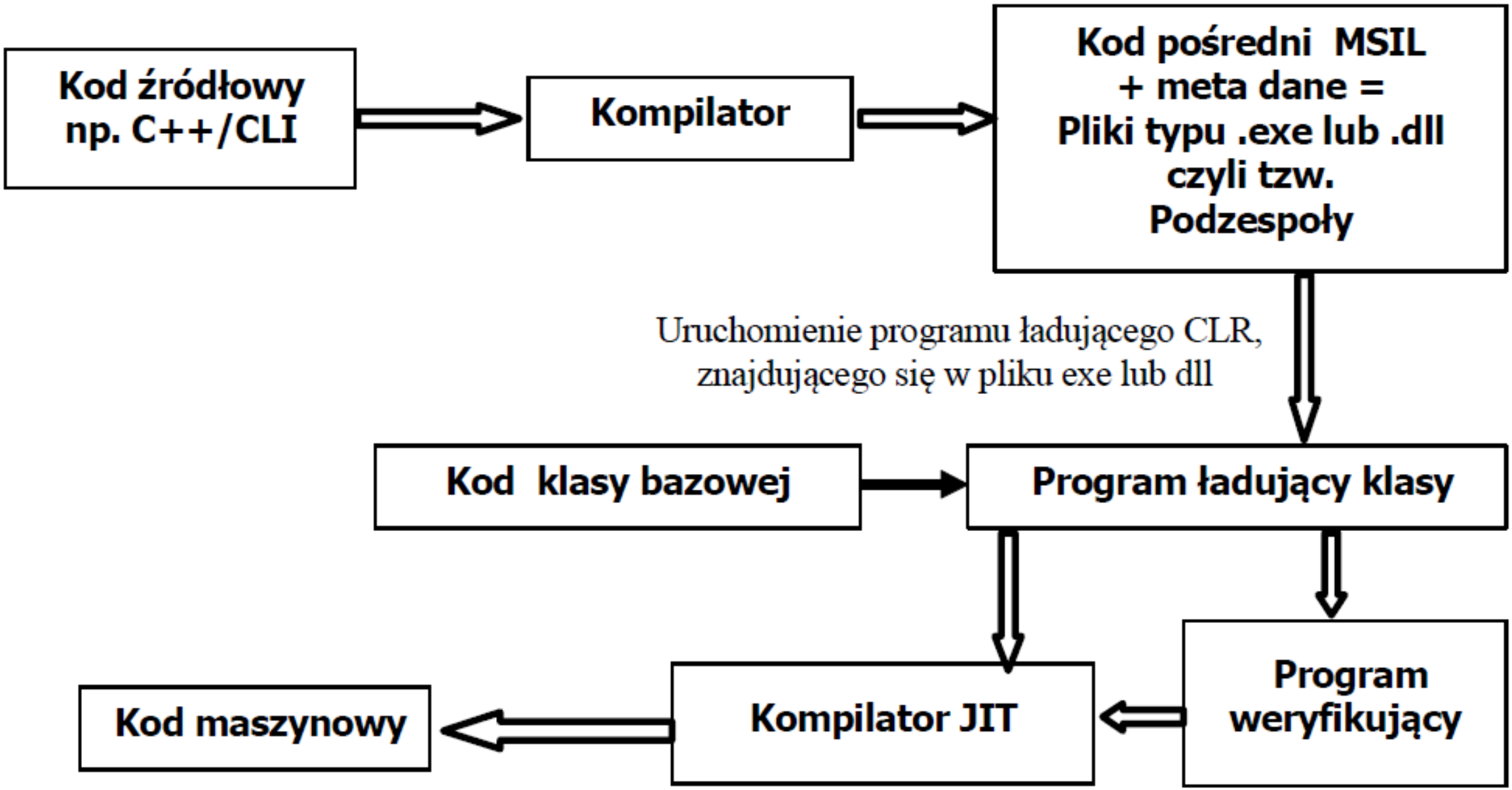
1 (cd). Zalety technologii Java SE

- **Łatwy** dla programistów znających język C lub C++.
- **Mniej kodu**: metryki programów (liczenie klas, liczenie metod itd.) o identycznej funkcjonalności napisanego w Javie i w C++ -mogą być **czterokrotnie mniejsze w Javie** niż w C ++.
- **Pisanie lepszego kodu**: język programowania Java zachęca do stosowania dobrych praktyk kodowania, a automatyczne usuwanie danych z pamięci pomaga uniknąć wycieków pamięci. Jego ukierunkowanie na obiekty, komponenty JavaBeans ™ oraz łatwe do rozbudowy interfejsy API **umożliwiają ponowne użycie istniejącego, przetestowanego kodu i wprowadzenie mniej błędów**.
- **Szybsze opracowywanie programów**: Język programowania Java jest prostszy niż C ++, a czas pisania tego programu może być nawet **dwukrotnie krótszy w Javie** niż w C ++.
- **Należy unikać zależności programu od systemów operacyjnych**: należy unikać stosowania bibliotek napisanych w innych językach.
- **Napisz raz, uruchom w dowolnym miejscu**: Aplikacje napisane w języku programowania Java są kompilowane do kodu niezależnego od systemu operacyjnego, działają one na dowolnej JVM.
- **Łatwe dystrybuowanie oprogramowania**: dzięki oprogramowaniu **Java Web Start** użytkownicy mogą uruchamiać swoje aplikacje jednym kliknięciem myszy, dokonać automatycznej kontroli ich wersji przy starcie programu - i jeśli aktualizacje są dostępne, oprogramowanie Java Web Start automatycznie je zaktualizuje.

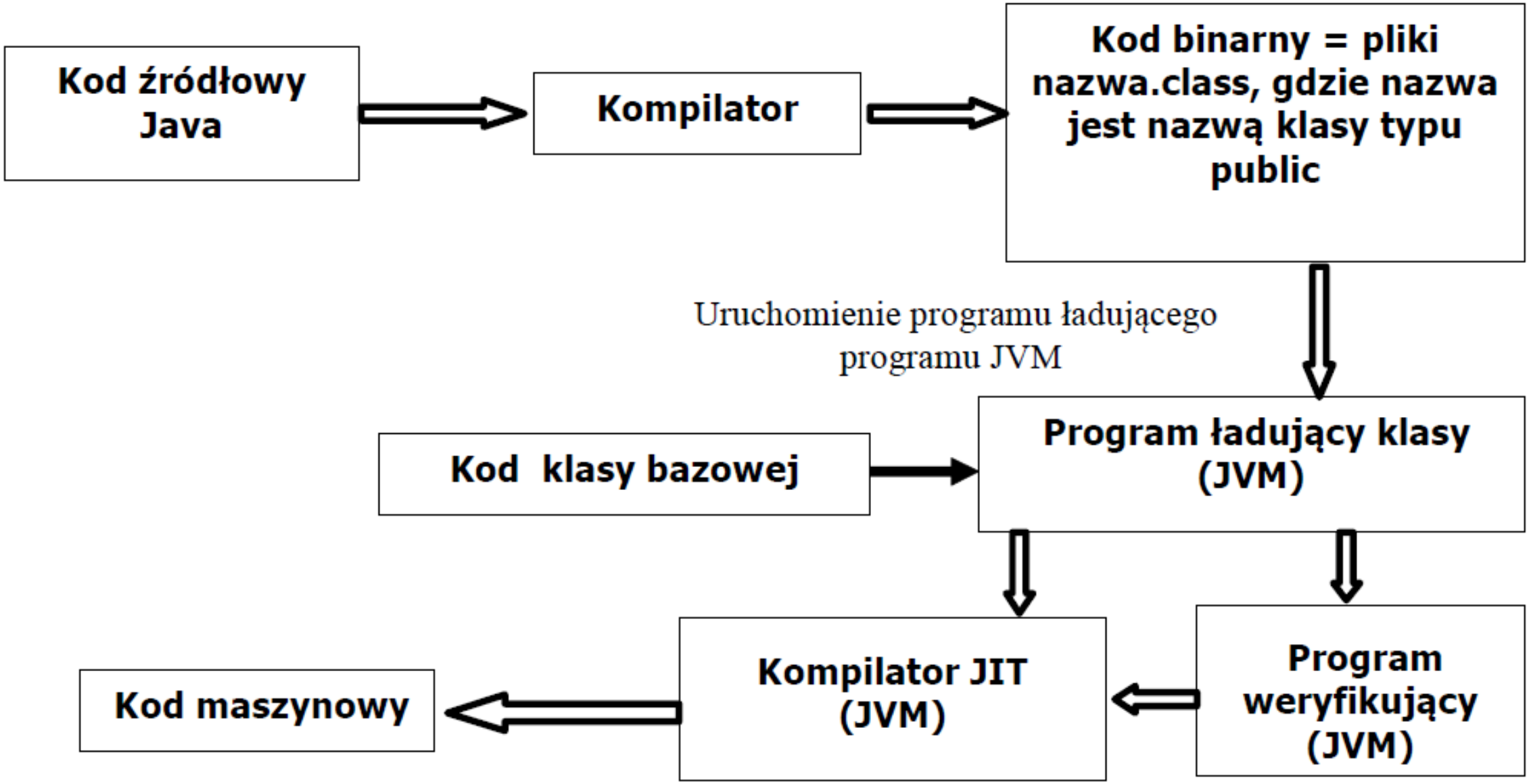
1 (cd). Tworzenie programów w językach C++, C++/CLR oraz Java (Ivor Horton „Od podstaw Visual C++ 2005”)



1 (cd). Przebieg tworzenia i działania programu w języku zarządzanym C++/CLR



1 (cd). Przebieg tworzenia i działania programu w języku Java



1 (cd). Porównanie równoważnych definicji w językach C++, C++/CLR i Java

C++/CLI - Definicje za pomocą typów fundamentalnych (są wtedy mapowane do typów klas wartości w CLR). Preferowane w programach C++/CLI – uniezależniają od konkretnej implementacji CLI .	C++/CLR - Definicje za pomocą typów C++/CLR (klasy wartości)	Java- Definicje typów za pomocą klasy typu **Class**. Obiekty typu **Class** reprezentują klasy i interfejsy działające w programie Javy oraz typy enum, tablice, typy nieobiektowe (**boolean, byte, char, short, int, long, float, double**) oraz słowo kluczowe **void**. Typ **Class** nie posiada konstruktora, natomiast JVM tworzy automatycznie obiekty tej klasy, reprezentujące typy danych ładowanych do programu za pomocą metody *defineClass* programu ładującego klasy.
int ile = 0; **double** arg1 = 0;	System::**Int**32 ile = 0; lub System::**Int**64 ile = 0; System::**Double** arg1 = 0;	**Integer** ile = 0; **Double** arg2 = 0.0;
void oblicz (**int** a) { }	System::**Void** oblicz (System::**Int**32 a) { } lub System::**Void** oblicz (System::**Int**64 a) { }	

1 (cd). Porównanie elementarnych typów danych w C++, C++/CLR i Java

Visual C++ type	.NET Framework type	Java type	Typy obiektowe opakowujące	Java type Class<?>
bool (1 bajt)	System.**Boolean**	**boolean** (1 bajt)	**Boolean**	**Class<Boolean>**
unsigned char (1 bajt)	System.**Byte**			
wchar_t (2 bajty)	System.**Char**	**char** (2 bajty)	**Character**	**Class<Character>**
signed char (1 bajt)	System.**SByte**	**byte** (1 bajt)	**Byte**	**Class<Character>**
short, signed short (2 bajty)	System.**Int16**	**short** (2 bajty)	**Short**	**Class<Short>**
unsigned short (2 bajty)	System.**UInt16**			

1 (cd).Porównanie elementarnych typów danych w C++, C++/CLR i Java

int, signed int, long, signed long (4 bajty)	System.Int32	int (4 bajty)	Integer	Class<Integer>
unsigned int, unsigned long (4 bajty)	System.UInt32	long (8 bajtów)	Long	Class<Long>
__int64, signed __int64 (8 bajtów)	System.Int64			
unsigned __int64 (8 bajtów)	System.UInt64			
double, long double (8 bajtów)	System.Double	double (8 bajtów)	Double	Class<Double>
float (4 bajty)	System.Single	float (4 bajty)	Float	Class<Float>
void	System.Void	void	Void	Class<Void>

2. Tworzenie programu w Javie - przykład

Aplikacja (application)

Program interpretujący aplikacje **java.exe** typu JVM jest uruchamiany w systemie operacyjnym.

Uruchamiana aplikacja (program użytkownika) zawiera **między innymi** jeden moduł źródłowy, którego klasa publiczna zawiera publiczną metodę klasową o nagłówku:

```
public static void main(String args[])
```

Wynik metody nazwa metody lista parametrów

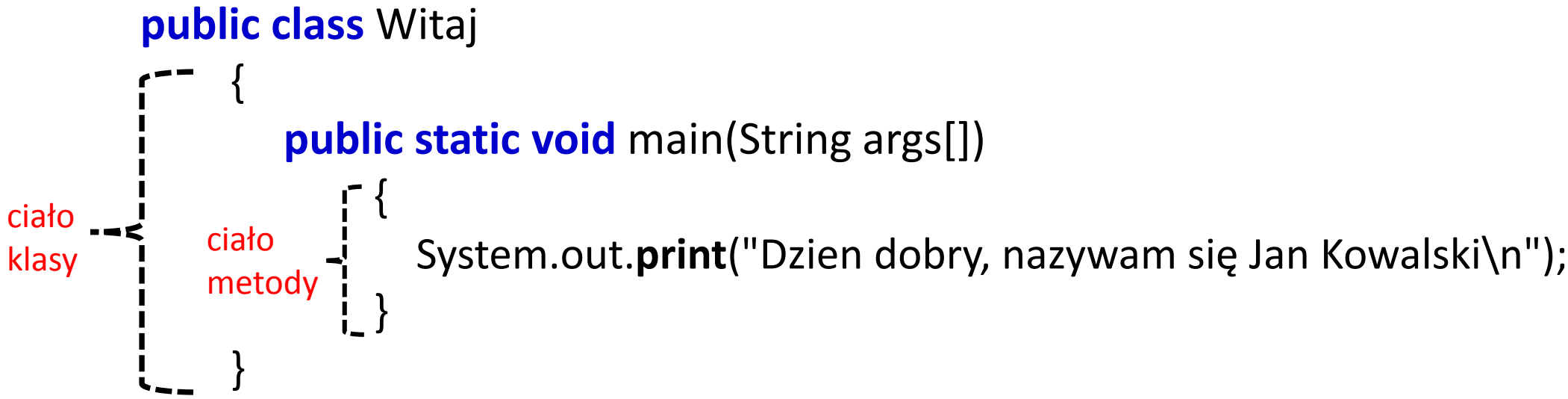
Aplet (applet)

Program interpretujący aplety jest wbudowany np. w przeglądarkę www.

