

# Języki i metody programowania – Java

## INF302W

### Wykład 1 (część 1)

na podstawie

<https://docs.oracle.com/javase/tutorial/>

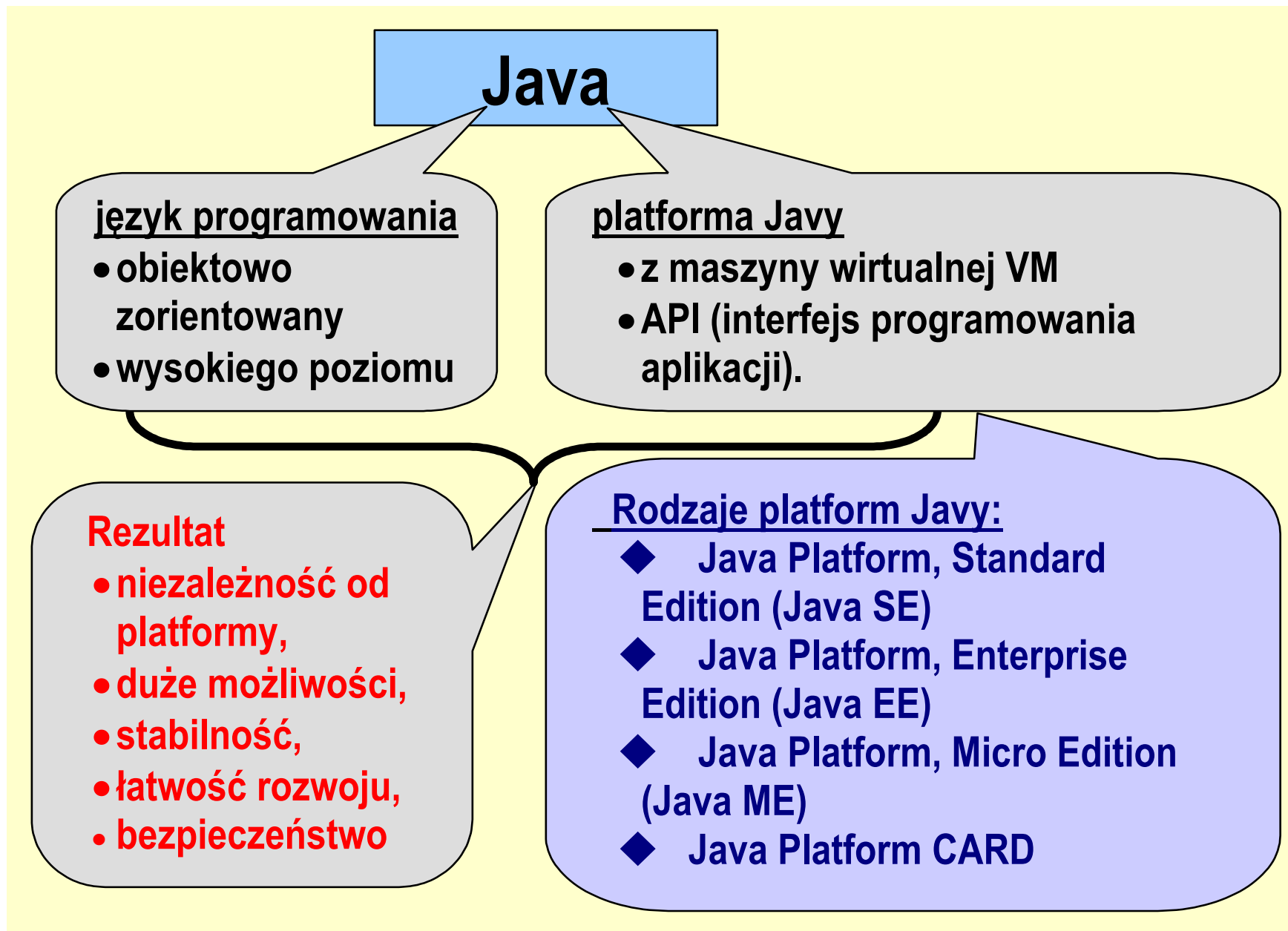
Autor

Dr inż. Zofia Kruczkiewicz

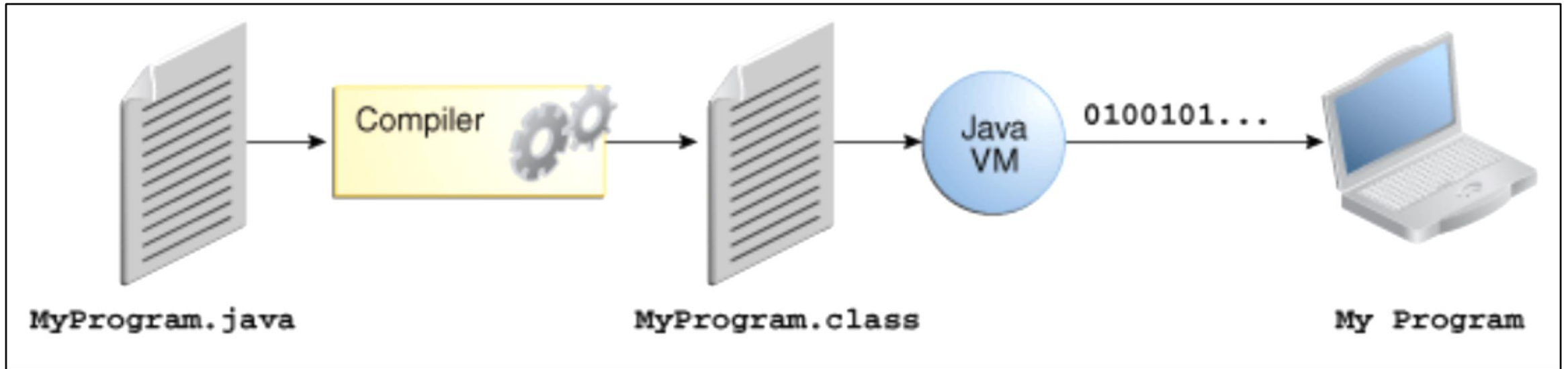
# Struktura wykładu

- 1. Porównanie języków Java i C++, proste programy typu aplikacja- budowa klasy, składowe statyczne i niestacyjne, 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 niestacyjnych, operator new, odwołanie do obiektów-operator kropka, wywołanie metod, przeciążenie metod (1- część).**

# 1. Charakterystyka technologii Java



# 1 (cd). Język Java – proces tworzenia programu

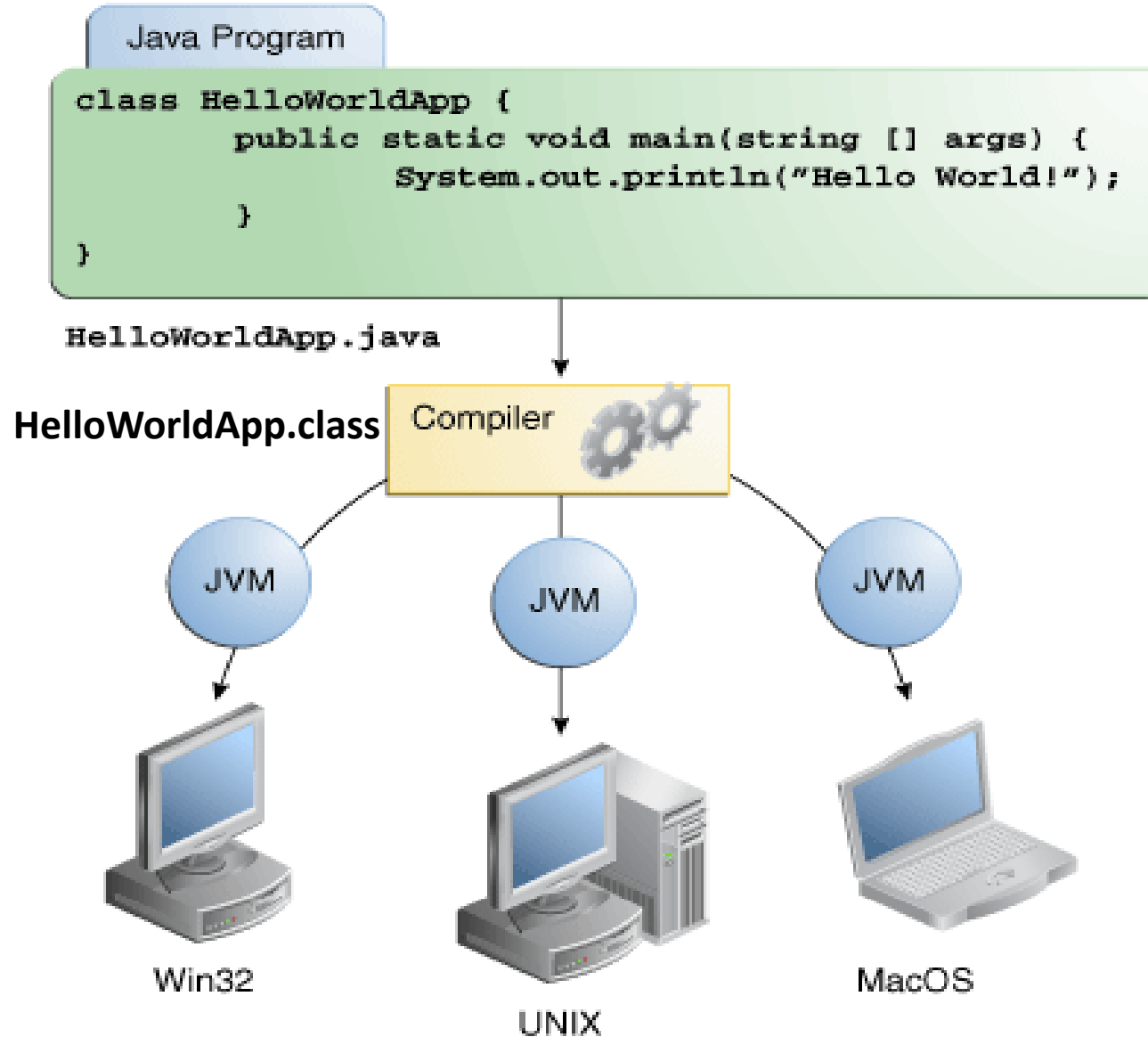


który jest:

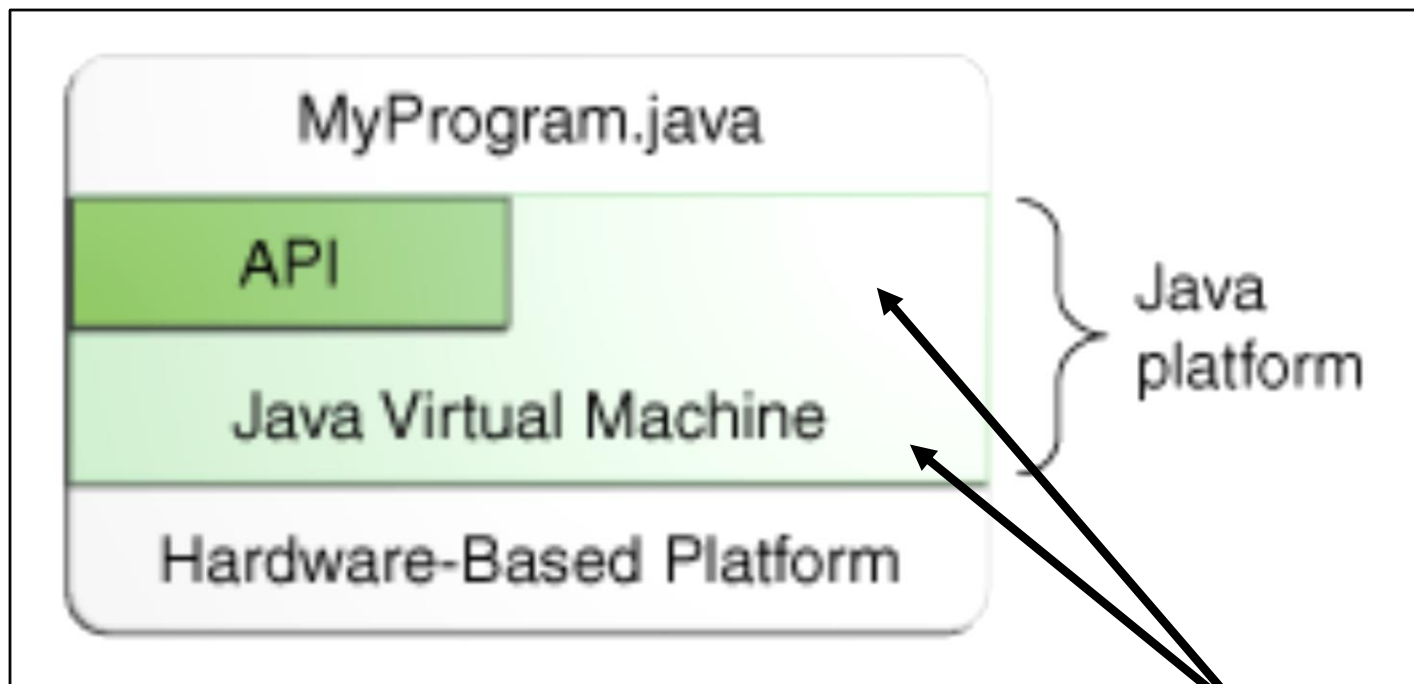
**prosty**  
**zorientowany obiektowo**  
**rozproszony**  
**wielowątkowy**  
**dynamiczny**