Program typu aplet zawiera między innymi jeden moduł źródłowy, którego klasa publiczna zawiera między innymi podstawowe metody: **init(), start(), stop(), paint(), destroy()**

Uwaga: możliwe jest napisanie programu w Javie, który będzie pracował jako applet i jako aplikacja.

2.1. Tekst źródłowy w Javie - przykład



2.2. Kompilacja

javac Witaj.java

gdzie położenie (katalog) programu javac (kompilator Javy) powinno być znane systemowi operacyjnemu, a katalog bieżący powinien zawierać plik źródłowy Witaj.java. Zostanie wygenerowany plik Witaj.class z instrukcjami dla JVM.

2.3. Interpretacja

java Witaj

interpretator **java** (położenie znane systemowi operacyjnemu)

- wyszuka plik o nazwie ***Witaj.class*** w katalogu bieżącym
- sprawdzi, czy klasa *Witaj* posiada publiczną metodę statyczną ***main***
- wykona instrukcje zawarte w bloku funkcji ***main***, czyli wyświetli na ekranie napis

Dzien dobry, nazywam się Jan Kowalski

i przejdzie do następnego wiersza

Uwagi:

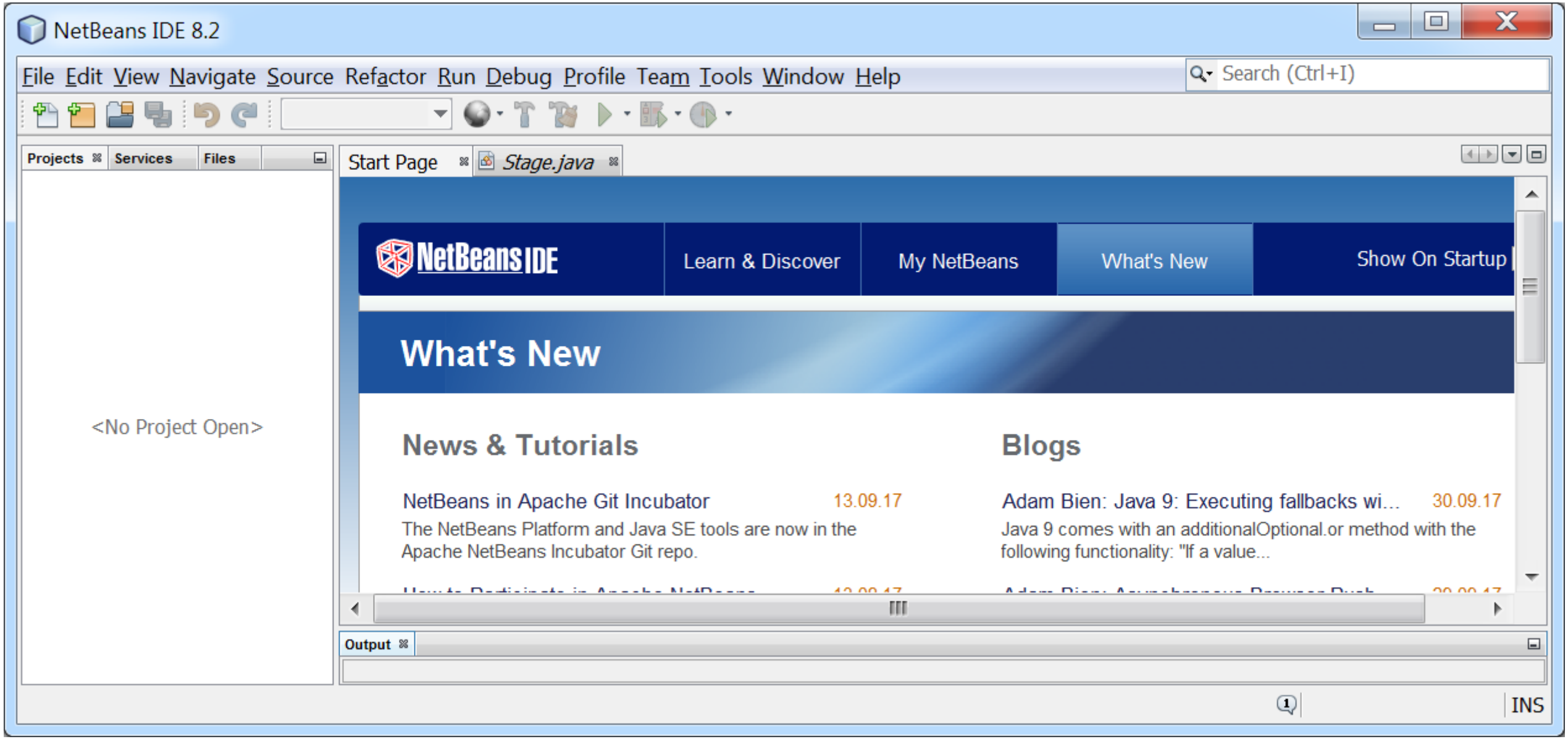
- do metody main z wiersza rozkazowego jako parametr jest przekazywana tablica args obiektów (łańcuchów) klasy ***String*** - w klasie **Witaj** jest ona pomijana
- każda instrukcja kończy się średnikiem
- standardowa klasa ***System***:

a) zawiera statyczny obiekt składowy typu ***PrintStream*** o nazwie ***out***

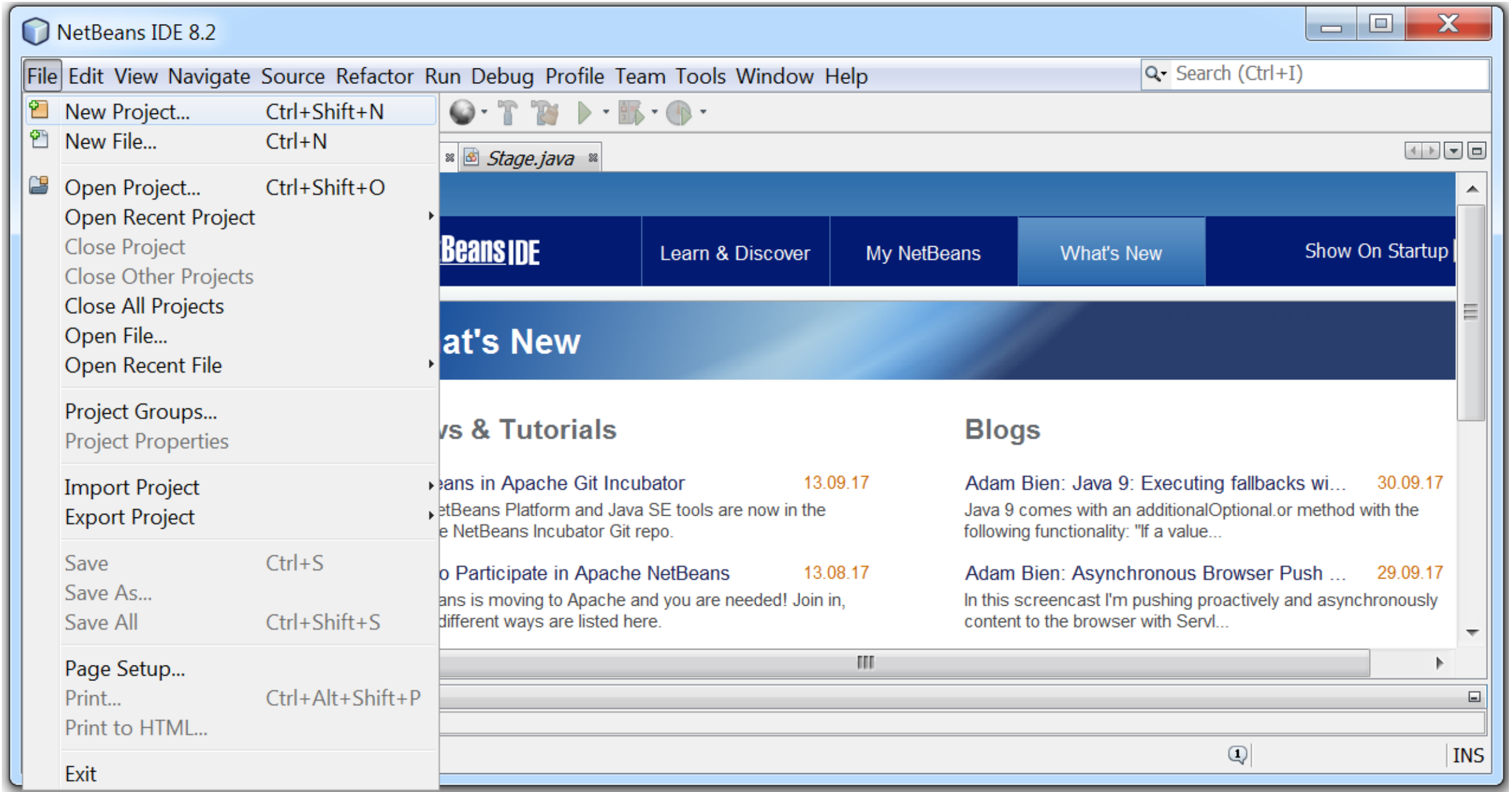
b) wywołanie ***System.out.print*** oznacza pisanie łańcucha typu ***String*** do standardowego strumienia wyjściowego, w tym wypadku ekranu

c) metoda ***print*** generuje jeden wiersz wyjściowy i powraca do metody ***main***

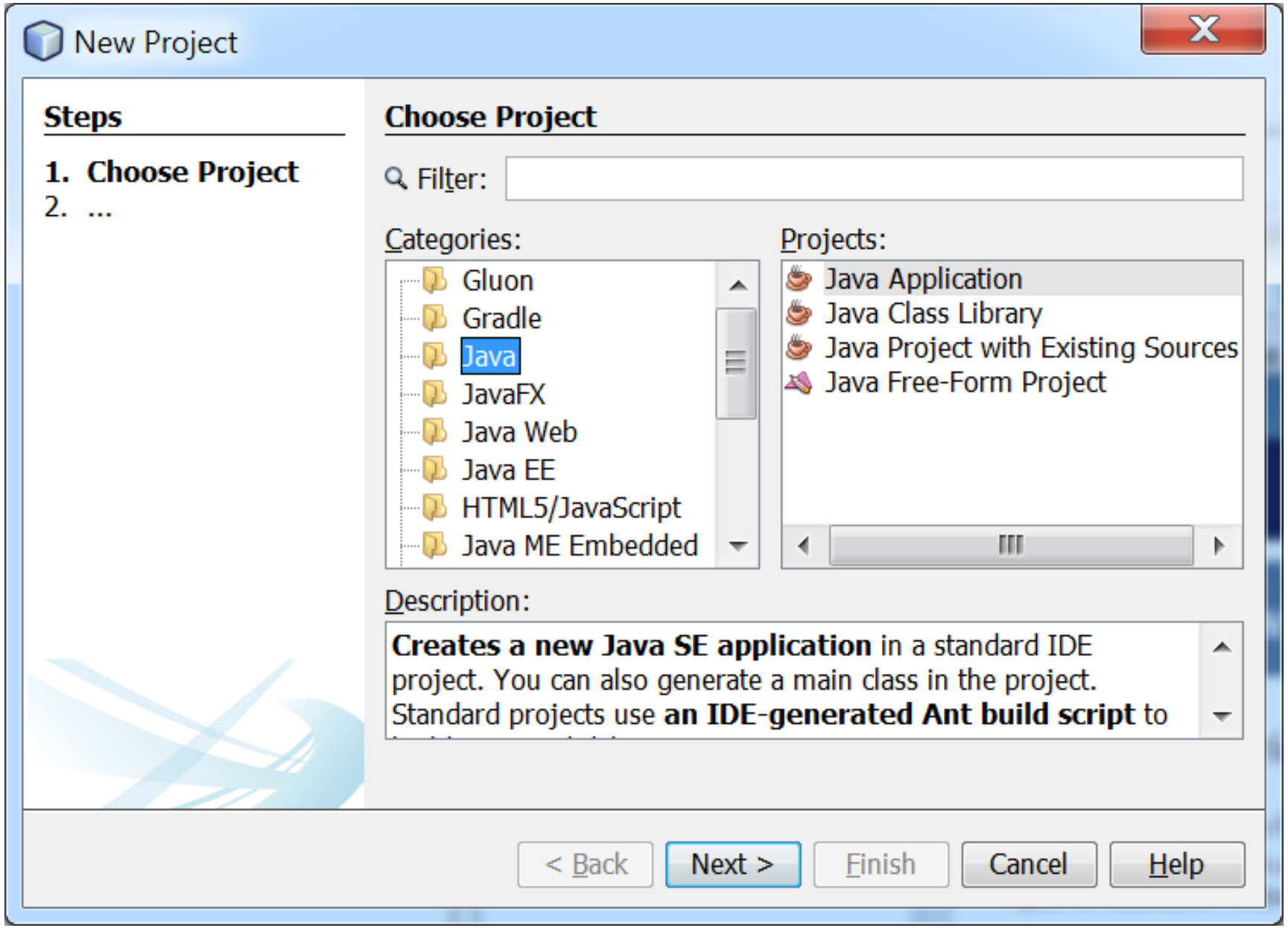
Przykład 1: Tworzenie aplikacji w środowisku NetBeans



Zakładanie nowego projektu

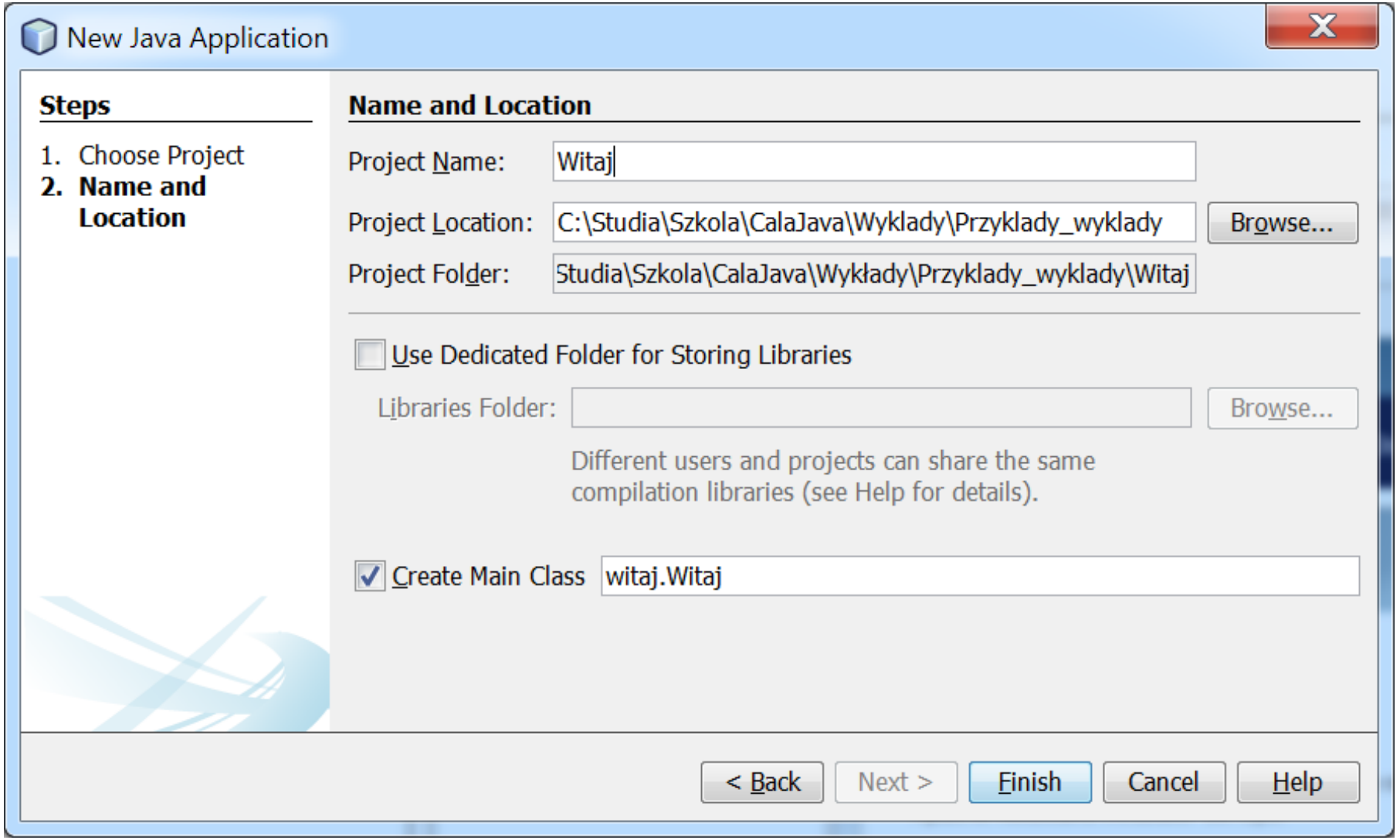


Wybór kategorii projektu Java typu Java Application

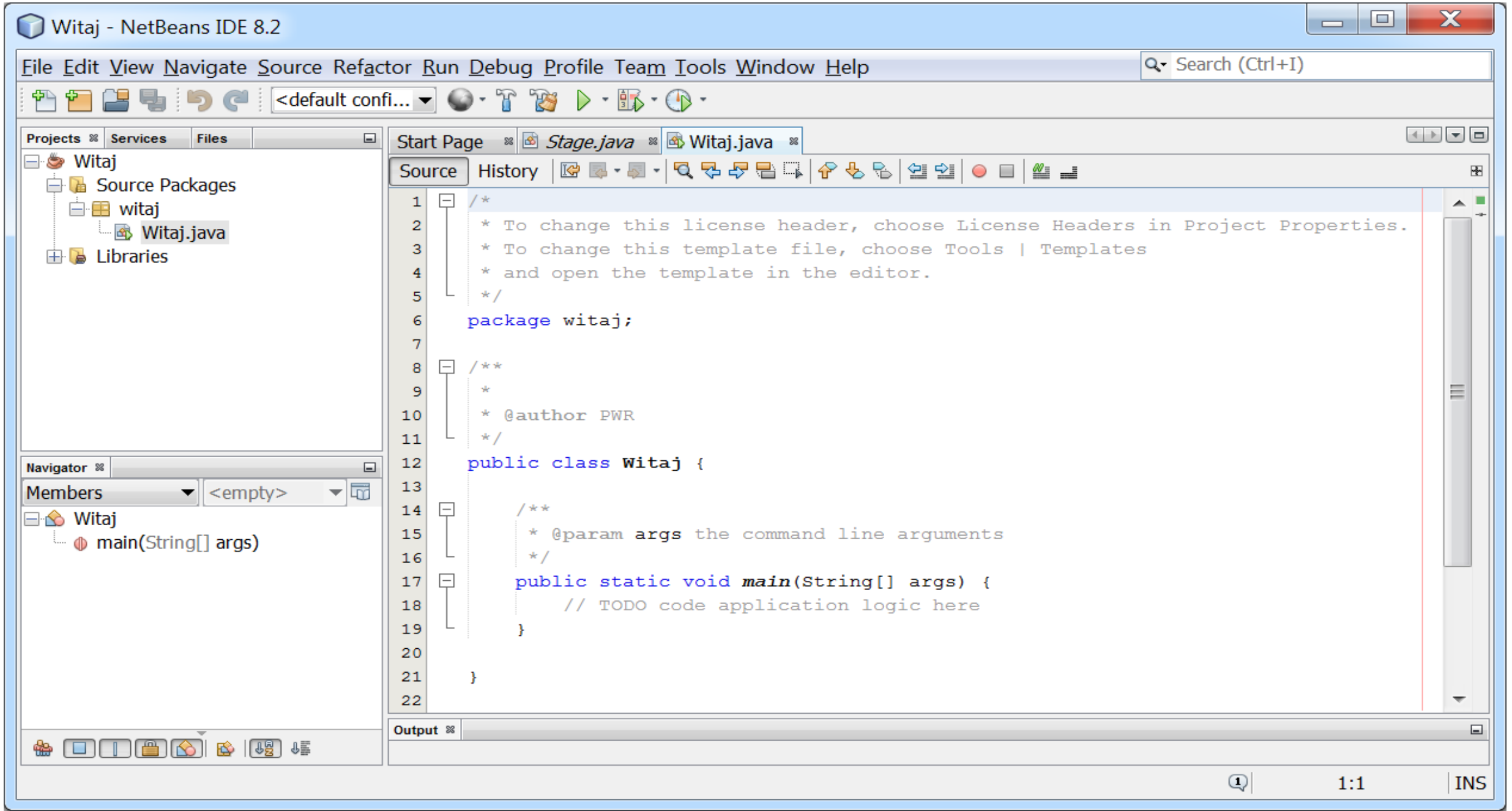


Domyślny formularz projektu po naciśnięcia klawisza Next.

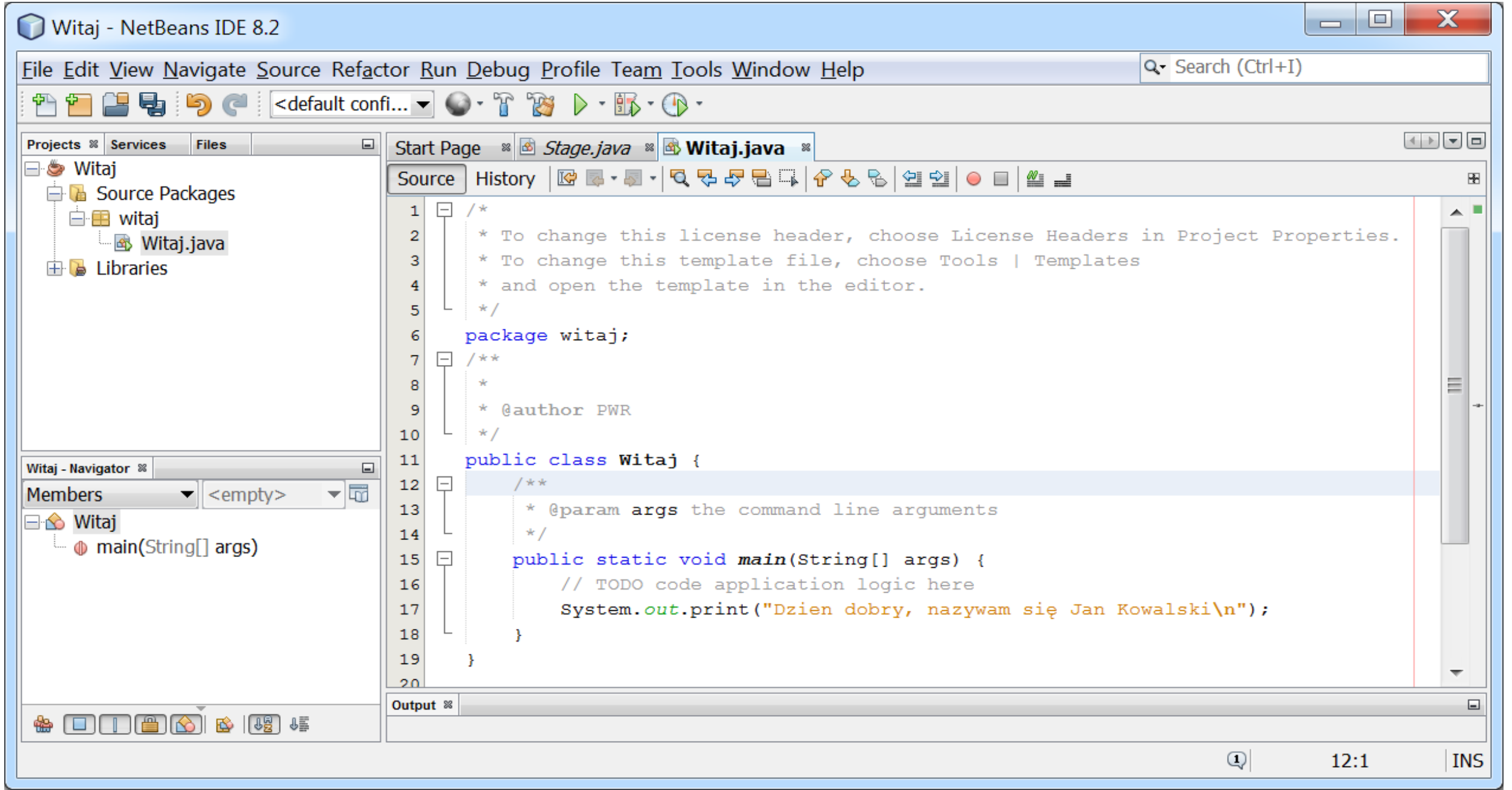
Podanie własnych ustawień projektu: nazwy projektu (Project Name), katalogu (Project Location) oraz nazwy klasy główne zawierającej funkcję main (Create Main Class). Należy zatwierdzić klawiszem Finish te wprowadzone dane.