**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



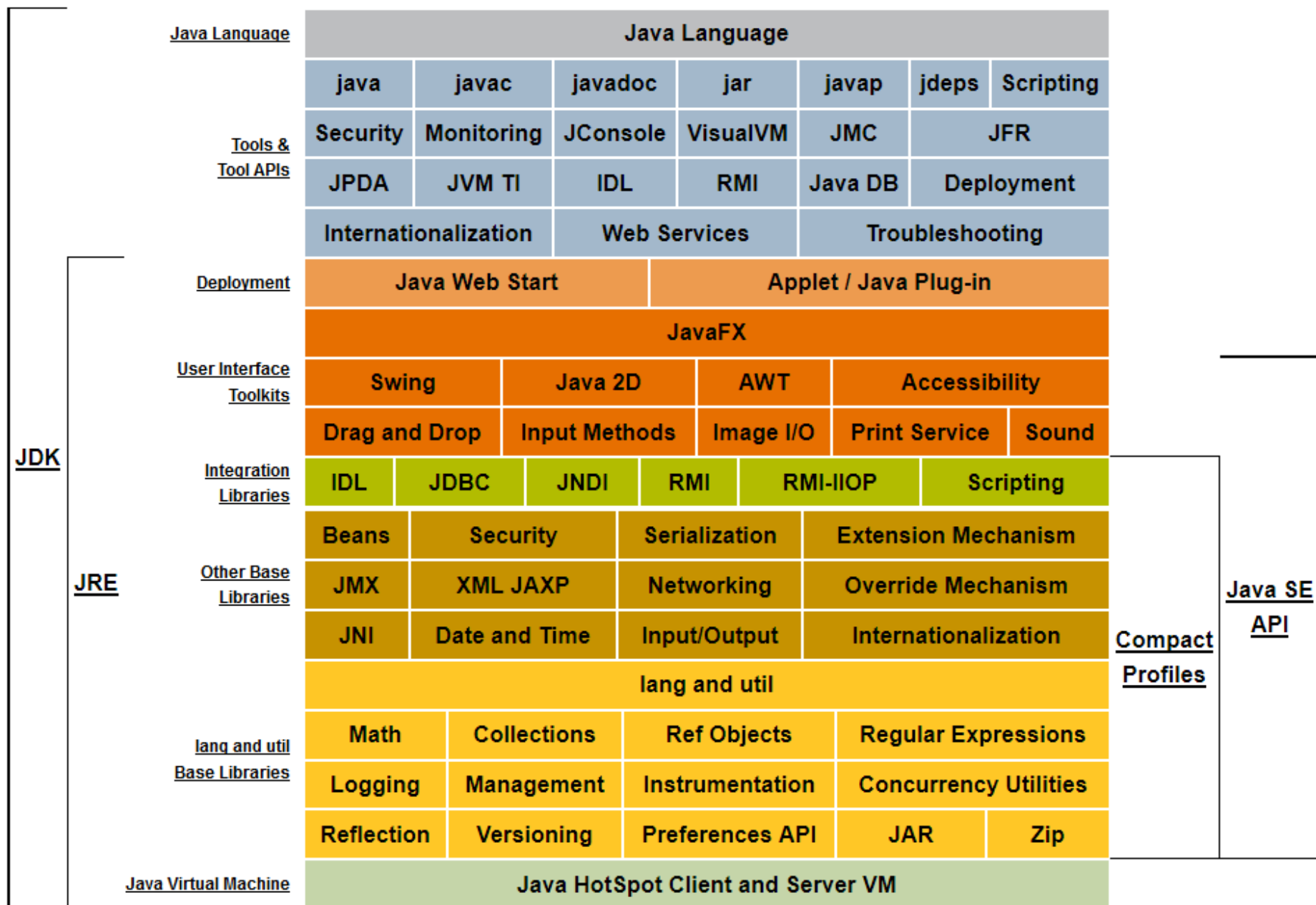
# 1 (cd). Platforma Javy SE



## 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.

Komponenty platformy Java

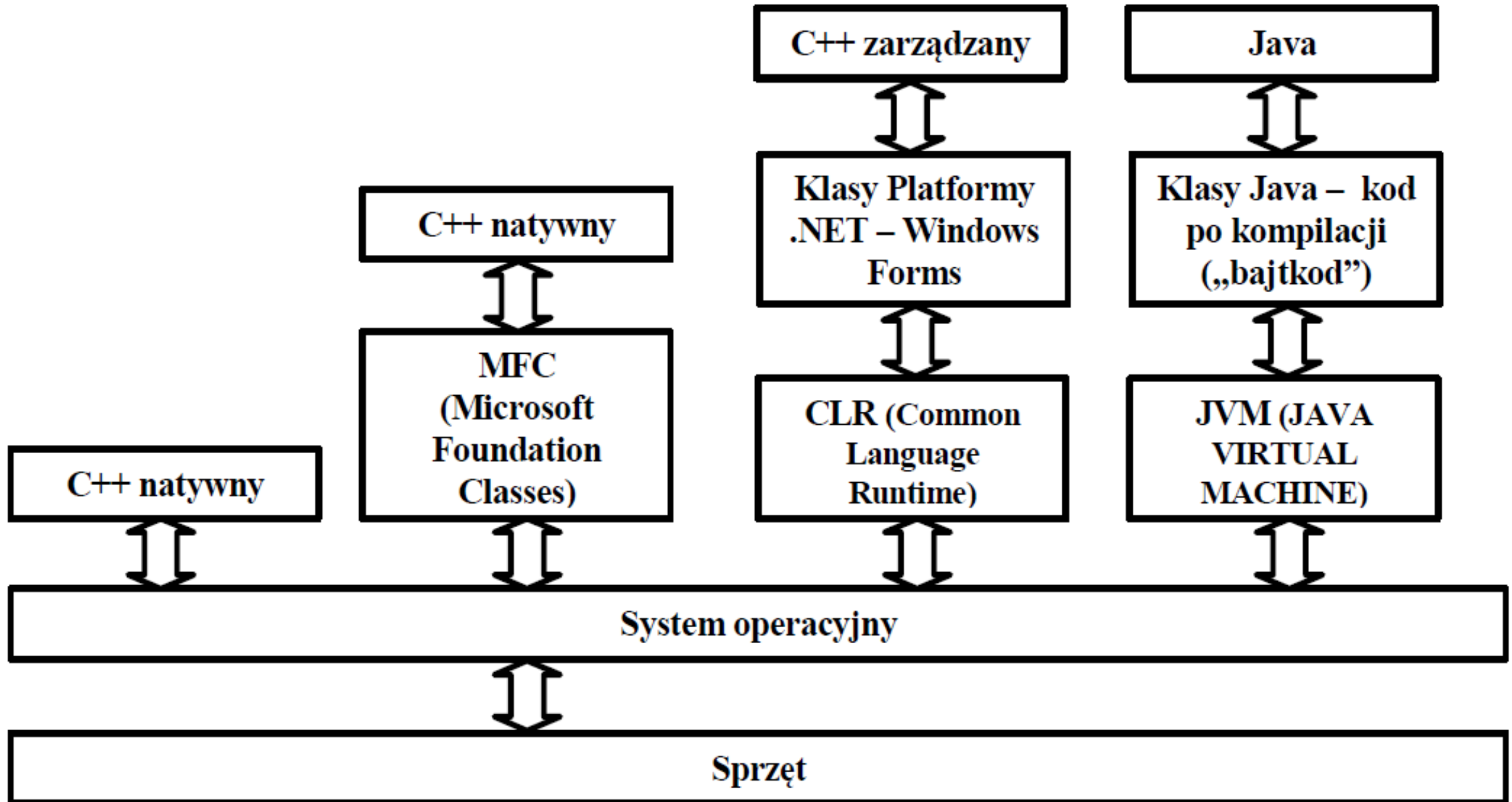


# 1 (cd). Zalety technologii Java SE

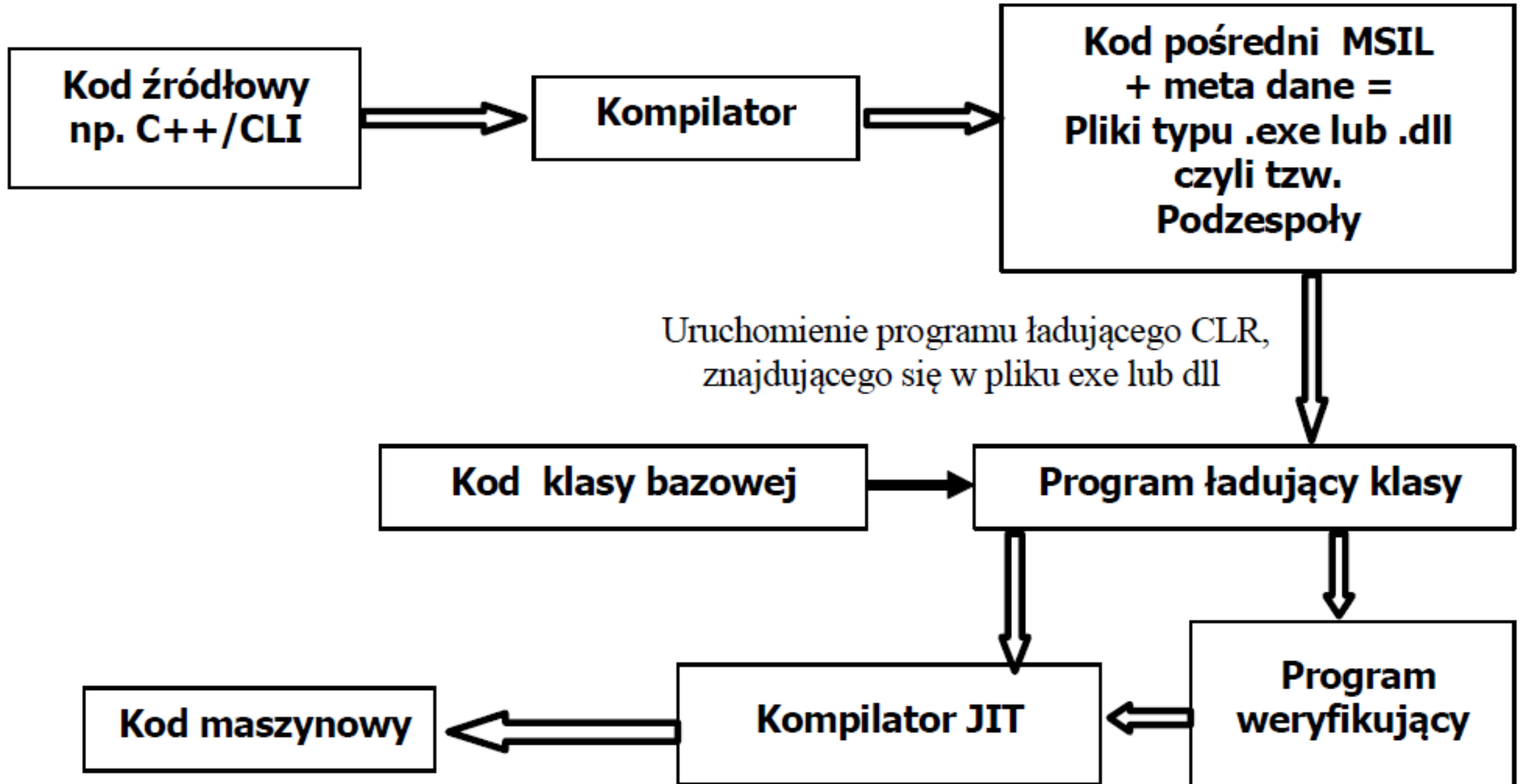
- **Łatwy** dla programistów znających język C lub C++.
- **Mniej kodu:** metryki programów (liczenie klas, liczenie metod itd.) o identycznej funkcjonalności napisanego w Javie i w C++ -mogą być **czterokrotnie mniejsze w Javie niż w C++**.
- **Pisanie lepszego kodu:** język programowania Java zachęca 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.