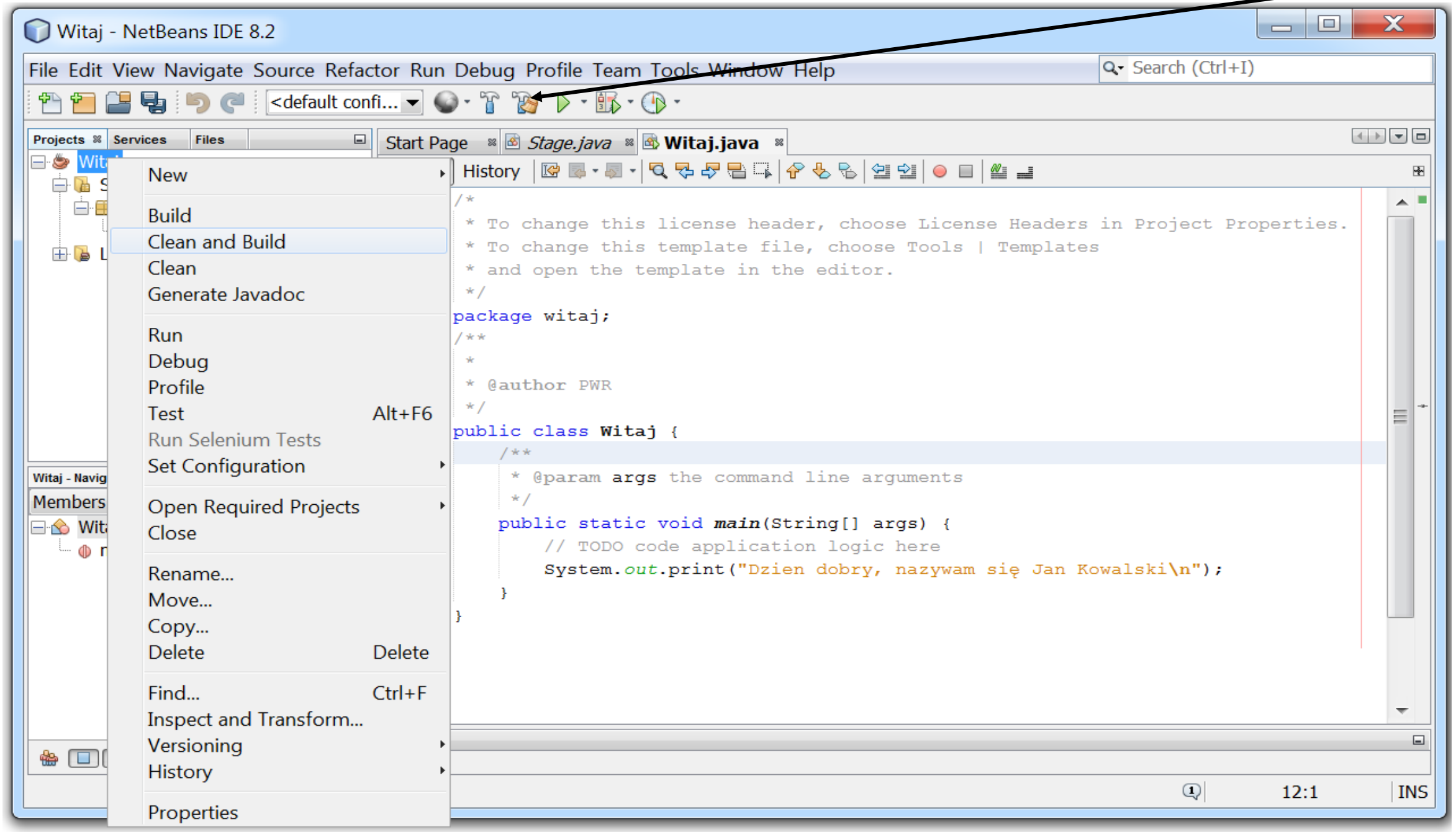
Formularz edycji programu źródłowego



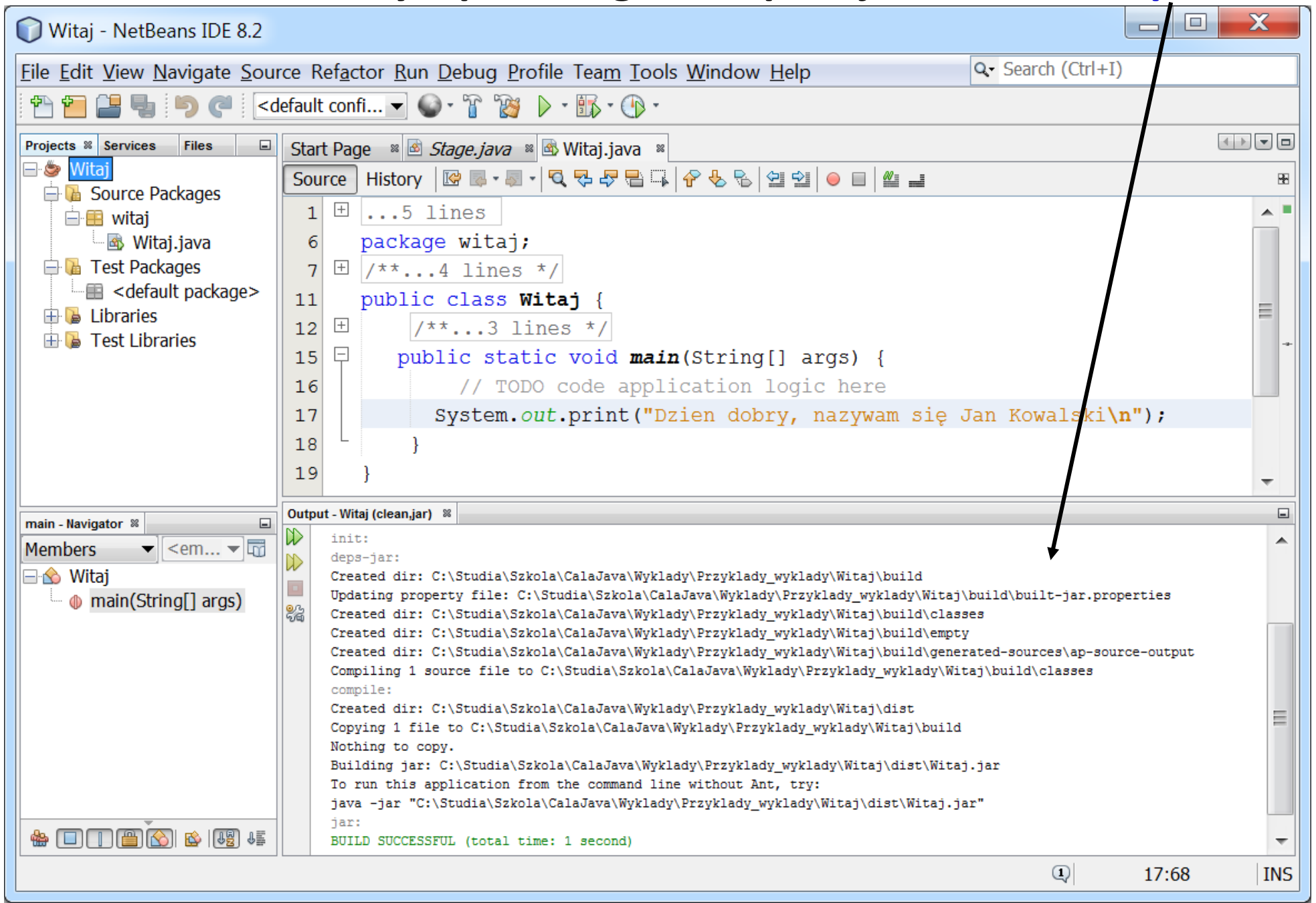
Wpisanie elementarnej instrukcji w funkcji main programu konsolowego



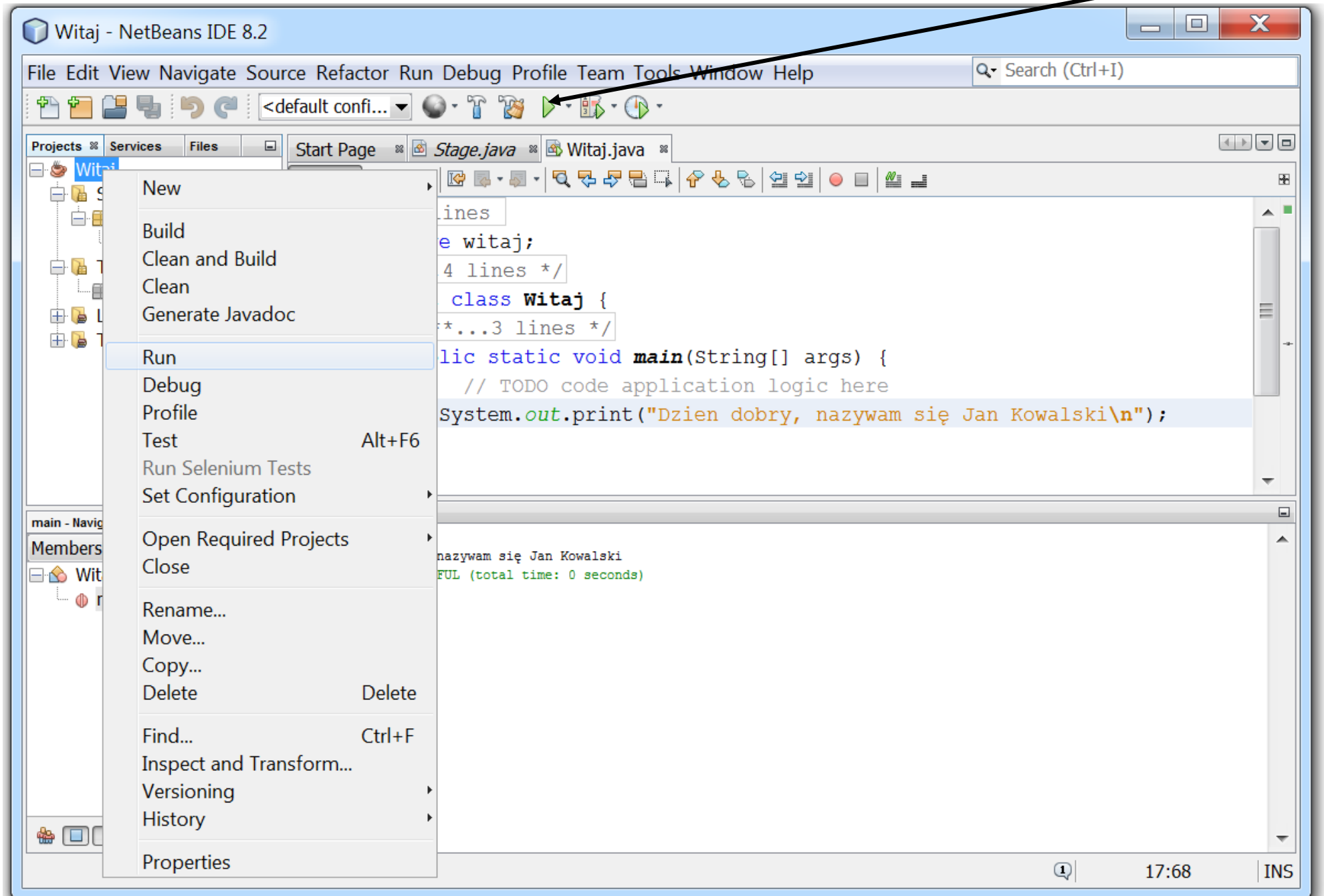
a) Kompilacja programu – wybór pozycji Clean and Build Project lub



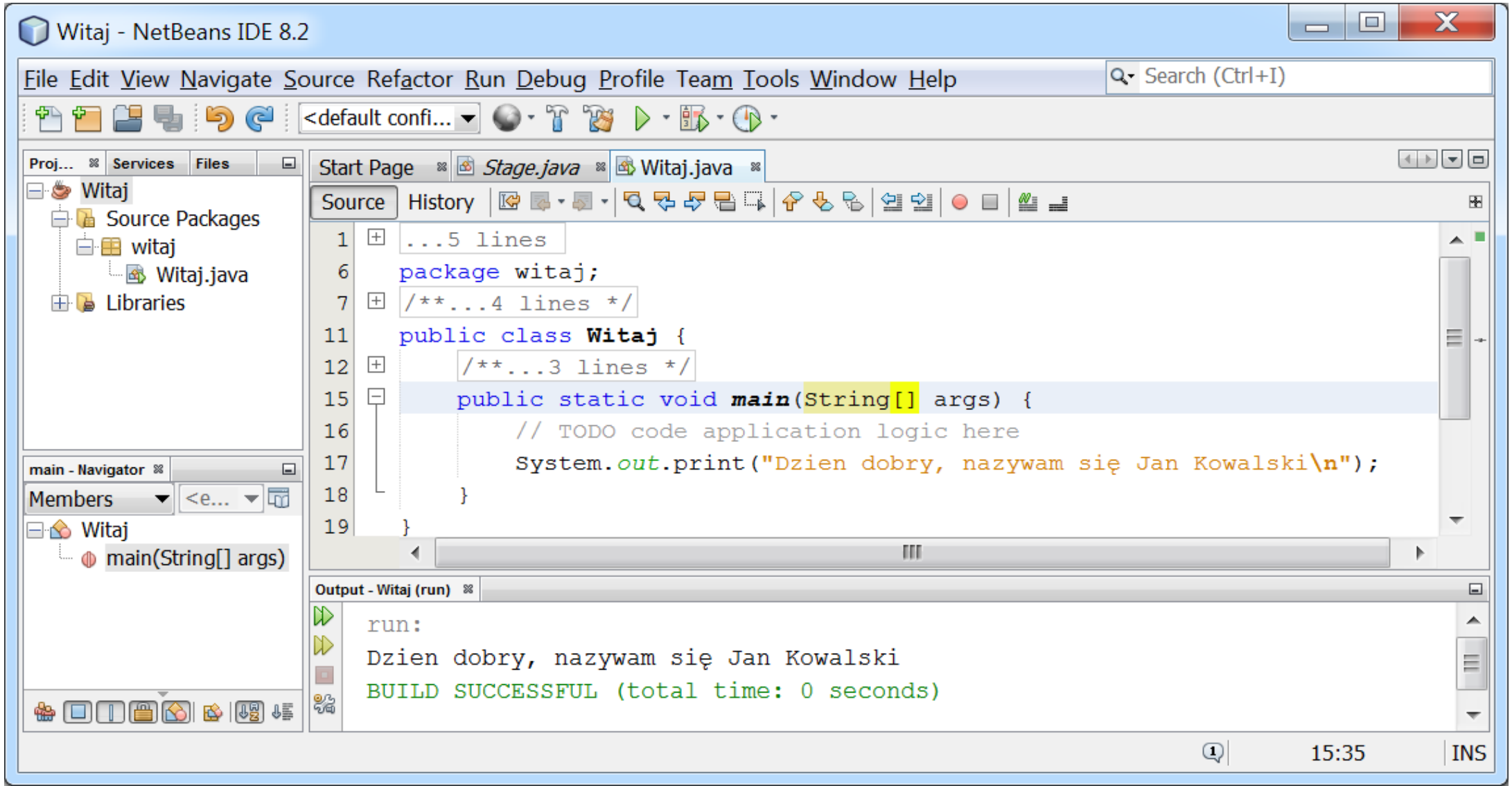
Komunikaty z przebiegu kompilacji w oknie Output



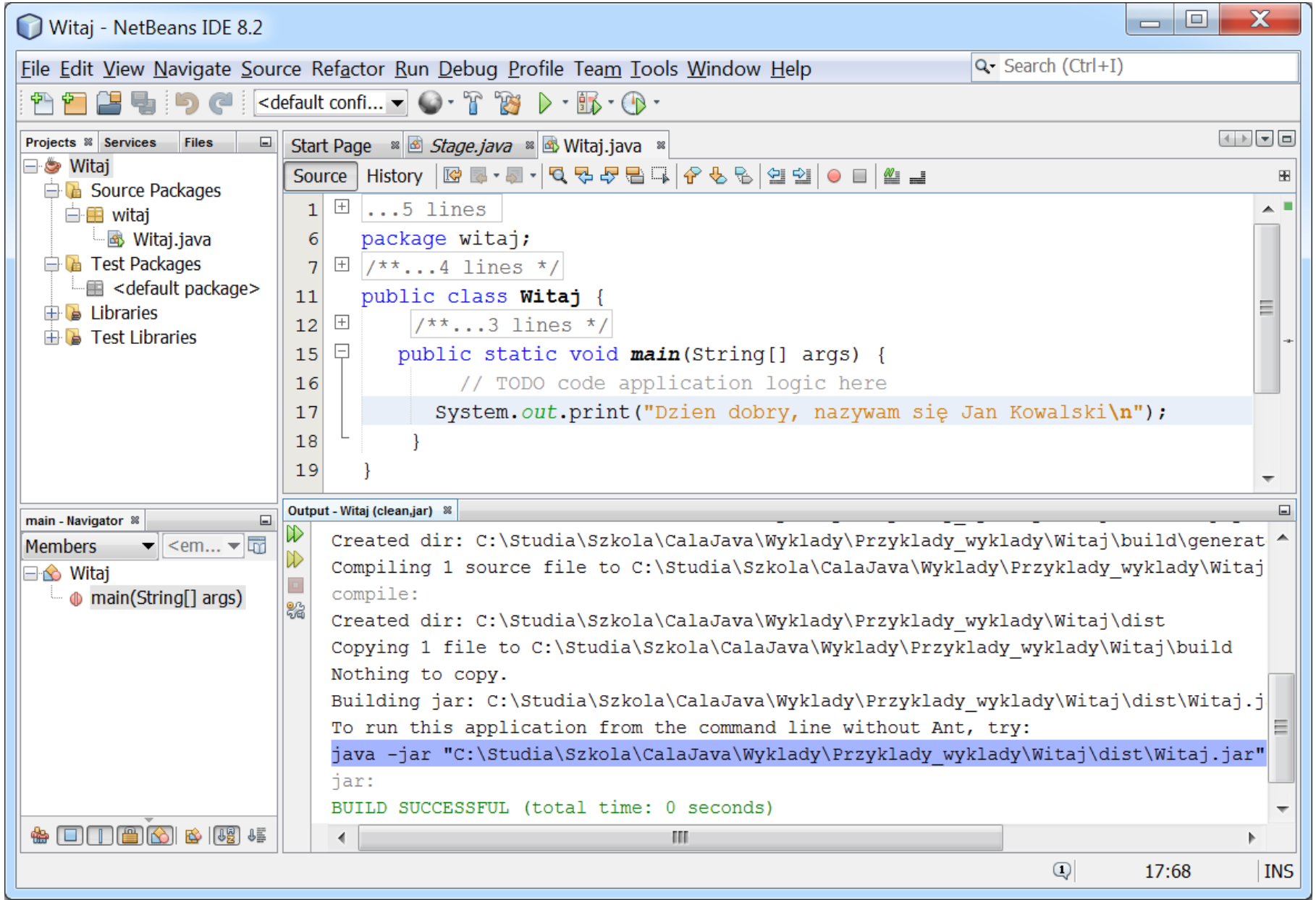
Uruchamianie programu – wybór pozycji Run lub

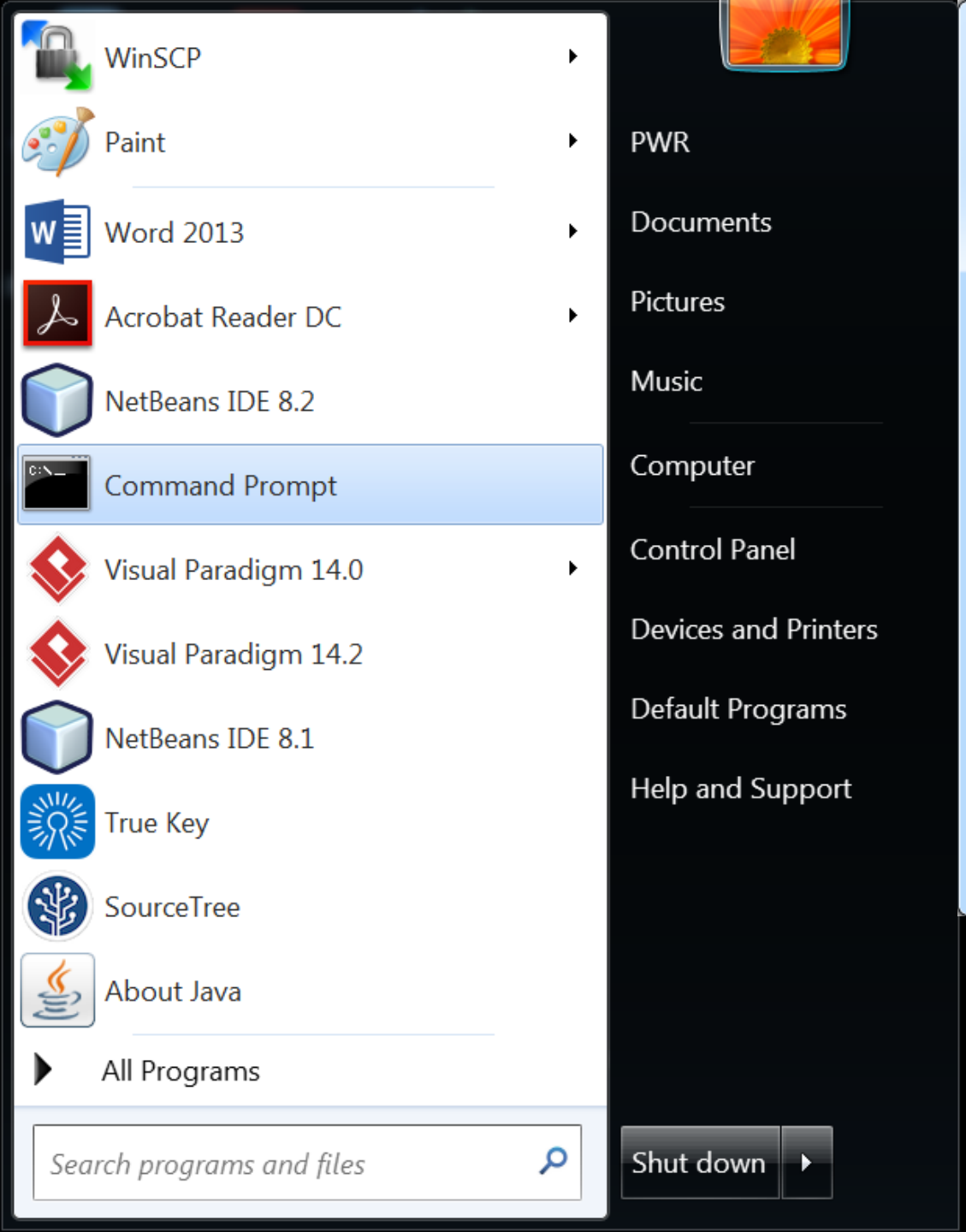


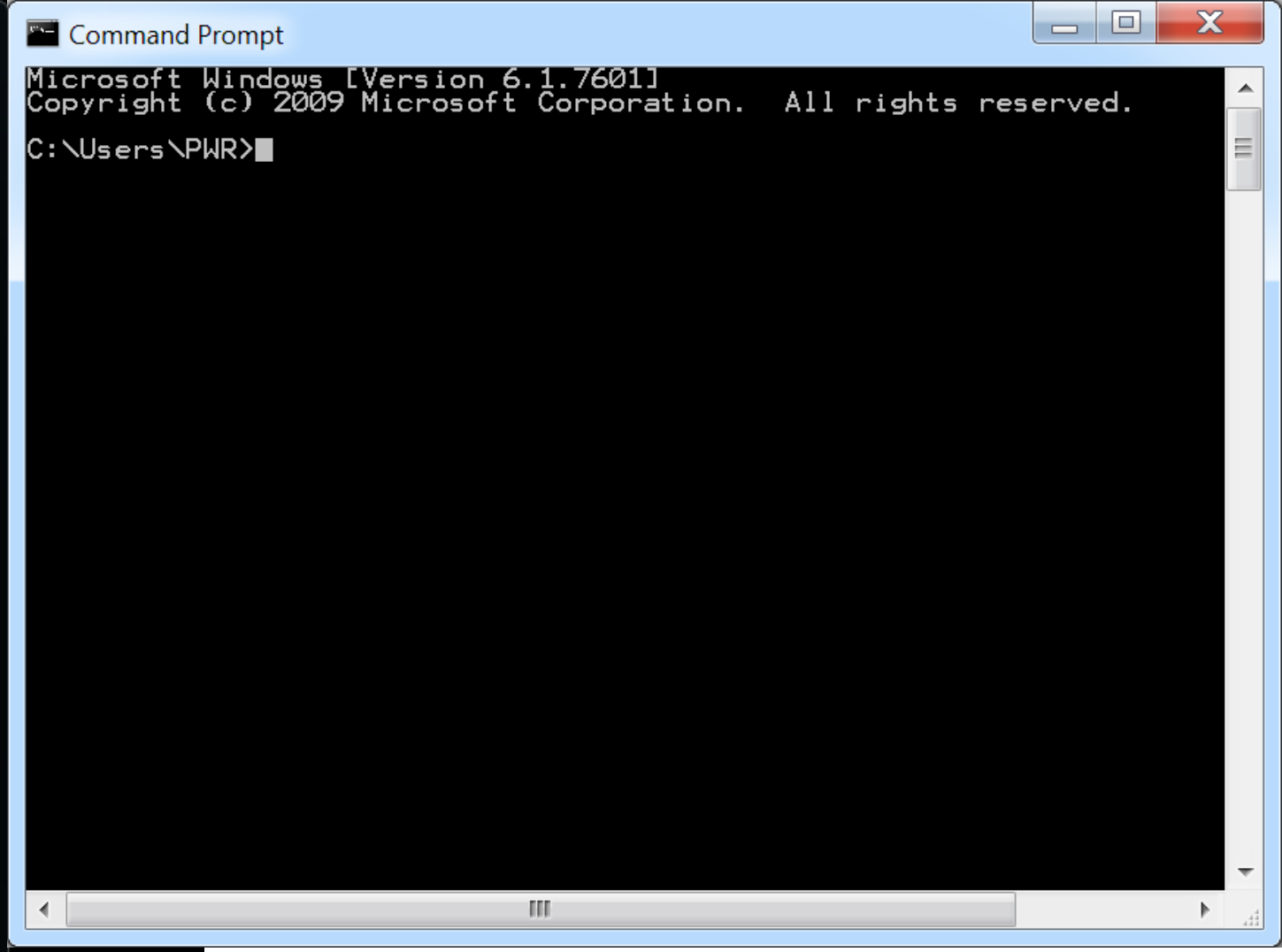
Wykonanie programu w środowisku narzędzia NetBeans 8.2 – okno Output. W środowisku **NetBeans** tworzony jest plik spakowany typu **jar**, który może zawierać wiele plików oraz plik zawierający klasę z funkcją **main.**



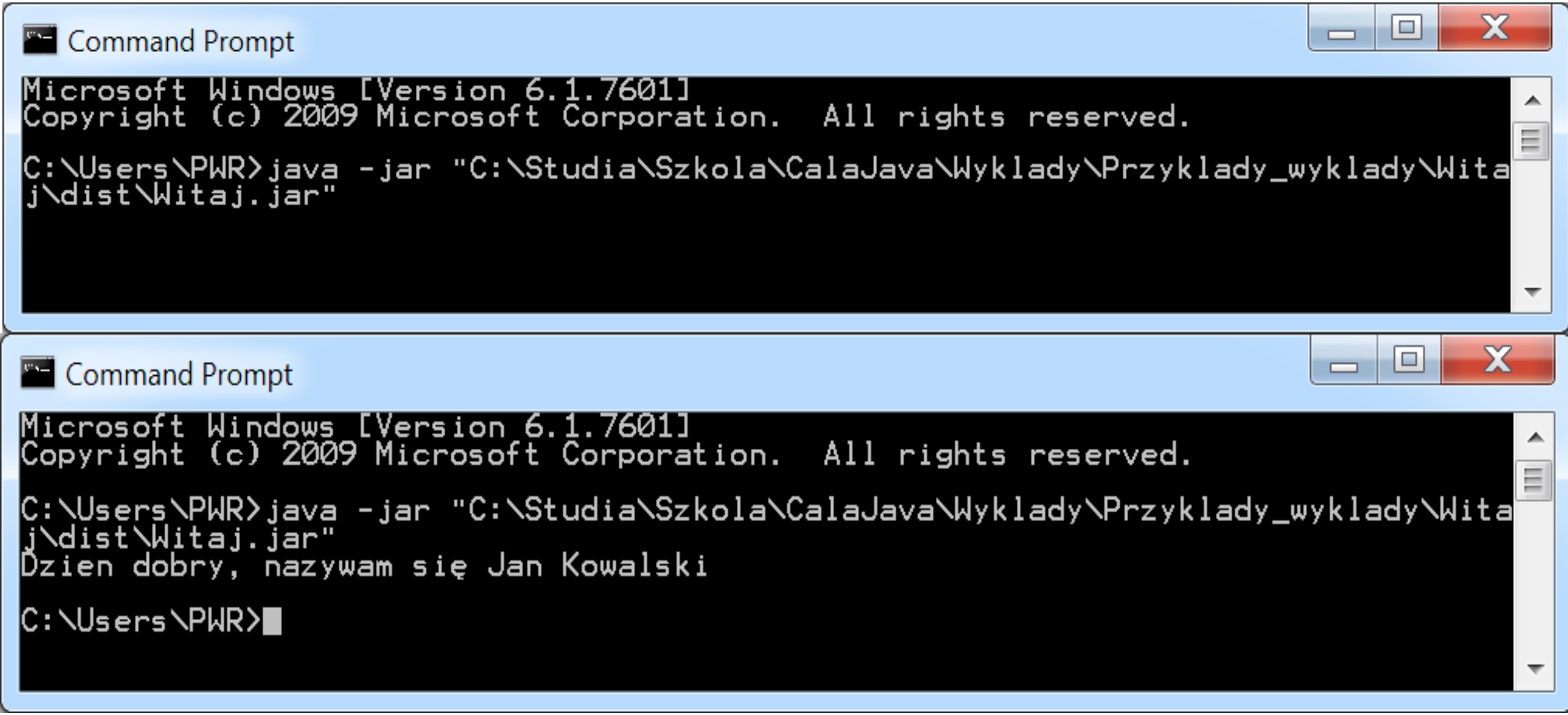
b) Uruchomienie programu konsolowego z linii poleceń – skopiowanie łańcucha uruchamiającego program do schowka




WinSCP
Paint
Word 2013
Acrobat Reader DC
NetBeans IDE 8.2
Command Prompt
Visual Paradigm 14.0
Visual Paradigm 14.2
NetBeans IDE 8.1
True Key
SourceTree
About Java
All Programs
Search programs and files
PWR
Documents
Pictures
Music
Computer
Control Panel
Devices and Printers
Default Programs
Help and Support
Shut down

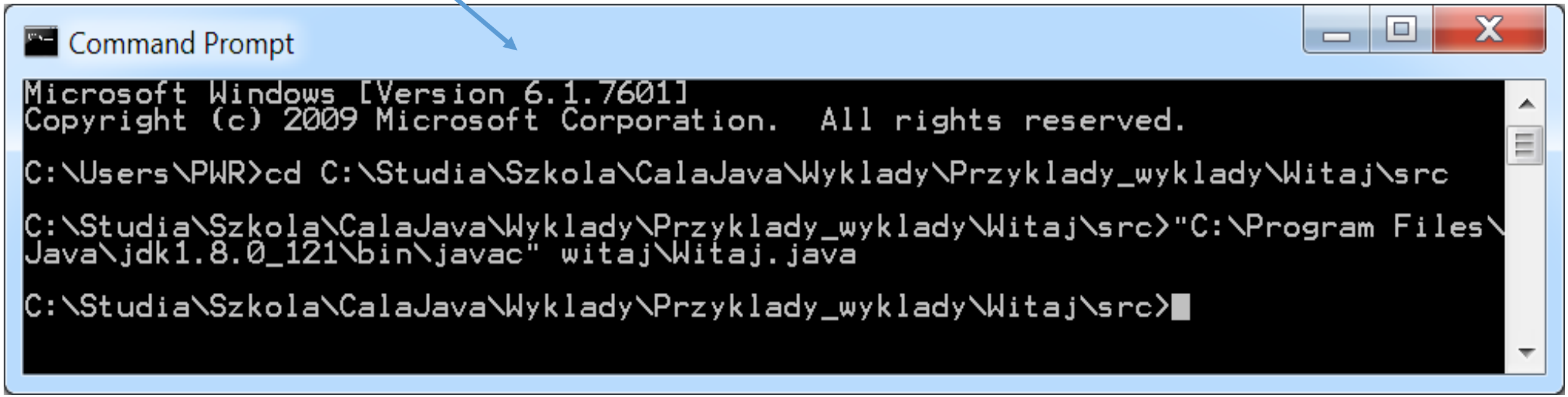

Command Prompt
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
C:\Users\PWR>