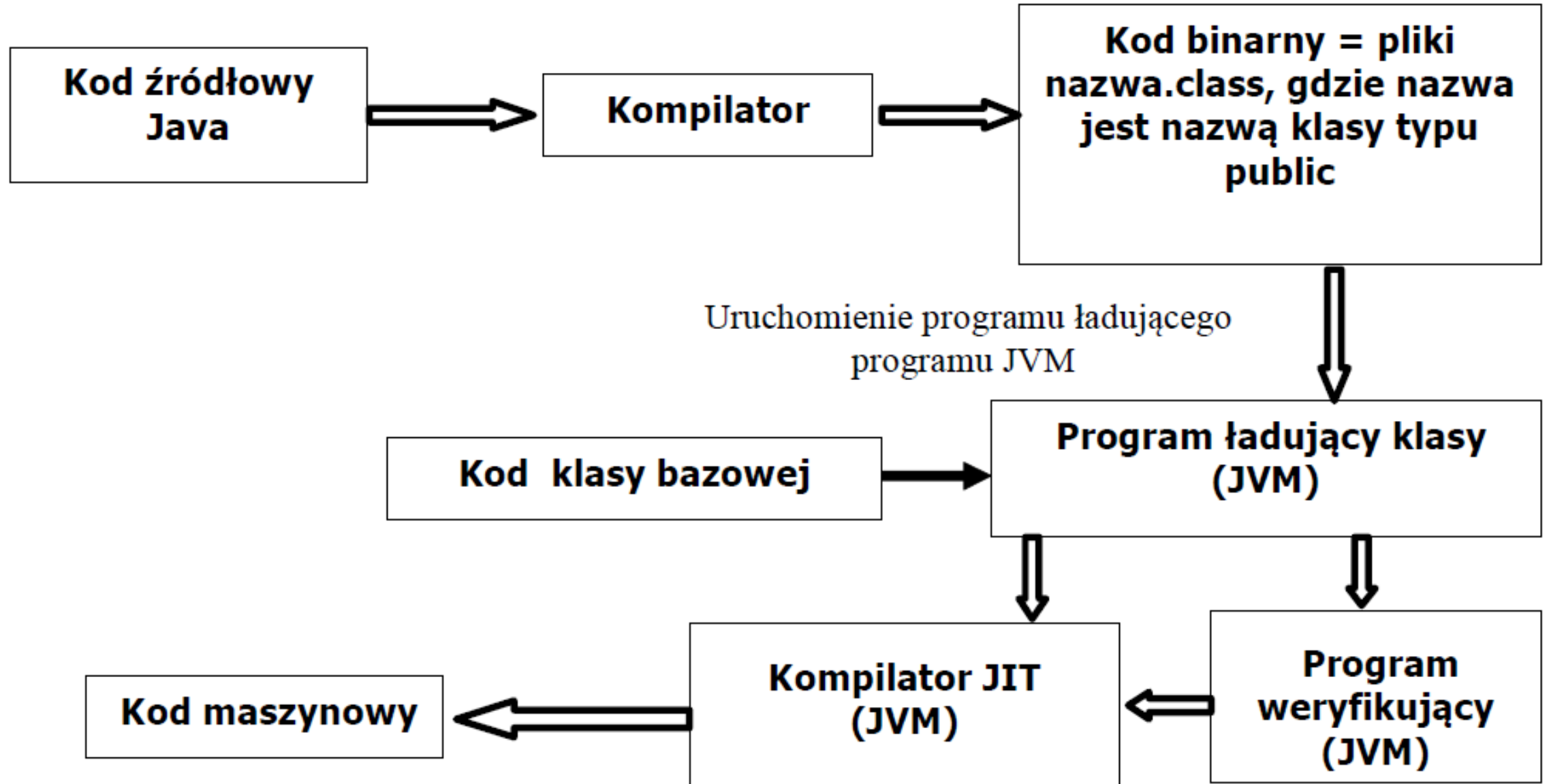
# 1 (cd). Tworzenie programów w językach C++, C++/CLR oraz Java (Ivor Horton „Od podstaw Visual C++ 2005”)



# 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

<p>C++/CLI - Definicje za pomocą typów fundamentalnych (są wtedy mapowane do typów klas wartości w CLR). <b>Preferowane</b> w programach C++/CLI – uniezależniają od konkretnej implementacji CLI .</p>	<p>C++/CLR - Definicje za pomocą typów C++/CLR (klasy wartości)</p>	<p>Java- Definicje typów za pomocą klasy typu <b>Class</b>. Obiekty typu <b>Class</b> reprezentują klasy i interfejsy działające w programie Javy oraz typy enum, tablice, typy nieobiektove (<b>boolean, byte, char, short, int, long, float, double</b>) oraz słowo kluczowe <b>void</b>. Typ <b>Class</b> nie posiada konstruktora, natomiast JVM tworzy automatycznie obiekty tej klasy, reprezentujące typy danych ładowanych do programu za pomocą metody <i>defineClass</i> programu ładującego klasy.</p>
<pre>int ile = 0;  double arg1 = 0;</pre>	<pre>System::<b>Int32</b> ile = 0; lub System::<b>Int64</b> ile = 0; System::<b>Double</b> arg1 = 0;</pre>	<pre><b>Integer</b> ile = 0;  <b>Double</b> arg2 = 0.0;</pre>
<pre>void oblicz (int a) { }</pre>	<pre>System::<b>Void</b> oblicz (System::<b>Int32</b> a) { } lub System::<b>Void</b> oblicz (System::<b>Int64</b> a) { }</pre>	

# 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 <code>Class&lt;?&gt;</code>
<code>bool</code> (1 bajt)	<code>System.Boolean</code>	<code>boolean</code> (1 bajt)	<code>Boolean</code>	<code>Class&lt;Boolean&gt;</code>
<code>unsigned char</code> (1 bajt)	<code>System.Byte</code>			
<code>wchar_t</code> (2 bajty)	<code>System.Char</code>	<code>char</code> (2 bajty)	<code>Character</code>	<code>Class&lt;Character&gt;</code>
<code>signed char</code> (1 bajt)	<code>System.SByte</code>	<code>byte</code> (1 bajt)	<code>Byte</code>	<code>Class&lt;Character&gt;</code>
<code>short, signed short</code> (2 bajty)	<code>System.Int16</code>	<code>short</code> (2 bajty)	<code>Short</code>	<code>Class&lt;Short&gt;</code>
<code>unsigned short</code> (2 bajty)	<code>System.UInt16</code>			

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

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

## 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 aplet i jako aplikacja.

## 2.1. Tekst źródłowy w Javie - przykład

```
public class Witaj
{
    public static void main(String args[])
    {
        System.out.print("Dzien dobry, nazywam się Jan Kowalski\n");
    }
}
```

Diagram illustrating the structure of the Java source code:

- ciało klasy** (class body) is indicated by a large dashed bracket on the left, encompassing the entire code block.
- ciało metody** (method body) is indicated by a smaller dashed bracket on the right, encompassing the `main` method's implementation.

## 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

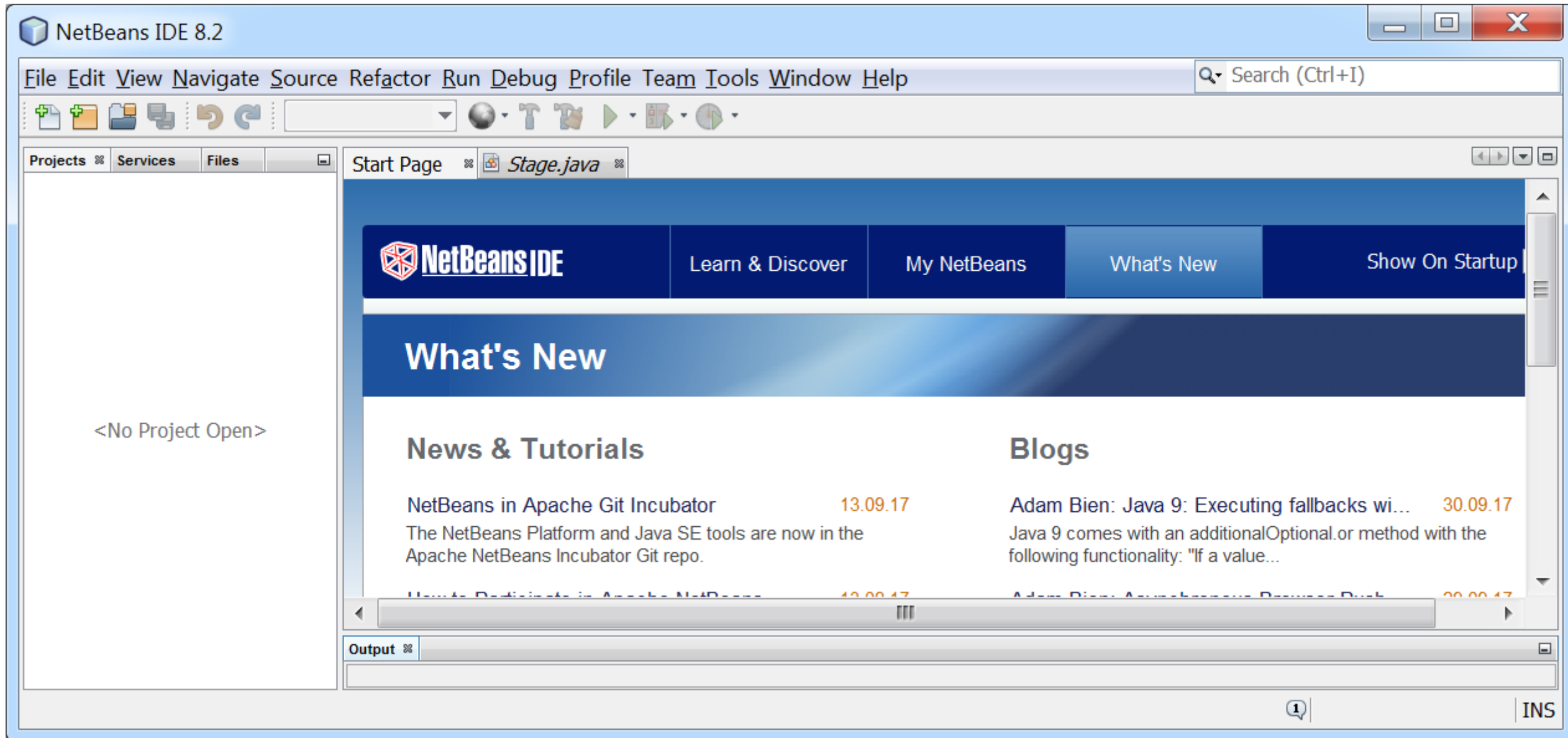
**Dzien dobry, nazywam się Jan Kowalski**

i przejdzie do następnego wiersza

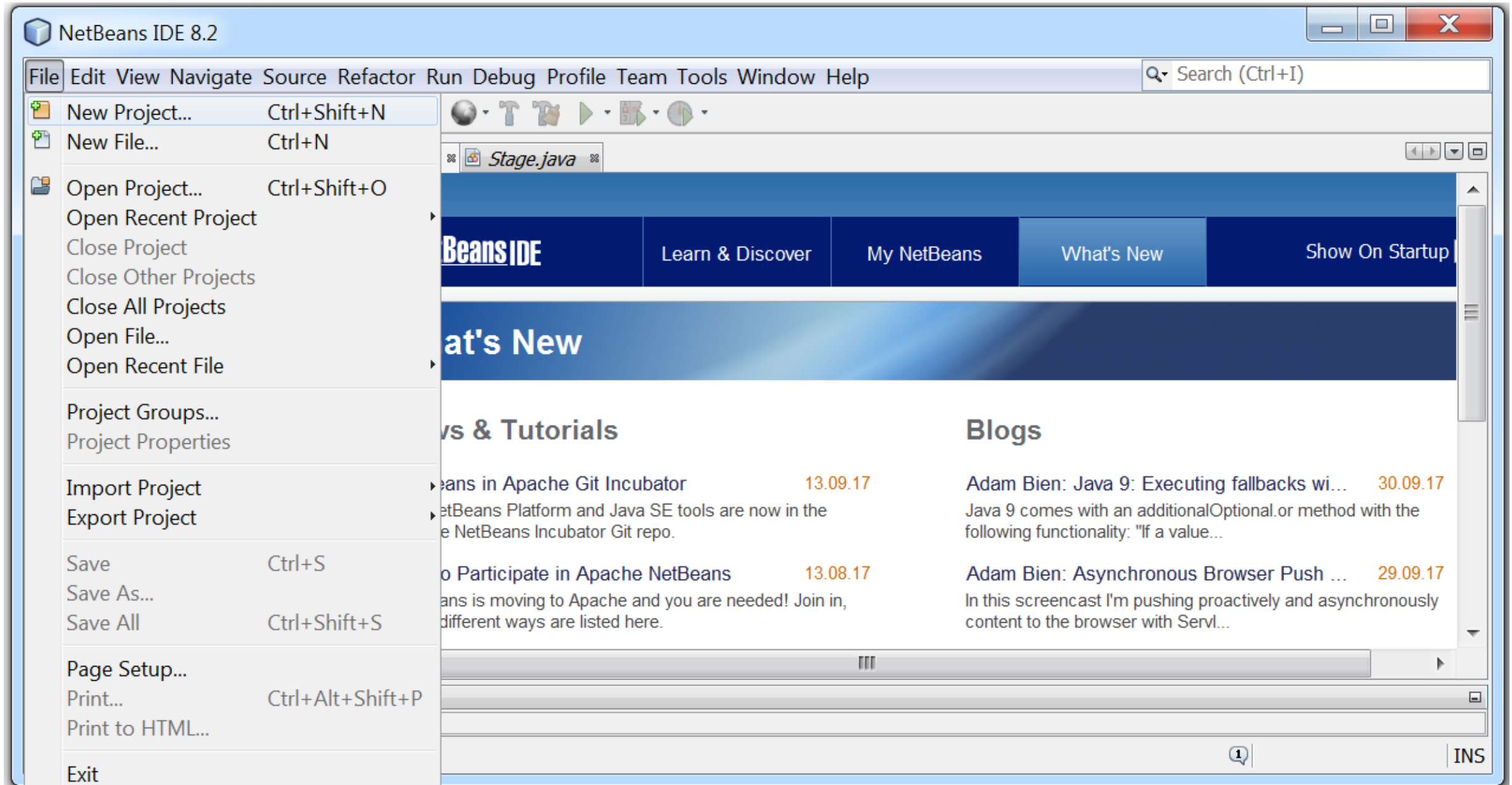
### Uwagi:

- 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**

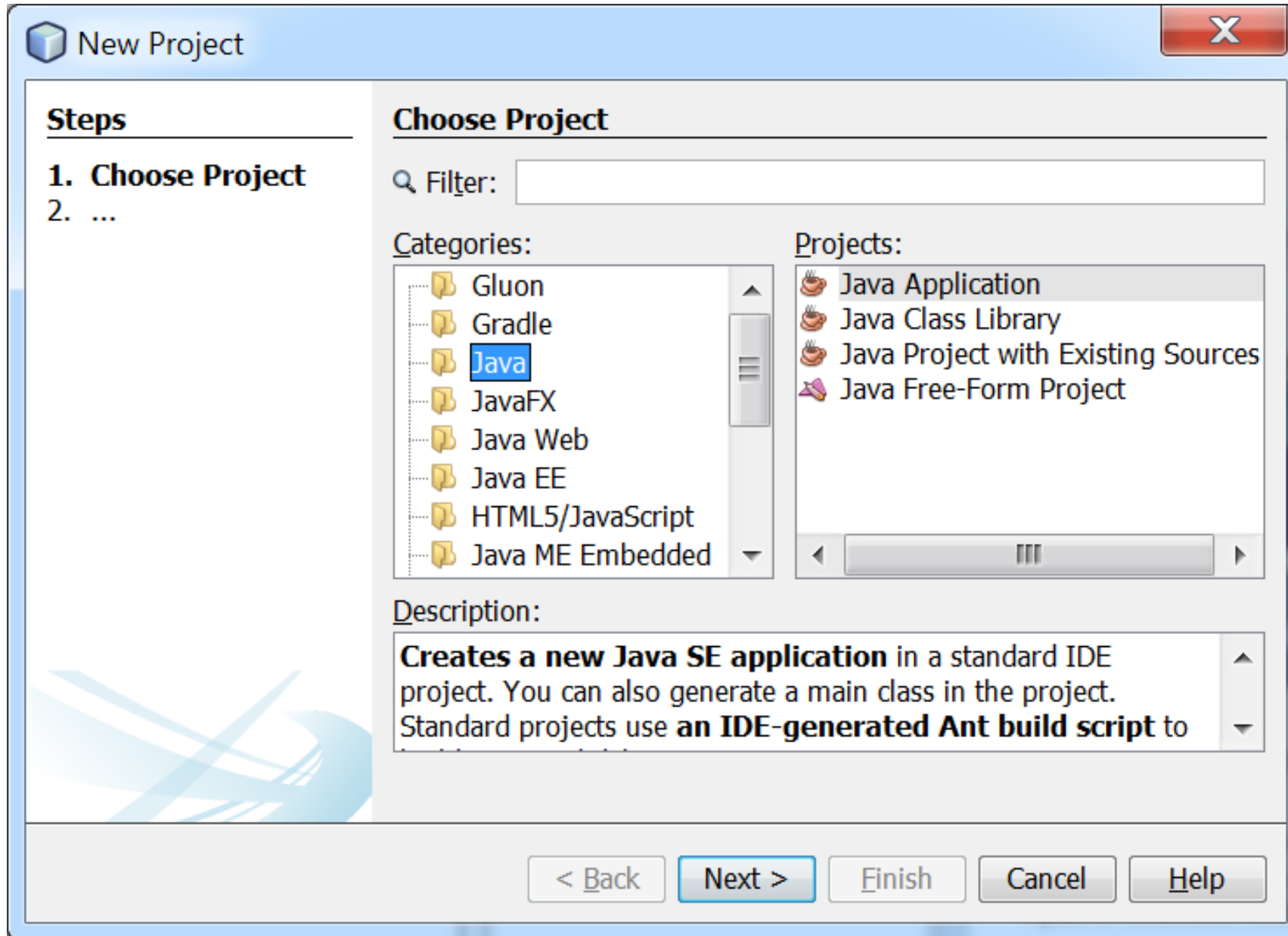
# Przykład 1: Tworzenie aplikacji w środowisku NetBeans



# Zakładanie nowego projektu



# Wybór kategorii projektu **Java** typu **Java Application**



# Domyślny formularz projektu po naciśnięciu klawisza **Next**.

Podanie własnych ustawień projektu: nazwy projektu (**Project Name**), katalogu (**Project Location**) oraz nazwy klasy głównej zawierającej funkcję main (**Create Main Class**).

Należy zatwierdzić klawiszem **Finish** te wprowadzone dane.

**Steps**

1. Choose Project
2. **Name and Location**

**Name and Location**

Project Name:

Project Location:

Project Folder:

Use Dedicated Folder for Storing Libraries

Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

Create Main Class

< Back   Next >   **Finish**   Cancel   Help

# Formularz edycji programu źródłowego

The screenshot shows the NetBeans IDE 8.2 interface. The title bar reads "Witaj - NetBeans IDE 8.2". The menu bar includes File, Edit, View, Navigate, Source, Refactor, Run, Debug, Profile, Team, Tools, Window, and Help. A search bar is located on the right of the menu bar with the text "Search (Ctrl+I)".

The toolbar contains various icons for file operations (new, open, save, print), navigation (back, forward), and execution (run, debug). Below the toolbar, the "Projects" tab shows a tree view of the project "Witaj", including "Source Packages", "witaj", "Witaj.java", and "Libraries".

The "Navigator" tab shows the "Members" of the "Witaj" class, listing "main(String[] args)".

The main editor window shows the source code for "Witaj.java". The code is as follows:

```
1  /*
2  * To change this license header, choose License Headers in Project Properties.
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6  package witaj;
7
8  /**
9   *
10  * @author PWR
11  */
12  public class Witaj {
13
14      /**
15       * @param args the command line arguments
16       */
17      public static void main(String[] args) {
18          // TODO code application logic here
19      }
20
21  }
```

The "Output" tab at the bottom is currently empty. The status bar at the bottom right shows "1:1" and "INS".

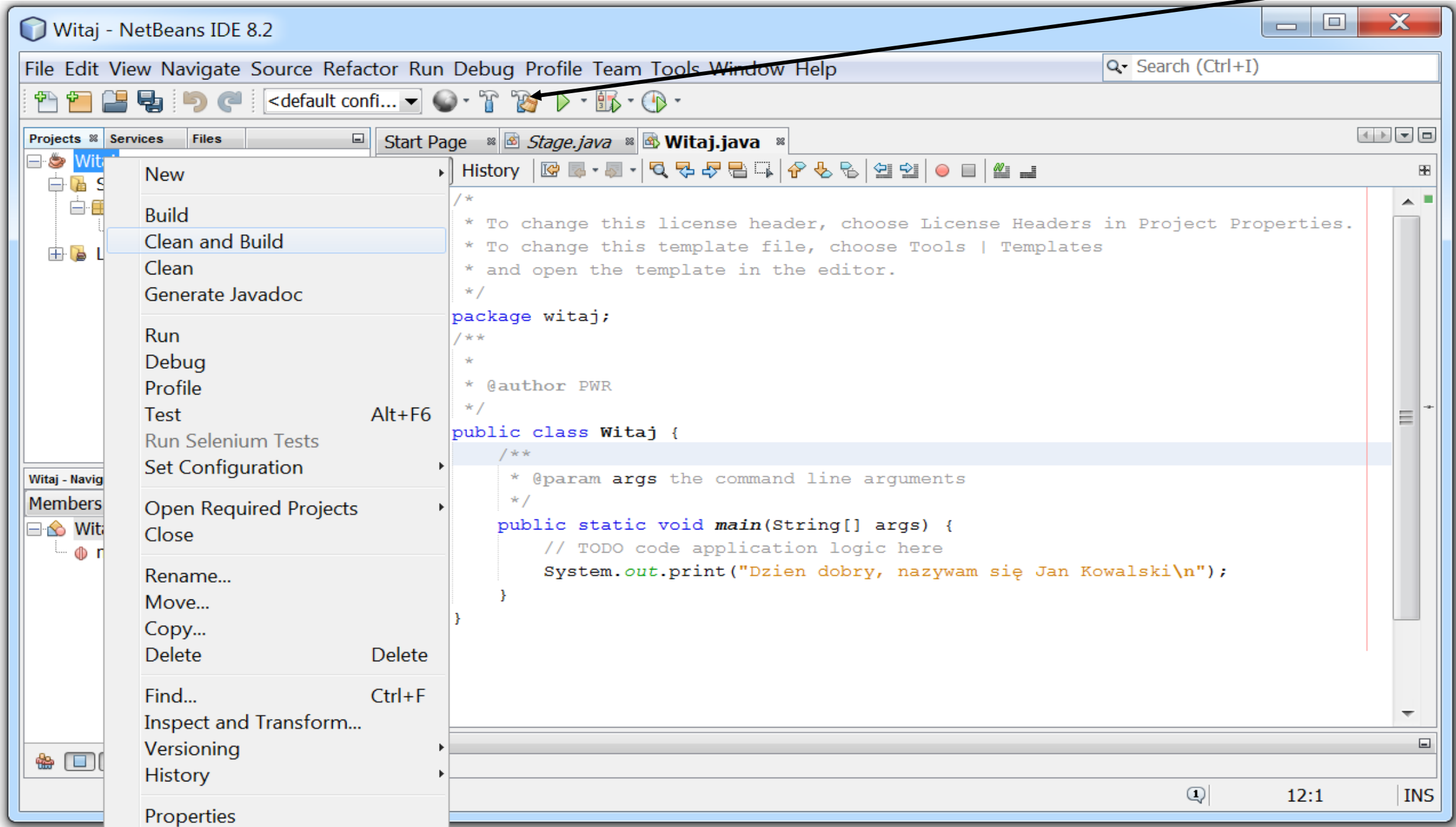
# Wpisanie elementarnej instrukcji w funkcji `main` programu konsolowego

The screenshot displays the NetBeans IDE 8.2 interface. The main editor window shows the source code for `Witaj.java`. The code includes a package declaration, a class declaration, and a `main` method. The `main` method contains a single line of code that prints a greeting message to the console.

```
1  /*
2  * To change this license header, choose License Headers in Project Properties.
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6  package witaj;
7  /**
8   *
9   * @author PWR
10 */
11 public class Witaj {
12     /**
13      * @param args the command line arguments
14      */
15     public static void main(String[] args) {
16         // TODO code application logic here
17         System.out.print("Dzien dobry, nazywam się Jan Kowalski\n");
18     }
19 }
20
```

The interface also shows the Project Explorer on the left, which displays the project structure with the `Witaj.java` file highlighted. The Navigator window below it shows the `main(String[] args)` method selected. The Output window at the bottom is currently empty.