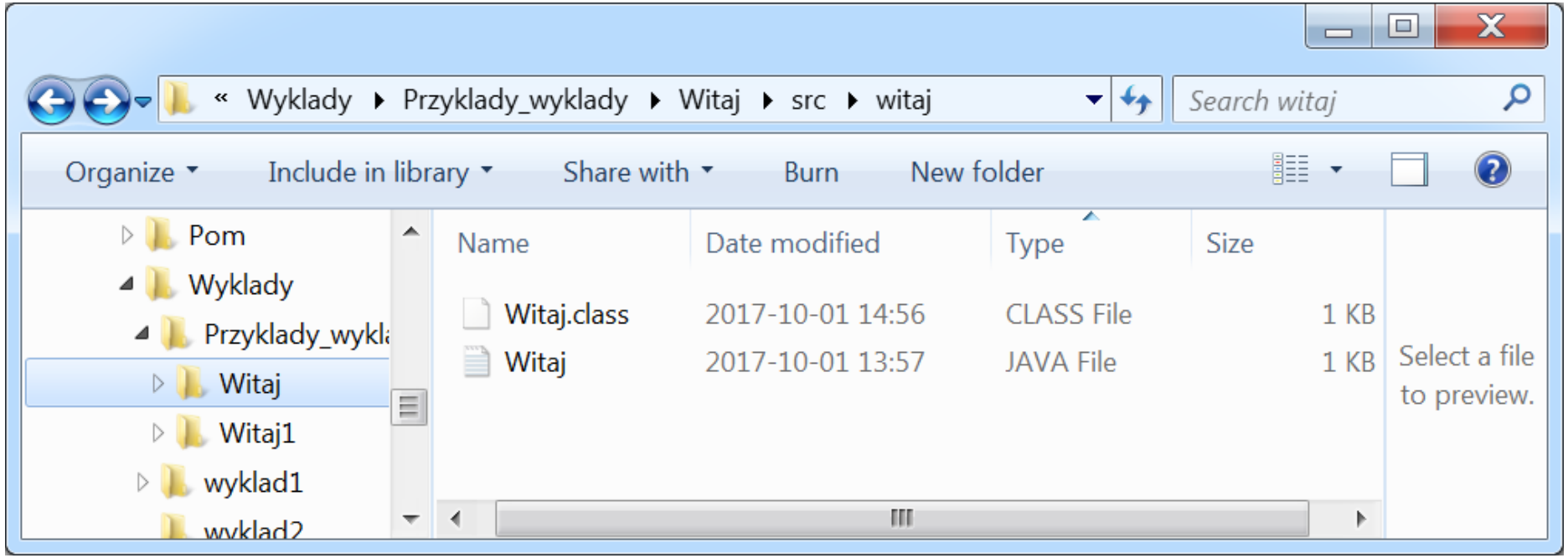
Uruchomienie z linii poleceń programu typu aplikacja: przeniesienie łańcucha uruchomienia programu ze „schowka” przez naciśnięcie prawego klawisza myszy



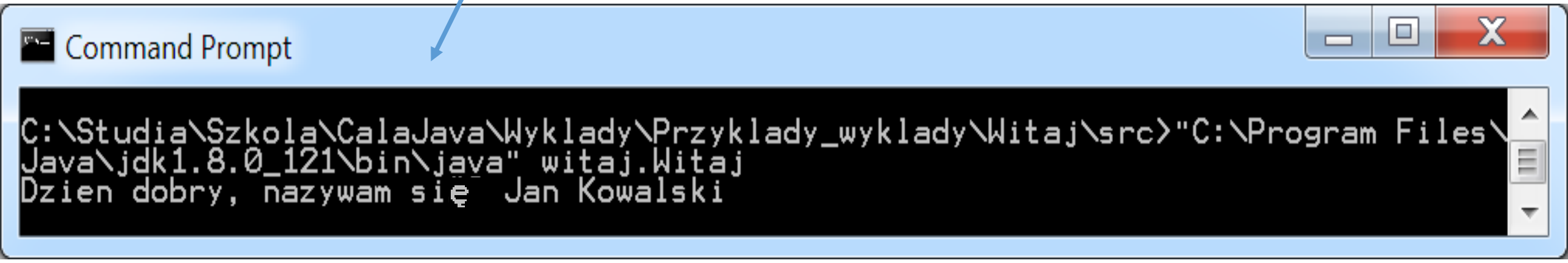
c) Kompilacja i uruchomienie programu z linii poleceń (kod źródłowy programu może być napisany w dowolnym edytorze tekstu po wyborze kodowania np UTF-8)

Kompilacja programu konsolowego z linii poleceń





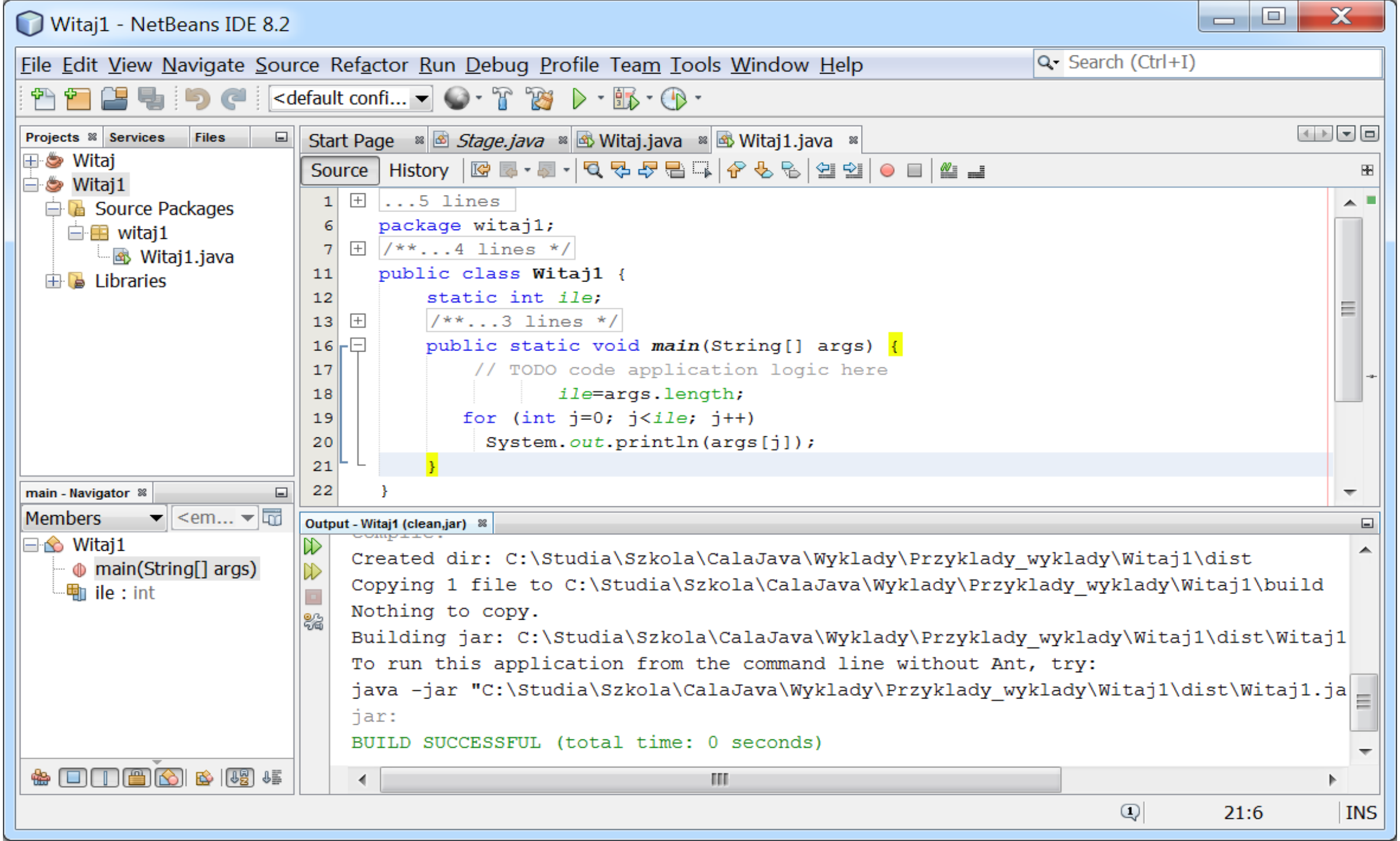
Uruchomienie programu konsolowego z linii poleceń



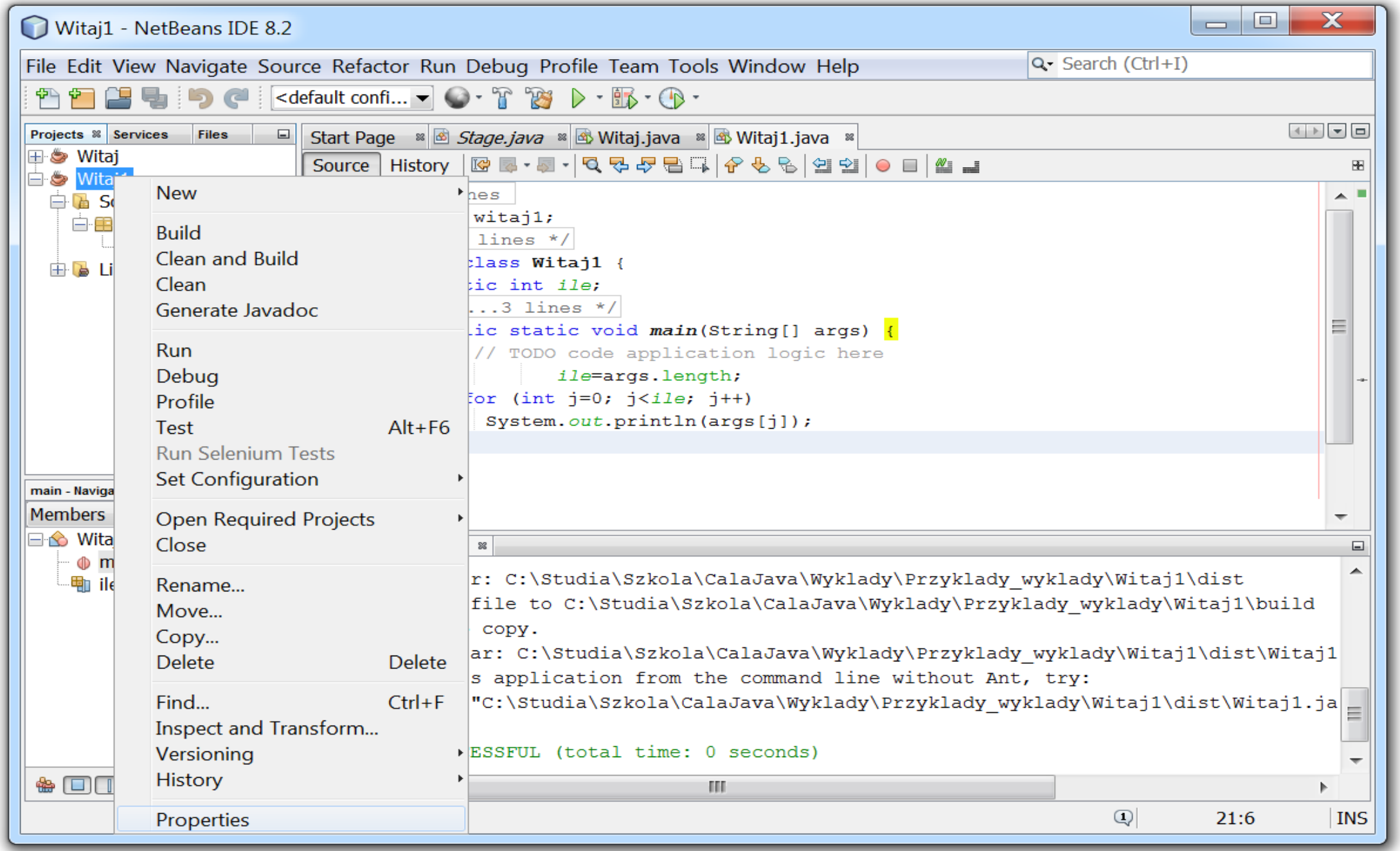
Przykład 2: Wywołanie programu z listą argumentów