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



The screenshot displays the NetBeans IDE 8.2 interface. The main window shows the 'Witaj.java' file open in the editor. A context menu is open over the file, with the 'Clean and Build' option highlighted. The menu items include: New, Build, Clean and Build, Clean, Generate Javadoc, Run, Debug, Profile, Test (Alt+F6), Run Selenium Tests, Set Configuration, Open Required Projects, Close, Rename..., Move..., Copy..., Delete (Delete), Find... (Ctrl+F), Inspect and Transform..., Versioning, History, and Properties. The editor window contains the following Java code:

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package witaj;
/**
 *
 * @author PWR
 */
public class Witaj {
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        System.out.print("Dzien dobry, nazywam się Jan Kowalski\n");
    }
}
```

The status bar at the bottom right shows the time 12:1 and the keyboard indicator INS.



# Komunikaty z przebiegu kompilacji w oknie Output

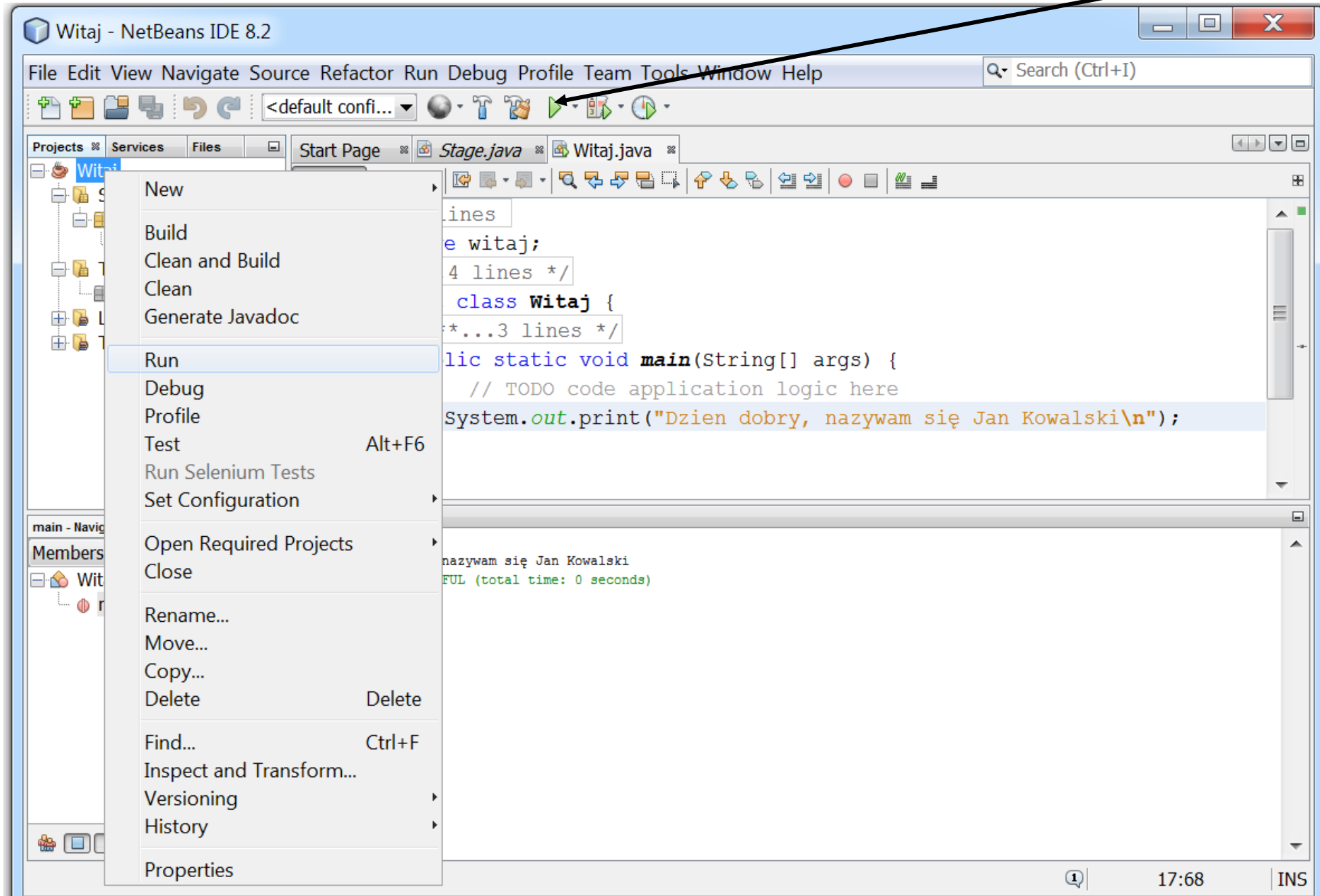
The screenshot displays the NetBeans IDE 8.2 interface. The main editor window shows the source code for `Witaj.java`. The code includes package declarations, comments, and a `main` method that prints a greeting message. The `main` method is highlighted in blue. The Output window at the bottom shows the compilation process, including directory creation, property file updates, and the final successful build message.

```
1  ...5 lines
6  package witaj;
7  /**...4 lines */
11 public class Witaj {
12     /**...3 lines */
15     public static void main(String[] args) {
16         // TODO code application logic here
17         System.out.print("Dzien dobry, nazywam się Jan Kowalski\n");
18     }
19 }
```

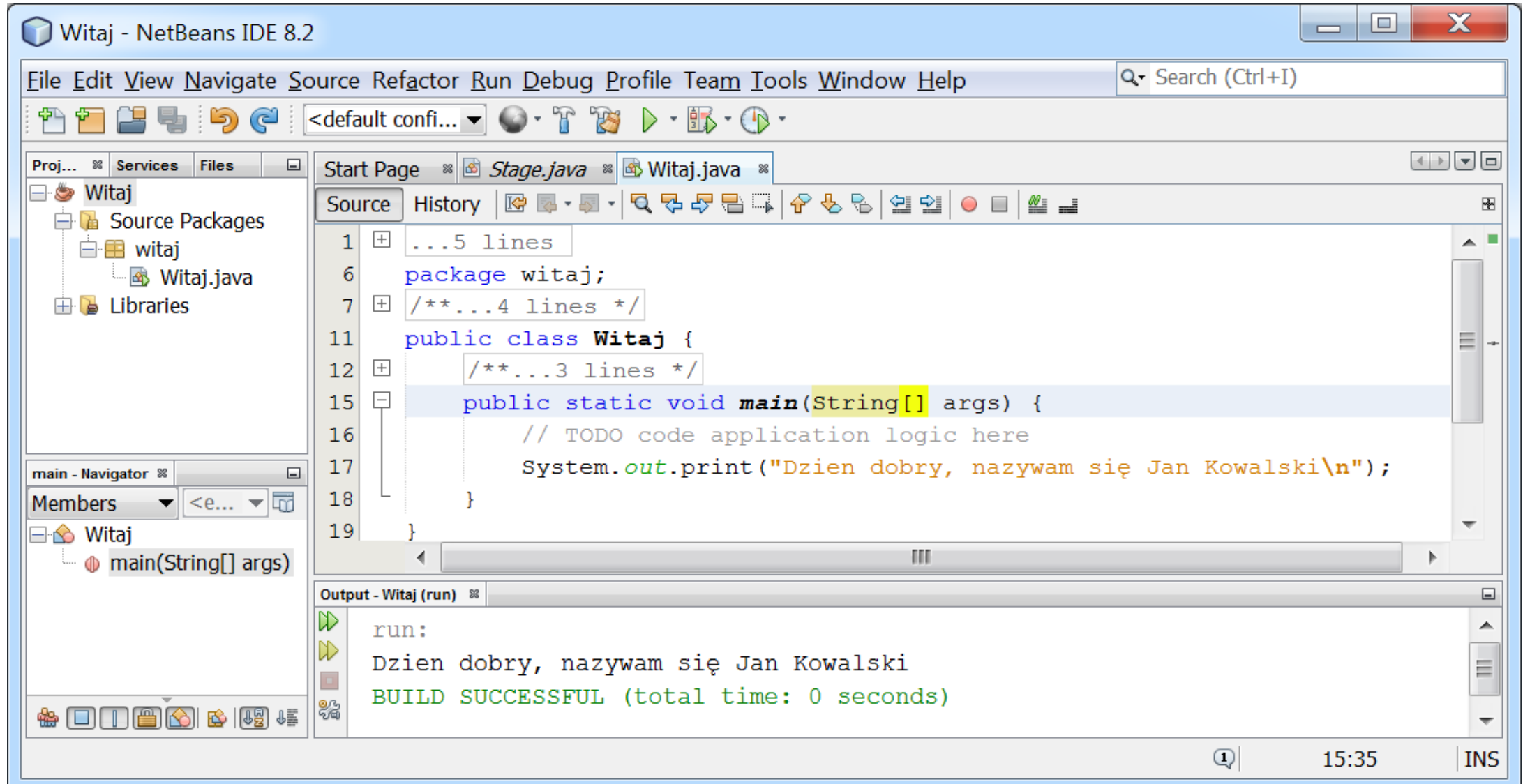
Output - Witaj (clean.jar) »

```
init:
deps-jar:
Created dir: C:\Studia\Szkola\CalaJava\Wyklady\Przyklady_wyklady\Witaj\build
Updating property file: C:\Studia\Szkola\CalaJava\Wyklady\Przyklady_wyklady\Witaj\build\build-jar.properties
Created dir: C:\Studia\Szkola\CalaJava\Wyklady\Przyklady_wyklady\Witaj\build\classes
Created dir: C:\Studia\Szkola\CalaJava\Wyklady\Przyklady_wyklady\Witaj\build\empty
Created dir: C:\Studia\Szkola\CalaJava\Wyklady\Przyklady_wyklady\Witaj\build\generated-sources\ap-source-output
Compiling 1 source file to C:\Studia\Szkola\CalaJava\Wyklady\Przyklady_wyklady\Witaj\build\classes
compile:
Created dir: C:\Studia\Szkola\CalaJava\Wyklady\Przyklady_wyklady\Witaj\dist
Copying 1 file to C:\Studia\Szkola\CalaJava\Wyklady\Przyklady_wyklady\Witaj\build
Nothing to copy.
Building jar: C:\Studia\Szkola\CalaJava\Wyklady\Przyklady_wyklady\Witaj\dist\Witaj.jar
To run this application from the command line without Ant, try:
java -jar "C:\Studia\Szkola\CalaJava\Wyklady\Przyklady_wyklady\Witaj\dist\Witaj.jar"
jar:
BUILD SUCCESSFUL (total time: 1 second)
```

# 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

The screenshot displays the NetBeans IDE 8.2 interface. The main editor window shows the source code for `Witaj.java` in the `witaj` package. The code includes a `main` method that prints a greeting message to the console.

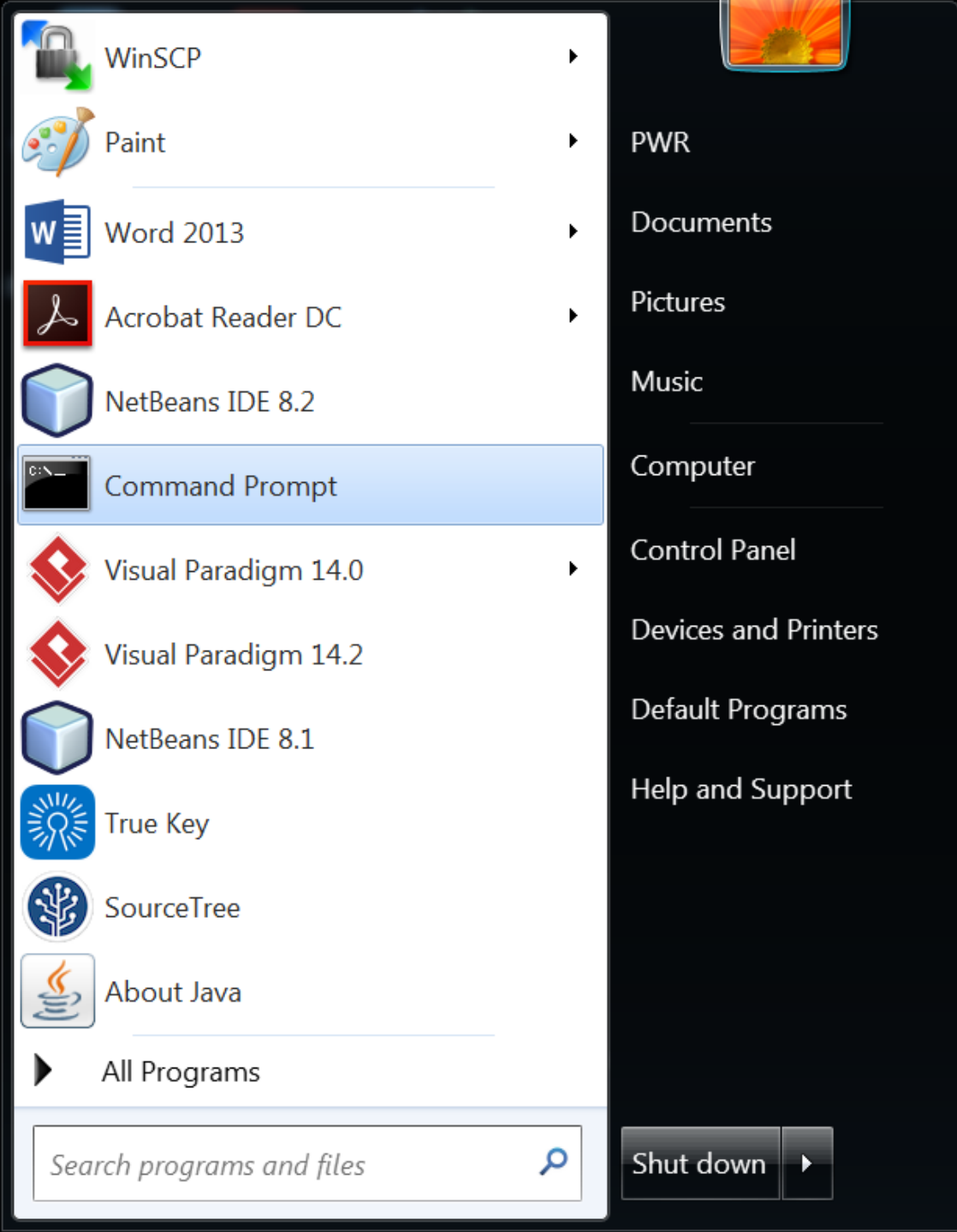
```
1 ...5 lines
6 package witaj;
7 /**...4 lines */
11 public class Witaj {
12     /**...3 lines */
15     public static void main(String[] args) {
16         // TODO code application logic here
17         System.out.print("Dzien dobry, nazywam się Jan Kowalski\n");
18     }
19 }
```

The `Output - Witaj (clean.jar)` window shows the successful compilation and execution of the program. The output includes the following text:

```
Created dir: C:\Studia\Szkola\CalaJava\Wyklady\Przyklady_wyklady\Witaj\build\generat
Compiling 1 source file to C:\Studia\Szkola\CalaJava\Wyklady\Przyklady_wyklady\Witaj
compile:
Created dir: C:\Studia\Szkola\CalaJava\Wyklady\Przyklady_wyklady\Witaj\dist
Copying 1 file to C:\Studia\Szkola\CalaJava\Wyklady\Przyklady_wyklady\Witaj\build
Nothing to copy.
Building jar: C:\Studia\Szkola\CalaJava\Wyklady\Przyklady_wyklady\Witaj\dist\Witaj.j
To run this application from the command line without Ant, try:
java -jar "C:\Studia\Szkola\CalaJava\Wyklady\Przyklady_wyklady\Witaj\dist\Witaj.jar"
jar:
BUILD SUCCESSFUL (total time: 0 seconds)
```

The `main - Navigator` window shows the `main(String[] args)` method in the `Witaj` class.

The status bar at the bottom right of the IDE shows the time as 17:68 and the cursor position as INS.

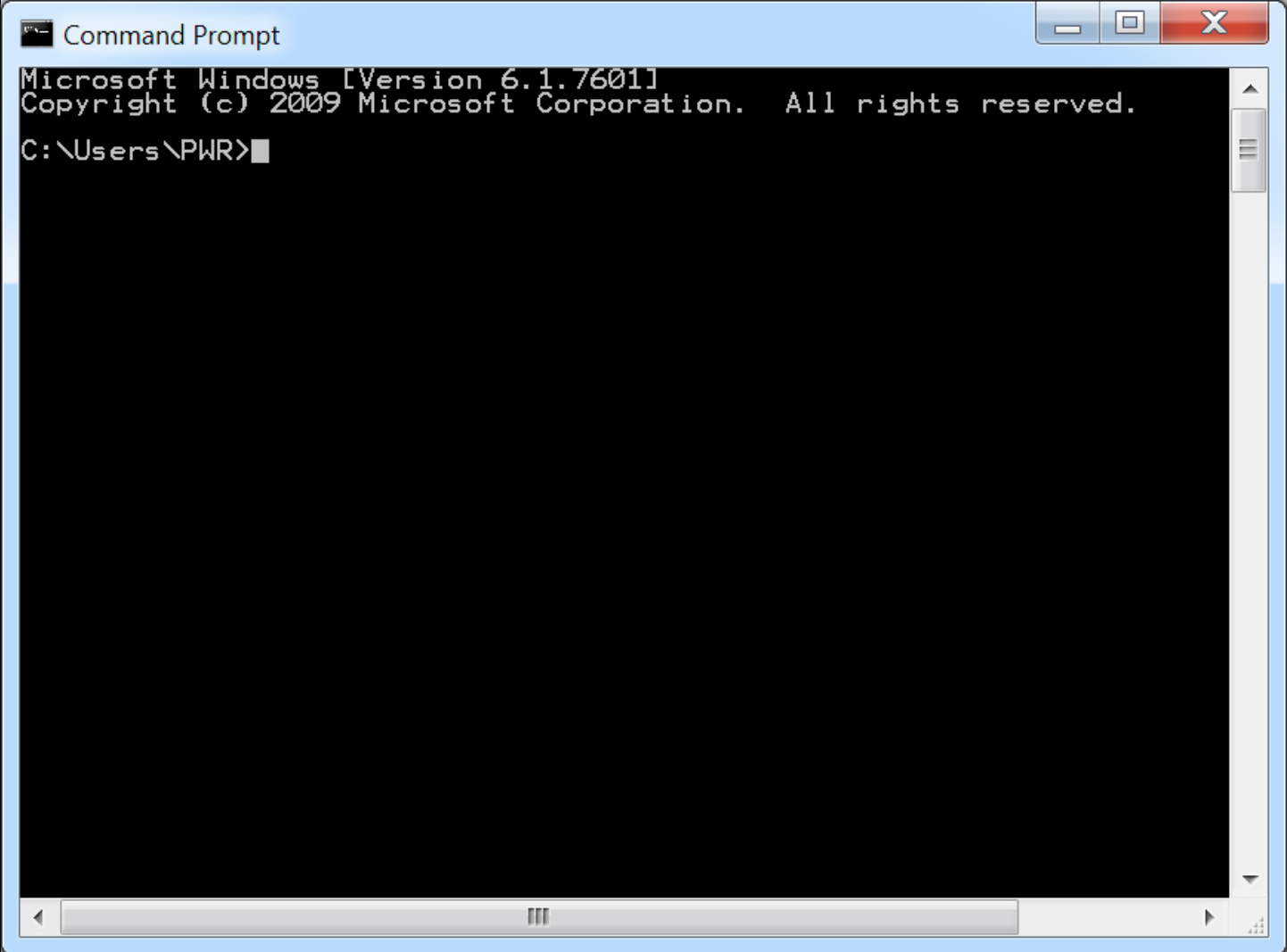


The image shows the Windows Start menu with a search bar at the top. The search bar contains the text "Search programs and files" and a magnifying glass icon. Below the search bar, there is a list of applications and system settings, each with an icon and a right-pointing arrow. The applications listed are WinSCP, Paint, Word 2013, Acrobat Reader DC, NetBeans IDE 8.2, Visual Paradigm 14.0, Visual Paradigm 14.2, NetBeans IDE 8.1, True Key, SourceTree, and About Java. The system settings listed are PWR, Documents, Pictures, Music, Computer, Control Panel, Devices and Printers, Default Programs, and Help and Support. At the bottom of the Start menu, there is a "Shut down" button with a right-pointing arrow.

- 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

Shut down

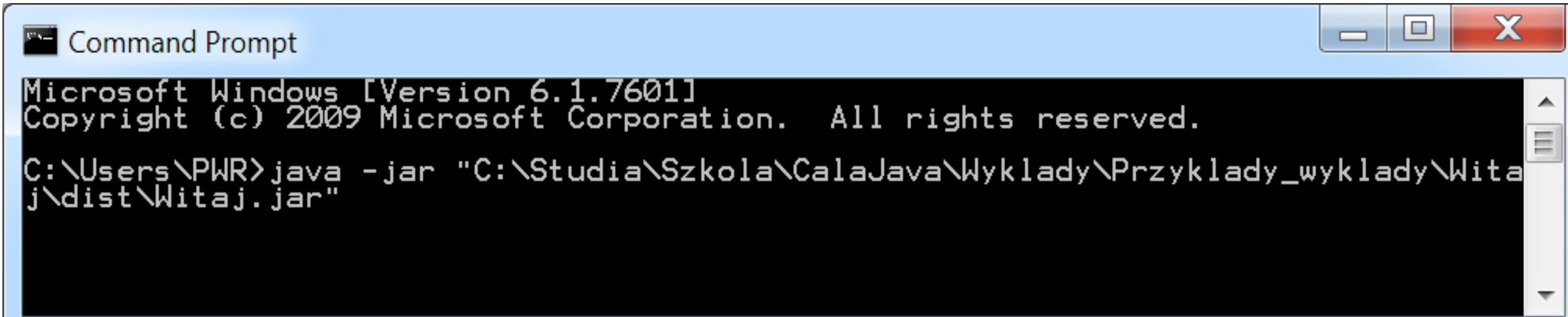


The image shows a Command Prompt window titled "Command Prompt". The window contains the following text:

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

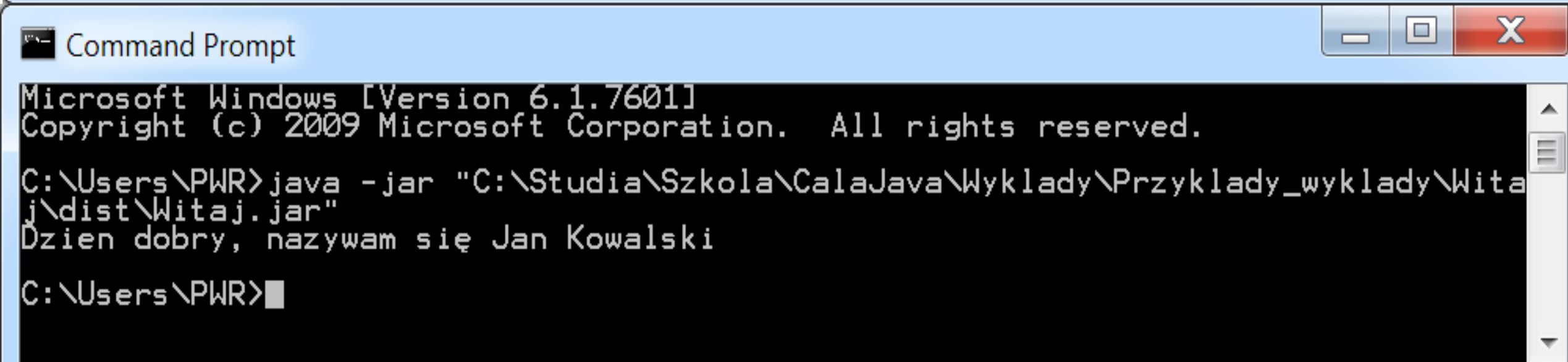
C:\Users\PWR>
```