java Witaj1 Jan Kowalski

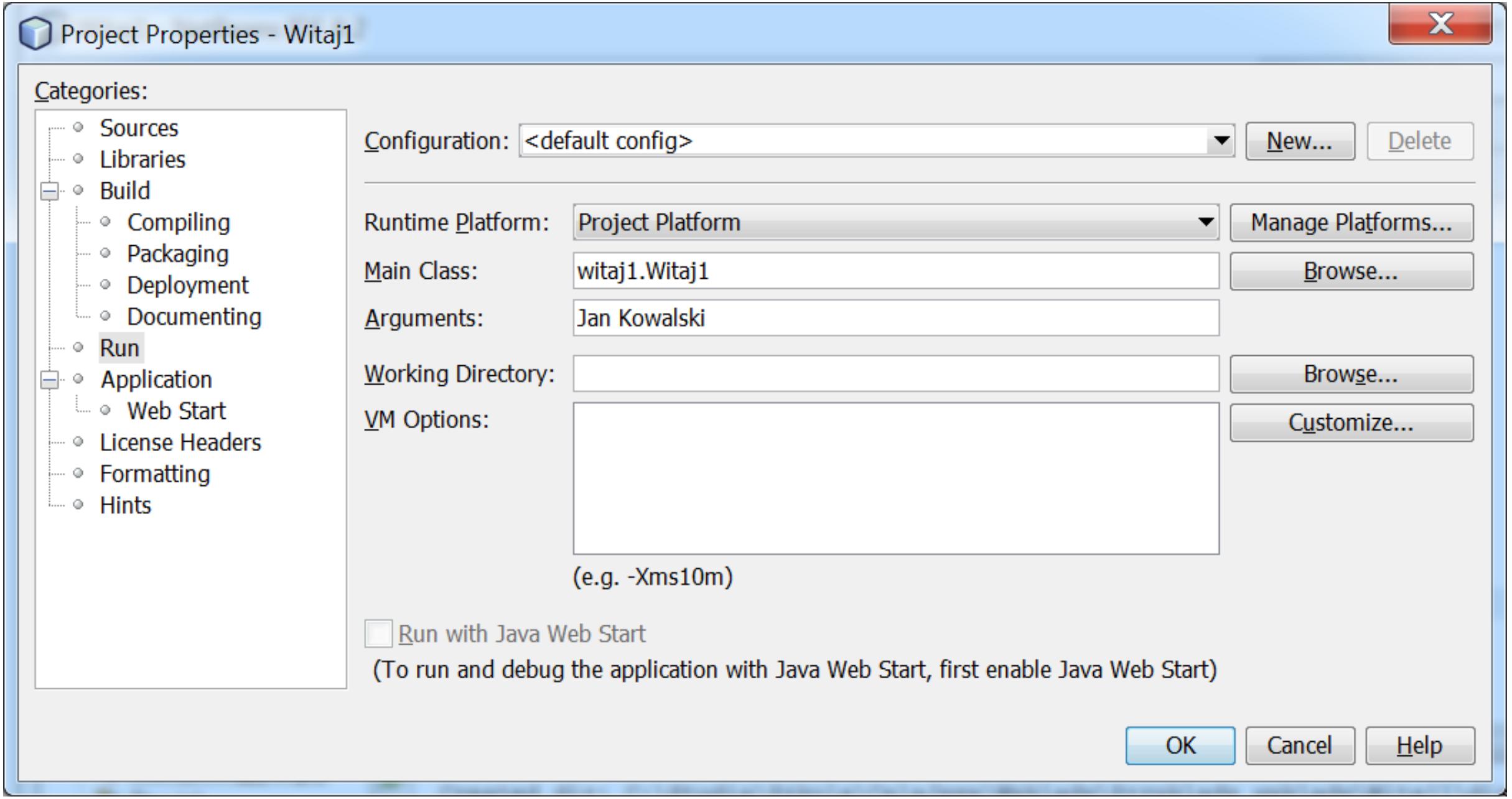
```
public class Witaj1                    // klasa publiczna
 {
   static int ile;                     //składowa klasowa

   public static void main(String args[])
    {
          //pobranie liczby parametrów (w przykładzie 2)
           // ile musi być składową typu static !
      ile=args.length;
      for (int j=0; j<ile; j++)
                    //args[0] – Jan (łańcuch bez białych znaków)
                    //args[1] - Kowalski
        System.out.println(args[j]);
    }
}
```


Witaj1 - NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default confi...
Projects
Services
Files
Witaj
Witaj1
Source Packages
witaj1
Witaj1.java
Libraries
Start Page
Stage.java
Witaj.java
Witaj1.java
Source
History
1 ...5 lines
6 package witaj1;
7 /**...4 lines */
11 public class Witaj1 {
12 static int ile;
13 /**...3 lines */
16 public static void main(String[] args) {
17 // TODO code application logic here
18 ile=args.length;
19 for (int j=0; j<ile; j++)
20 System.out.println(args[j]);
21 }
22 }
main - Navigator
Members
<em...
Witaj1
main(String[] args)
ile : int
Output - Witaj1 (clean,jar)
Created dir: C:\Studia\Szkola\CalaJava\Wyklady\Przyklady_wyklady\Witaj1\dist
Copying 1 file to C:\Studia\Szkola\CalaJava\Wyklady\Przyklady_wyklady\Witaj1\build
Nothing to copy.
Building jar: C:\Studia\Szkola\CalaJava\Wyklady\Przyklady_wyklady\Witaj1\dist\Witaj1
To run this application from the command line without Ant, try:
java -jar "C:\Studia\Szkola\CalaJava\Wyklady\Przyklady_wyklady\Witaj1\dist\Witaj1.ja
jar:
BUILD SUCCESSFUL (total time: 0 seconds)
21:6
INS

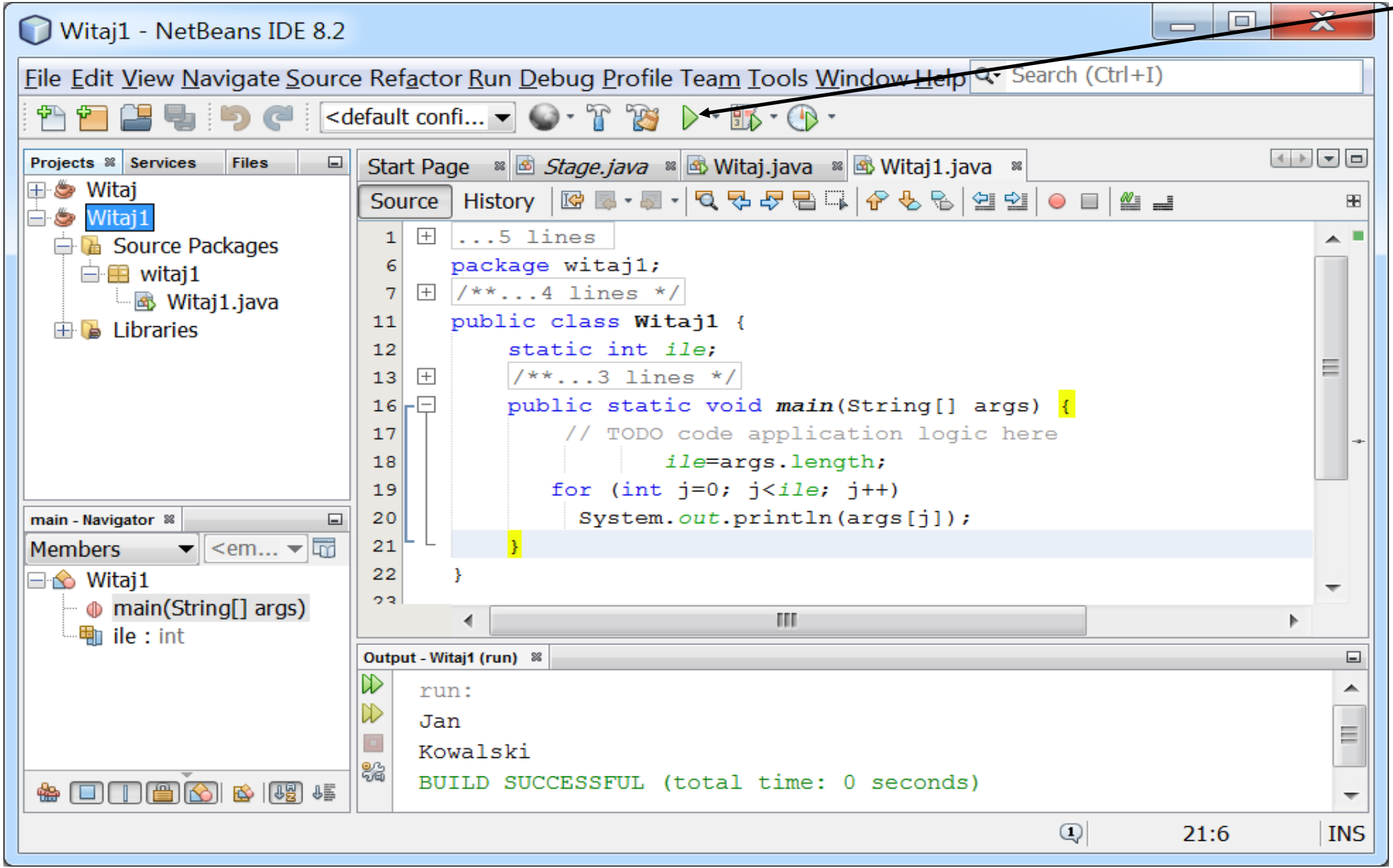
Wybór opcji Properties aktywnego projektu



W oknie Properties wybór opcji Run i wpisanie łańcuchów w linii *Arguments*

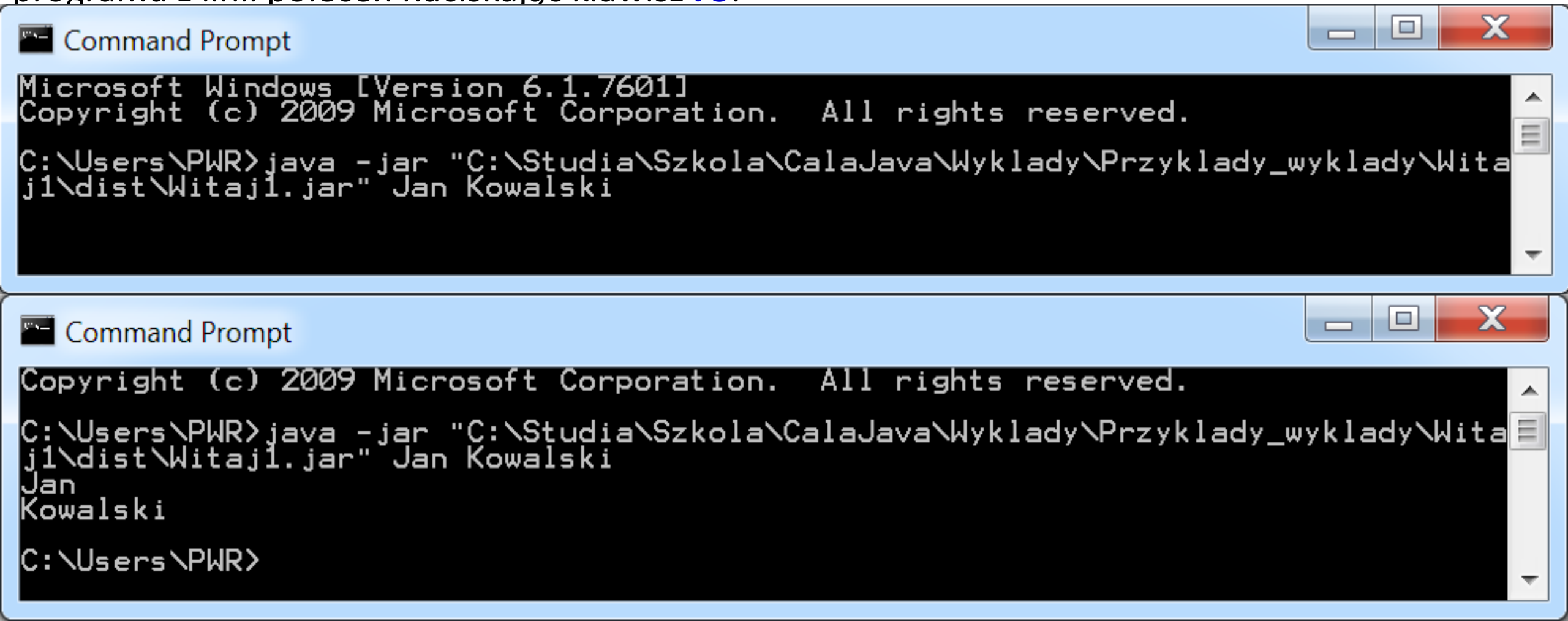


a) Kompilacja typu Clean and Build i uruchomienie programu za pomocą Run lub



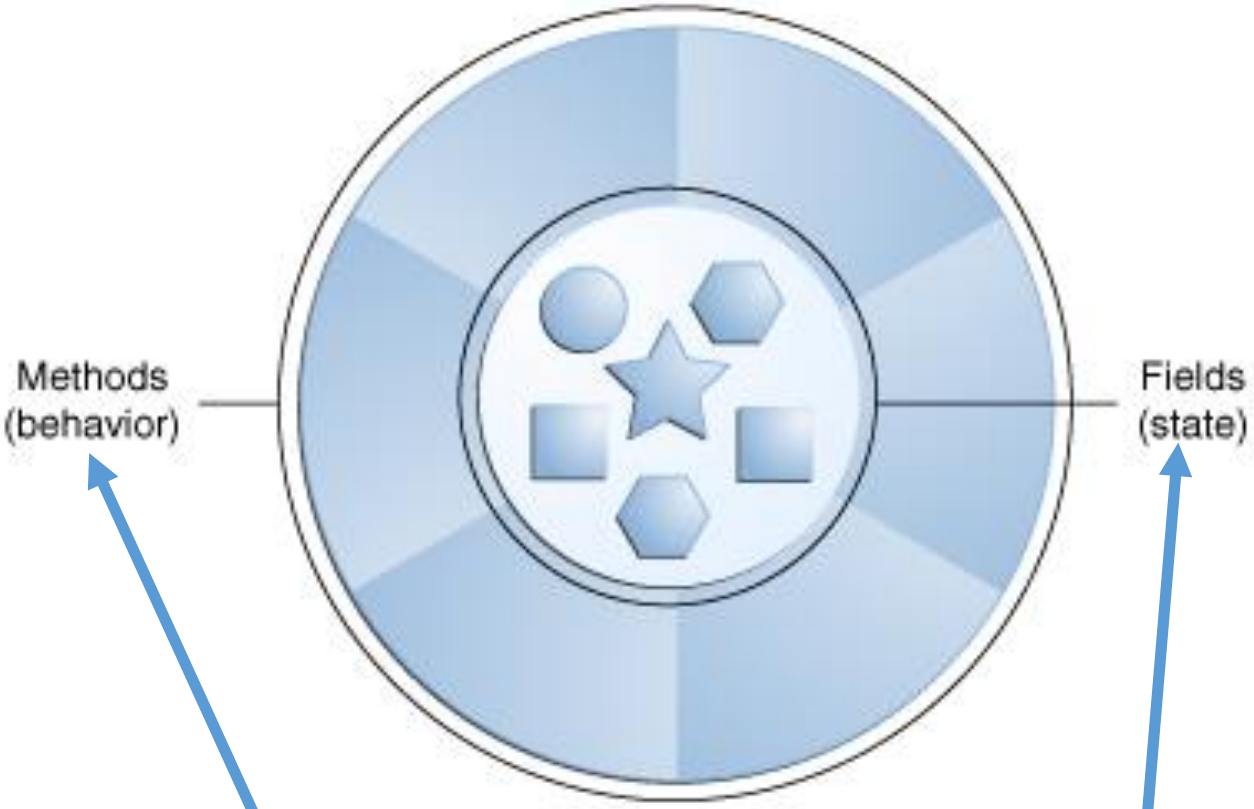
b) Uruchomienie programu z linii poleceń

Przeniesienie ze schowka, naciskając prawy klawisz myszy, łańcucha uruchomienia pobranego z okienka **Output** po wykonaniu **Build Project** dla programu z **przykładu 2** i dopisaniu łańcucha **Jan Kowalski**. Po jednorazowym przeniesieniu łańcucha można powtórzyć uruchomienie programu z linii poleceń naciskając klawisz **F5**.