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



```
Command Prompt
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\PWR>java -jar "C:\Studia\Szkola\CalaJava\Wyklady\Przyklady_wyklady\Witaj\dist\Witaj.jar"
```



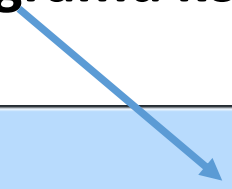
```
Command Prompt
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\PWR>java -jar "C:\Studia\Szkola\CalaJava\Wyklady\Przyklady_wyklady\Witaj\dist\Witaj.jar"
Dzien dobry, nazywam się Jan Kowalski

C:\Users\PWR>
```

**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

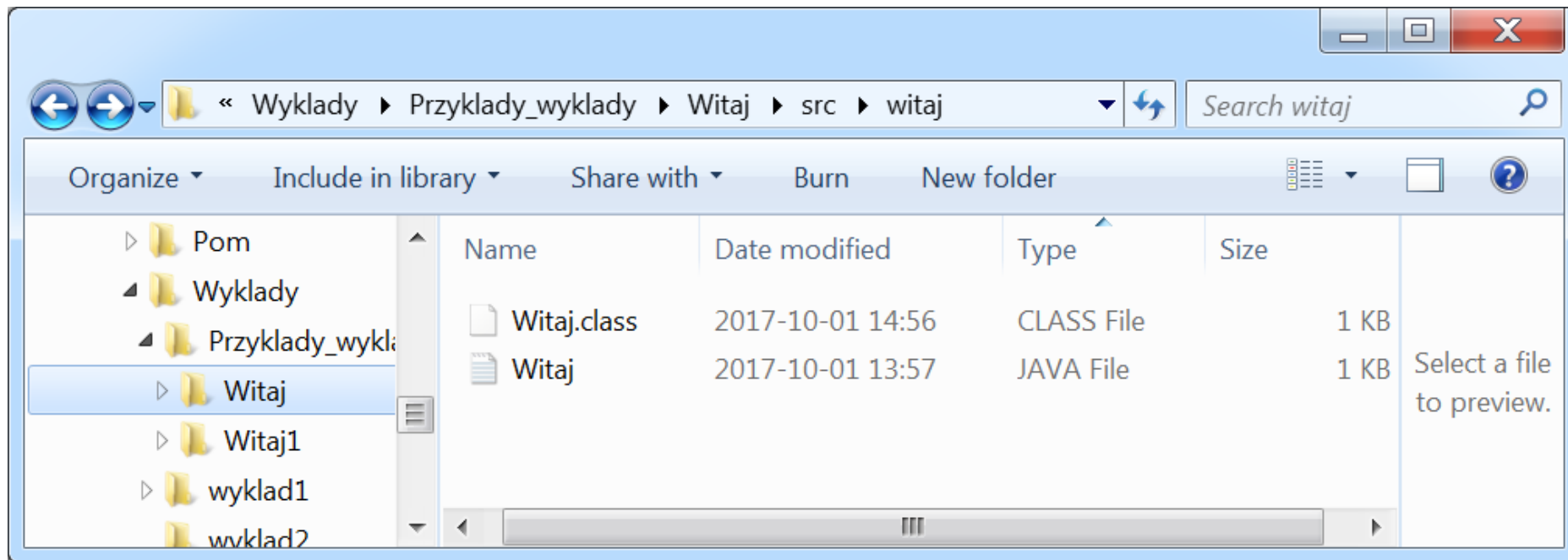


```
Command Prompt
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

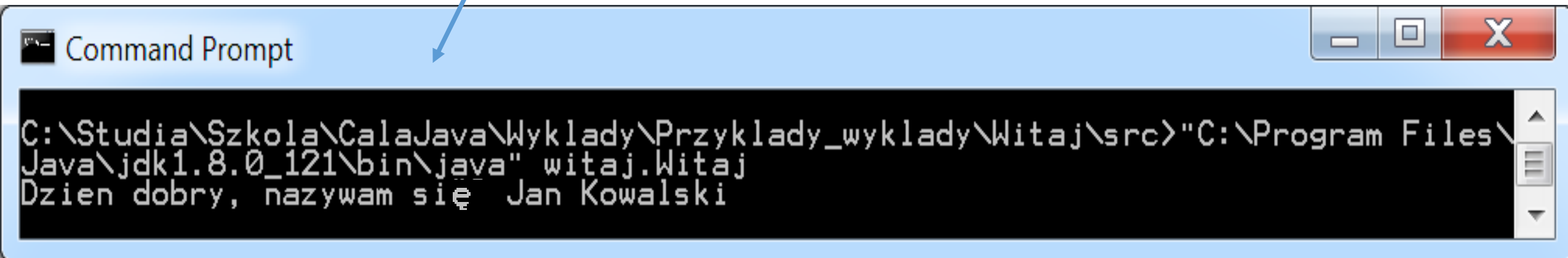
C:\Users\PWR>cd C:\Studia\Szkola\CalaJava\Wyklady\Przyklady_wyklady\Witaj\src

C:\Studia\Szkola\CalaJava\Wyklady\Przyklady_wyklady\Witaj\src>"C:\Program Files\Java\jdk1.8.0_121\bin\javac" witaj\Witaj.java

C:\Studia\Szkola\CalaJava\Wyklady\Przyklady_wyklady\Witaj\src>
```



## 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]);
    }
}
```



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

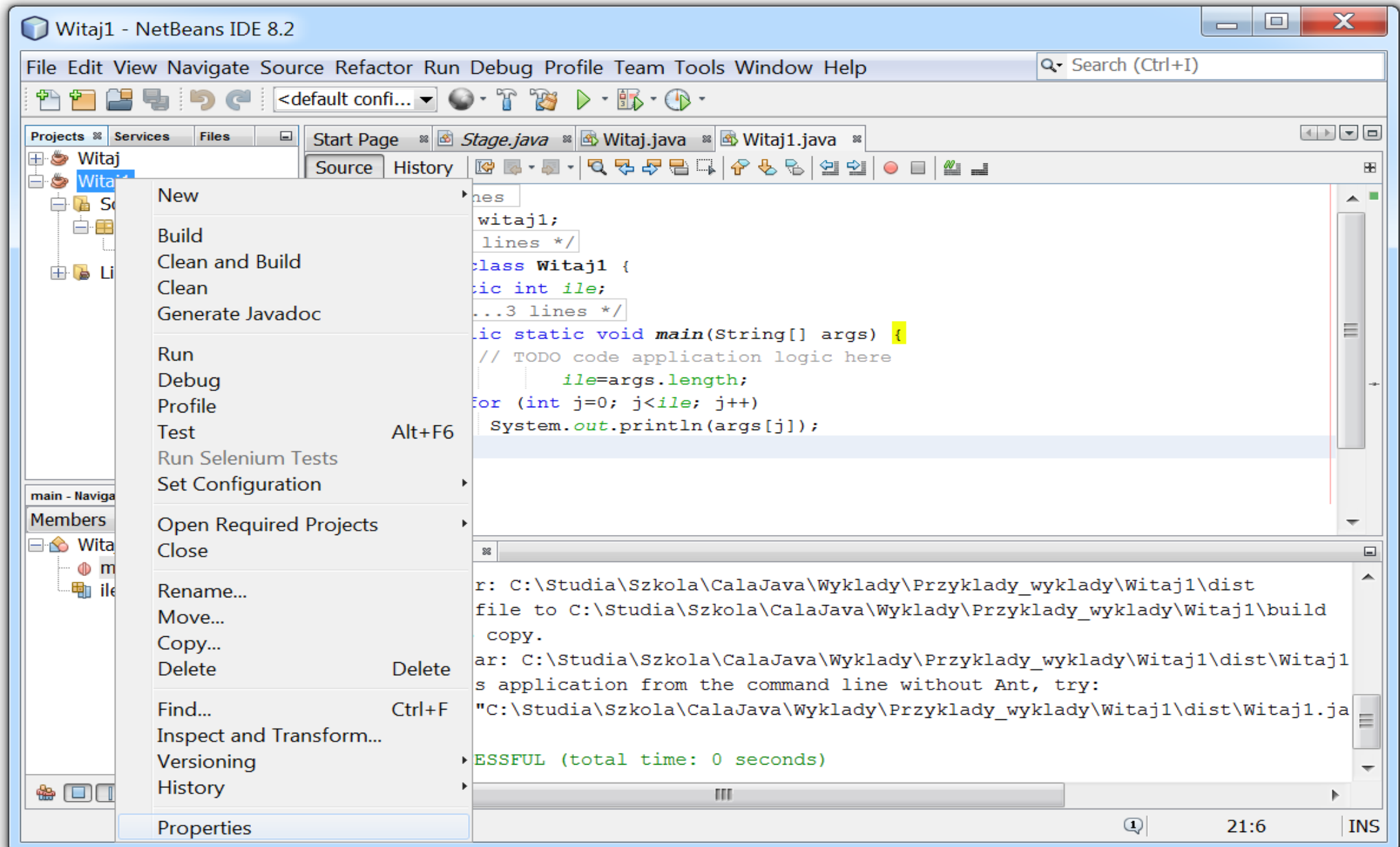
Members

- Witaj1
  - main(String[] args)
  - ile : int

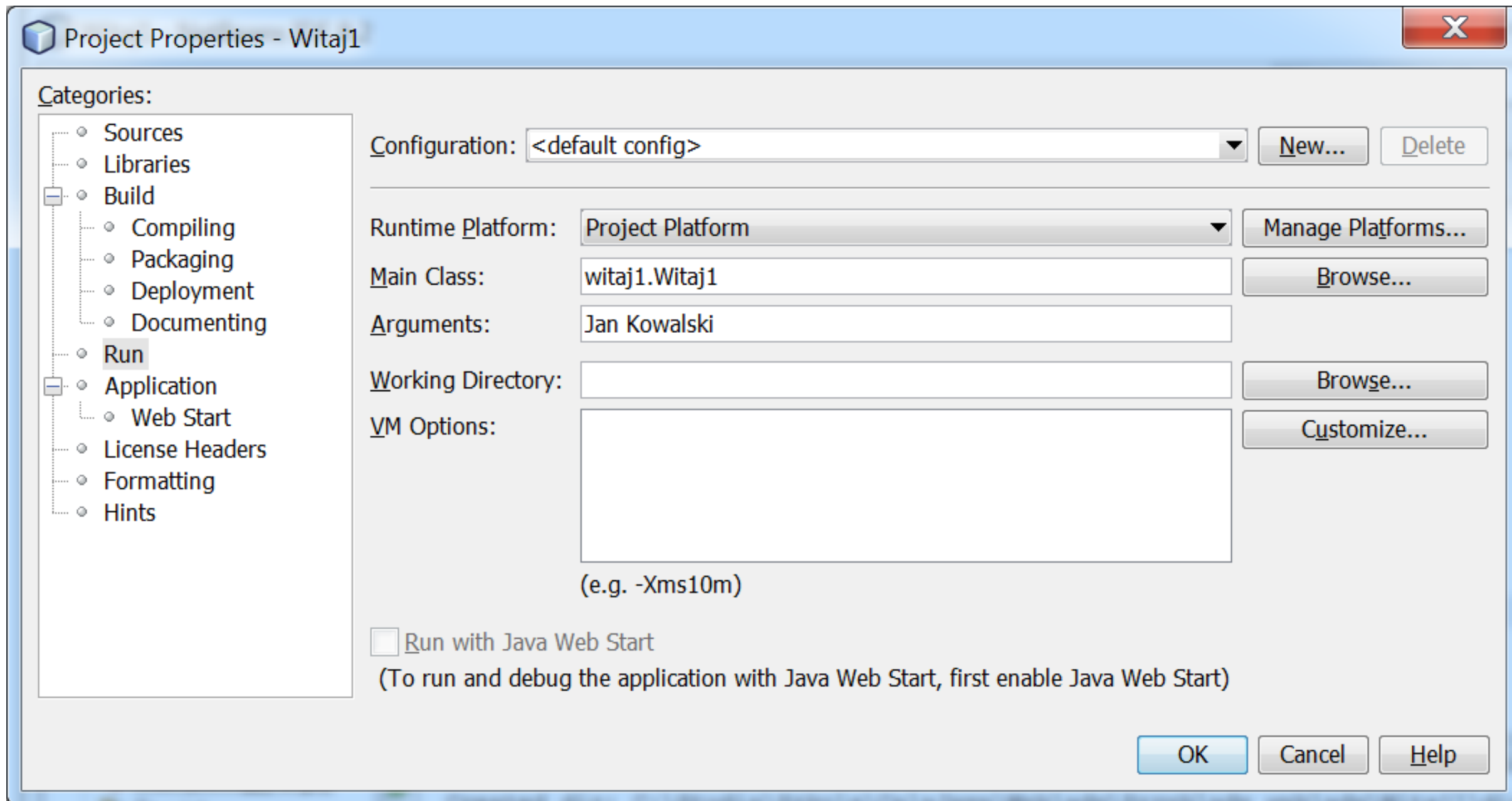
Output - Witaj1 (clean.jar)