3. Podstawy programowania obiektowego

Obiekt



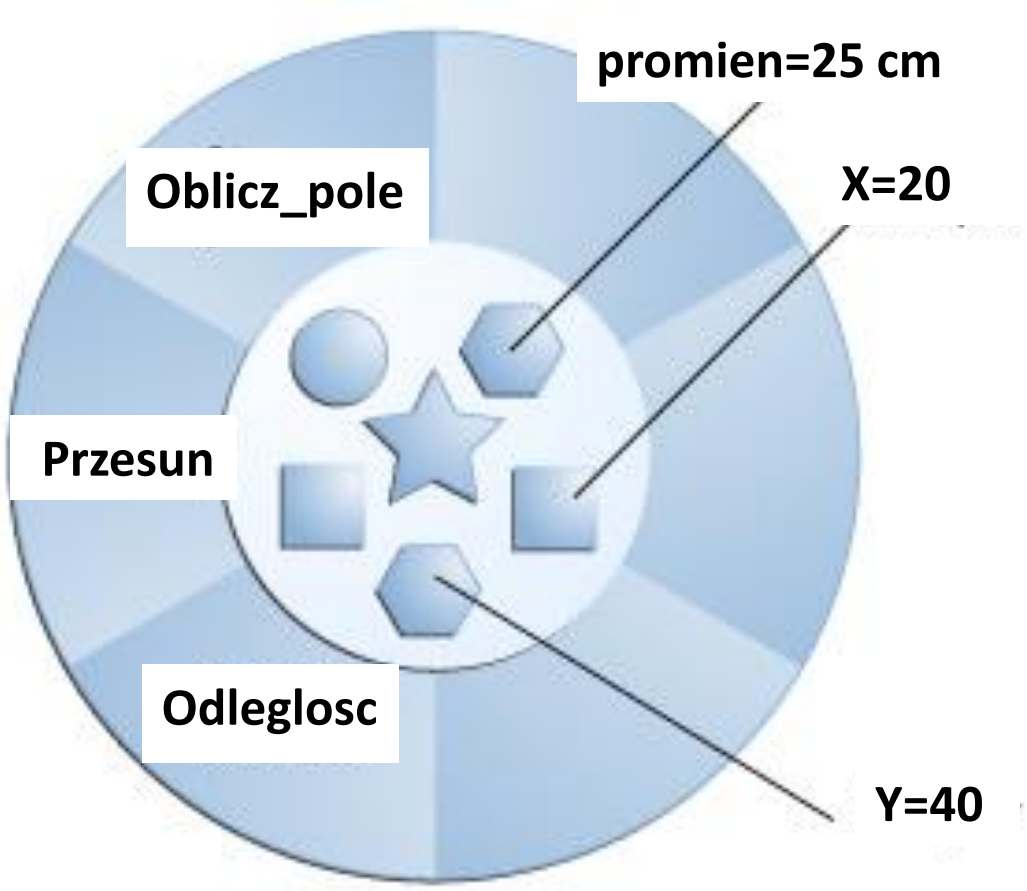
Paradygmaty programowania obiektowego:

1. Hermetyzacja
2. Dziedziczenie
3. Polimorfizm
4. Abstrakcja klas
5. Agregacja

Obiekt:
zachowanie (metody) oraz **stan** (atrybuty)

Obiekt (cd)

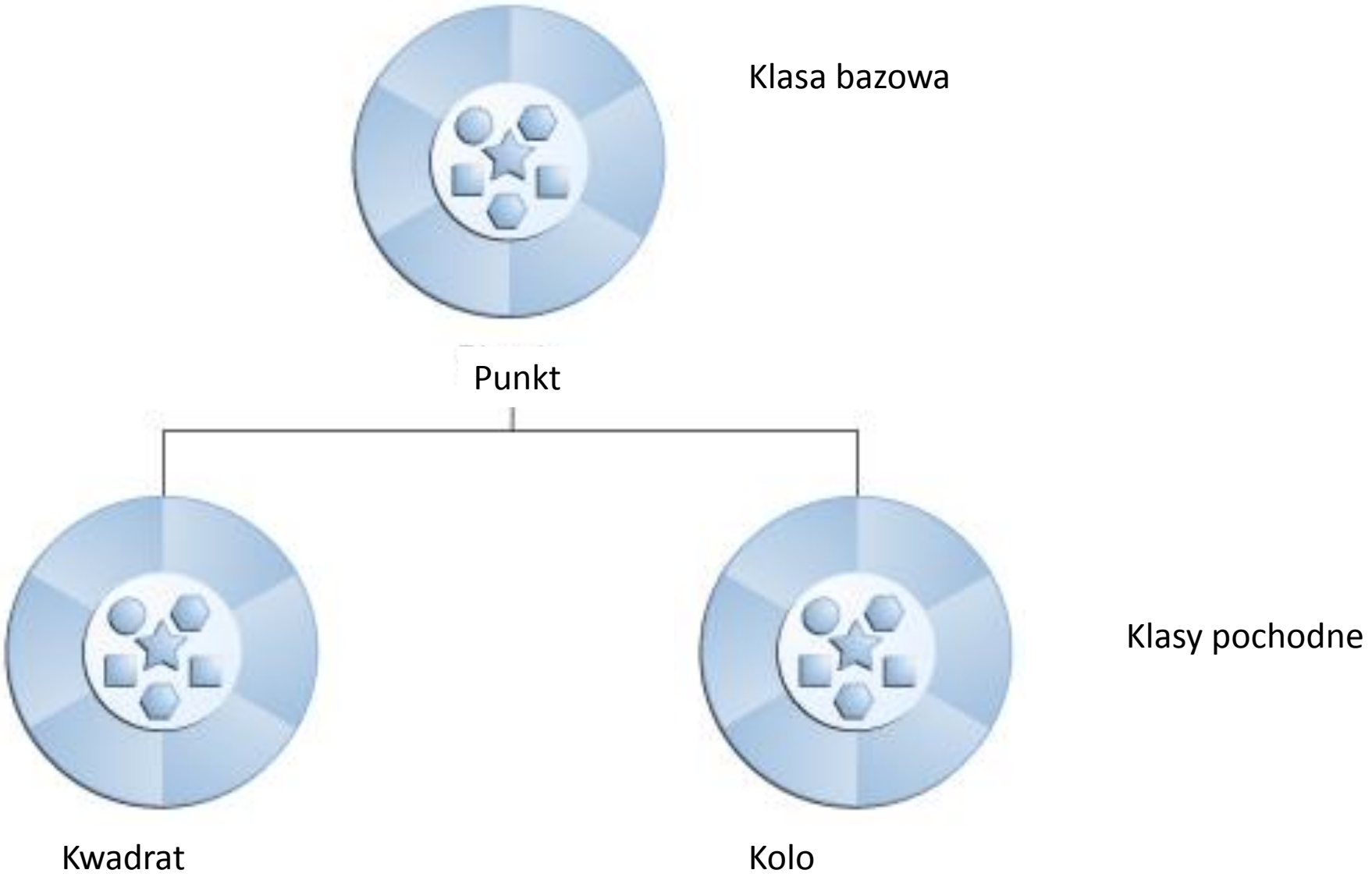
Przykład obiektu typu Koło



Zalety (przy zachowaniu zasad poprawnego programowania):

- **Modularność (kod obiektu może być przenoszony i uruchamiany niezależnie od położenia kodu źródłowego)**
- **Hermetyzacja kodu obiektu**
- **Wielożywalność kodu**
- **Łatwe usuwanie lub wstawianie kodu obiektu z/do programu bez konieczności modyfikacji całego kodu**

Dziedziczenie



Dziedziczenie (cd)

Definicja klasy bazowej

```
class Punkt
{   protected  int x, y;
    Punkt(int wspX, int wspY)
        {     x = wspX; y = wspY; }

  void zmien(int wspX, int wspY)
        {     x = wspX;y = wspY;}

  void przesun(int dx, int dy)
        {    x+=dx;  y+=dy;  }

  double odleglosc(Punkt p)
       { return Math.sqrt((x-p.x)*(x-p.x)+(y-p.y)*(y-p.y));  }

  int pole ()
        {      return 0; }

  public void rysuj()
      {   System.out.println('\n'+" Punkt");  }
}
```

Definicja klasy pochodnej

```
class Kwadrat extends Punkt
{
   int dlugosc;
   Kwadrat(int wspX, int wspY, int dlugosc_)
      {  super(wspX,wspY);
         dlugosc= dlugosc_;    }

 @Override
  int pole()                          //metoda przedefiniowana
     {   return dlugosc*dlugosc;  }

 @Override
  public void rysuj()                 //metoda przedefiniowana
    {  System.out.println('\n'+ " Kwadrat");  }
}
```

Interface

```
interface Figura
{  String figura ="figura:  ";
                    //domyślnie public static final
   void rysuj(); //domyślnie public
}
```

Definicja klasy pochodnej

```
class Kwadrat extends Punkt
 {
   int bok;
   Kwadrat(int wspX, int wspY, int dlugosc_)
      {  super(wspX,wspY);
         bok= dlugosc_;    }
   @Override
   int pole()
    {   return bok*bok;  }
  @Override
   public void rysuj() //implementacja metody
                       //interfejsu Figura
    {  System.out.println('\n'+figura+ " Kwadrat");  }
 }
```

Definicja klasy bazowej

```
class Punkt implements Figura
{   protected  int x, y;
    Punkt(int wspX, int wspY)
        {     x = wspX; y = wspY; }

  void zmien(int wspX, int wspY)
        {     x = wspX; y = wspY;}

 void przesun(int dx, int dy)
        {   x+=dx;  y+=dy;  }

 double odleglosc(Punkt p)
        {  return Math.sqrt((x-p.x)*(x-p.x)+(y-p.y)*(y-p.y));  }

 int pole ()
        {      return 0;}
 @Override
 public void rysuj() //implementacja metody interfejsu Figura
         {   System.out.println('\n'+figura +" Punkt");  }
 }
```

Podsumowanie

Definicja klasy, dziedziczenie, implementowanie metod interfejsów

```
class nazwa_klasy
   {
                  //ciało klasy
  }
```

Klasa (opisuje typ obiektu czyli jego składowe):

przed słowem class może wystąpić jeden ze specyfikatorów:

public – klasa dostępna publicznie

final, public final - klasa ta nie może mieć następcy

abstract class, public abstract class – klasy z takimi słowami kluczowymi nie mają wystąpień

Klasa abstrakcyjna może zawierać metody abstrakcyjne, poprzedzone słowem kluczowym **abstract;** w miejscu ciała metody abstrakcyjnej występuje średnik; każda jej podklasa musi podawać implementacje tych metod. Każda klasa, która odziedziczy metodę abstrakcyjną, ale jej nie implementuje, staje się klasą abstrakcyjną

brak specyfikatora – klasa dostępna tylko dla klas zdefiniowanych w tym samym pakiecie

dziedziczenie - po nazwie klasy wystąpią słowa: **extends** nazwa_superklasy

(czyli klasa dziedziczy zawsze publicznie i tylko od jednej od klasy nazwa_superklasy)

Każda klasa dziedziczy od predefiniowanej klasy Object. Jeżeli w definicji klasy nie występuje słowo **extends,** to oznacza to niejawne wystąpienie w tej definicji słów **extends** Object

implementowanie- po nazwie klasy wystąpią słowa: implements nazwy_interfejsów

(czyli w danej klasie zostaną zdefiniowane metody, zadeklarowane w implementowanych interfejsach. Jeżeli dana klasa implementuje więcej niż jeden interfejs, wtedy nazwy kolejnych interfejsów oddziela się przecinkami. Implementowanie metod kilku interfejsów odpowiada dziedziczeniu wielobazowe w C++

Podsumowanie (cd)

Ciało klasy:

- zamknięte w nawiasy klamrowe
- może zawierać zmienne składowe (to jest pola lub może zawierać zmienne instancji)
- może zawierać zmienne klasowe (statyczne, tj. poprzedzone słowem kluczowym **static**)
- może zawierać konstruktory (metody o nazwie klasy bez zwracanego typu)
- może zawierać metody klasowe (nagłówek poprzedzony słowem kluczowym **static**)
- może zawierać metody zwykłe – można je wywołać, gdy utworzono obiekt
- nazwa każdej zmiennej składowej, zmiennej klasy, metody lub funkcji klasy musi być poprzedzona nazwą typu podstawowego (**byte**, **short**, **int, long, double, float, char, boolean, void**) lub **klasowego**
- przed nazwą typu składowej klasy może wystąpić jeden ze specyfikatorów dostępu:
  private (dostęp tylko dla elementów klasy - **private int** d;),
  protected (dostęp tylko w podklasie, nawet dla podklas z innego pakietu; nie dotyczy zmiennych klasy)
  public (dostęp publiczny). Brak specyfikatora oznacza, że dany element jest dostępny tylko dla klas w tym samym **pakiecie**
- słowo **final** po specyfikatorze dostępu przed nazwą typu zmiennej wystąpienia lub zmiennej klasy deklaruje jej nie modyfikowalność

  np. **public static final int** stala1 = 10;
  final int stala2= 10;
- słowo **final** po specyfikatorze dostępu przed nazwą metody oznacza, że nie może ona być redefiniowana w klasie dziedziczącej

  np. **public final void** koncowa_wersja () {/* ... */ } – definicja publicznej metody koncowa_wersja może wystąpić tylko raz w rodzinie klas

Podsumowanie (cd)

- **Dostęp do zmiennych składowych klasy** (statycznych) jest możliwy bez tworzenia obiektów tej klasy

np. **System.out.println**("Dzien dobry, nazywam się Jan Kowalski\n");

- **Klasy i interfejsy** są typami referencyjnymi.

Wartościami zmiennych tych typów są referencje (odnośniki) do wartości lub zbiorów wartości reprezentowanych przez te zmienne.

np. instrukcja ***Random rand;*** jedynie powiadamia kompilator, że będzie używana zmienna ***rand,*** której typem jest ***Random*** (brak przydzielonego miejsca w pamięci na taki obiekt)

- **Do zmiennej *rand*** można przypisać dowolny obiekt typu ***Random*** przydzielając mu pamięć za pomocą **new**

np. **Random rand = new Random();**

Argumentem operatora **new** jest generowany przez kompilator konstruktor ***Random()***, który inicjuje obiekt utworzony przez operator **new**. Operator **new** zwraca referencję do tego obiektu, po czym przypisuje go do zmiennej ***rand***.

np. **Random rand = new Random(20L);**

Argumentem operatora **new** jest definiowany przez programistę konstruktor **Random(20L)**, który inicjuje obiekt utworzony przez operator **new**. Operator **new** zwraca referencję do tego obiektu, po czym przypisuje go do zmiennej ***rand***.

- **Dostęp do elementów klasy** uzyskuje się za pomocą **operatora kropkowego**