```
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.jar"
BUILD SUCCESSFUL (total time: 0 seconds)
```

# 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

The screenshot shows the NetBeans IDE 8.2 interface. The title bar reads "Witaj1 - NetBeans IDE 8.2". The menu bar includes "File", "Edit", "View", "Navigate", "Source", "Refactor", "Run", "Debug", "Profile", "Team", "Tools", "Window", and "Help". The toolbar contains various icons, with a red arrow pointing to the "Run" button (a green play icon). The "Projects" window on the left shows a project named "Witaj1" with a sub-project "Witaj1" containing a "Source Packages" folder, a "witaj1" package, and a "Witaj1.java" file. The "main - Navigator" window shows the "Members" of the "Witaj1" class, including "main(String[] args)" and "ile : int". The main editor displays the source code for "Witaj1.java":

```
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 }
23
```

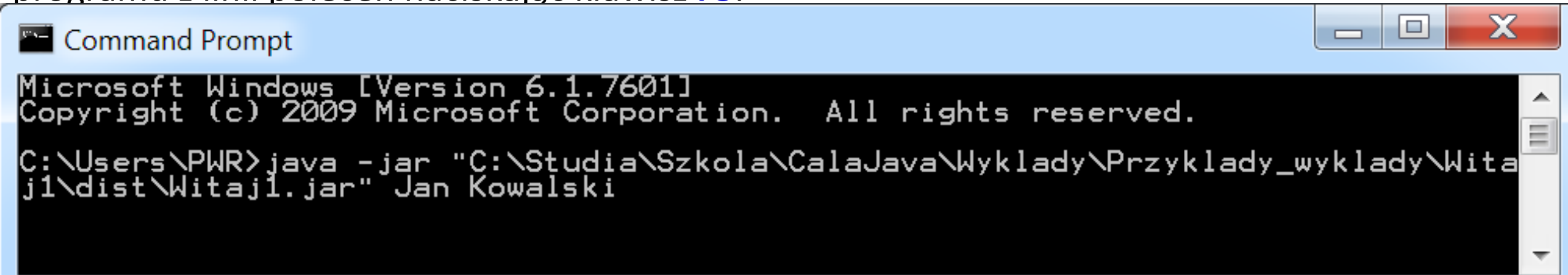
The "Output - Witaj1 (run)" window shows the following output:

```
run:
Jan
Kowalski
BUILD SUCCESSFUL (total time: 0 seconds)
```

The status bar at the bottom right shows "21:6" and "INS".

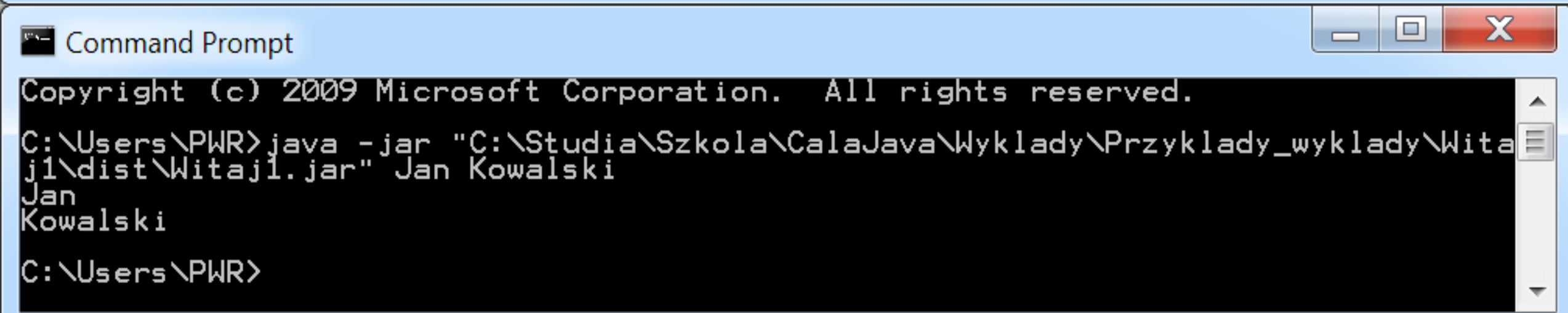
## b) Uruchomienie programu z linii poleceń

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**.



```
Command Prompt
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\PWR>java -jar "C:\Studia\Szkola\CalaJava\Wyklady\Przyklady_wyklady\Witaj1\dist\Witaj1.jar" Jan Kowalski
```



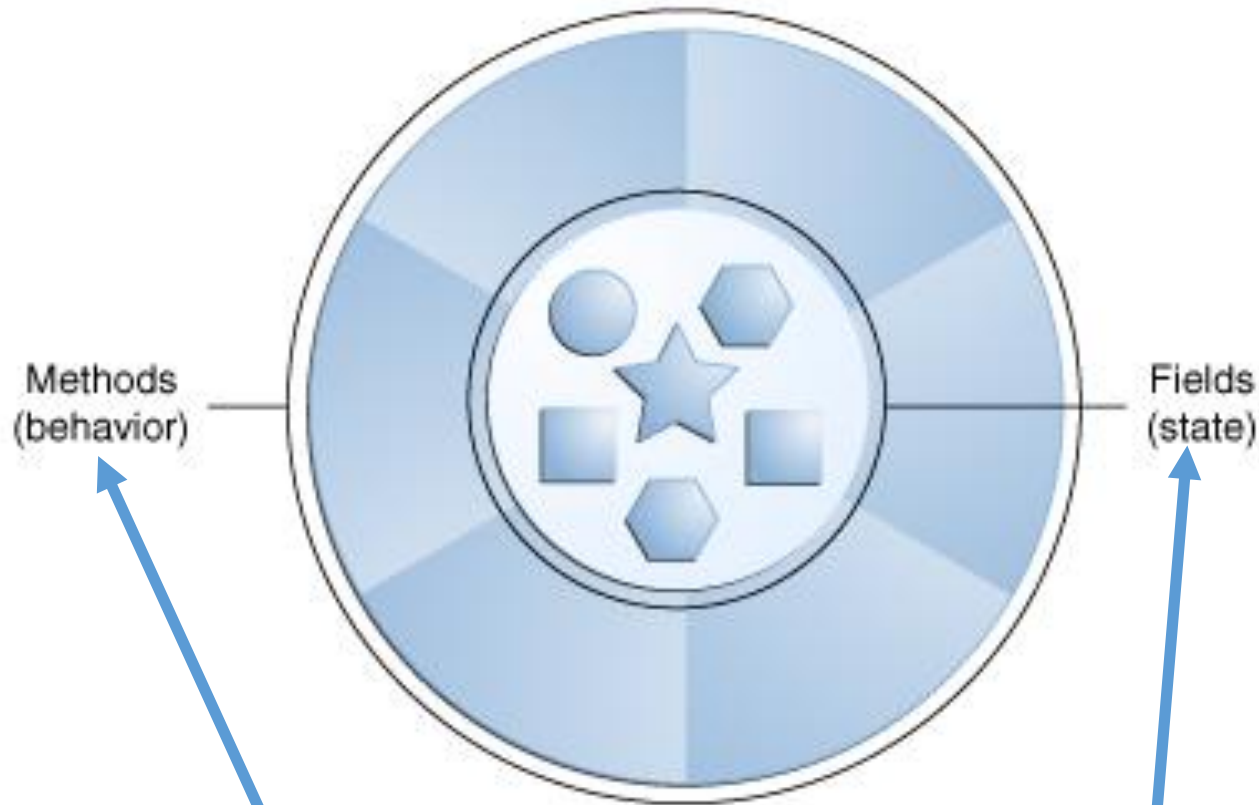
```
Command Prompt
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\PWR>java -jar "C:\Studia\Szkola\CalaJava\Wyklady\Przyklady_wyklady\Witaj1\dist\Witaj1.jar" Jan Kowalski
Jan
Kowalski

C:\Users\PWR>
```

# 3. Podstawy programowania obiektowego

# Obiekt



## Paradygmaty programowania obiektowego:

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

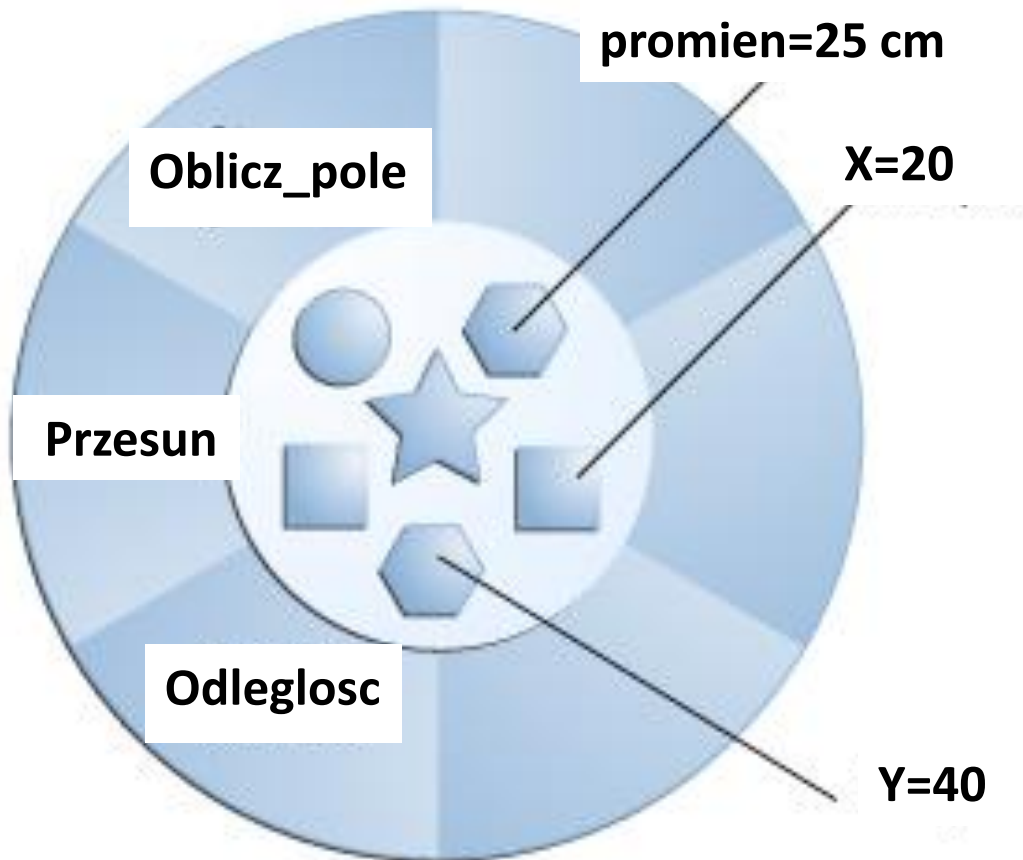
**Obiekt:**

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



# Obiekt (cd)

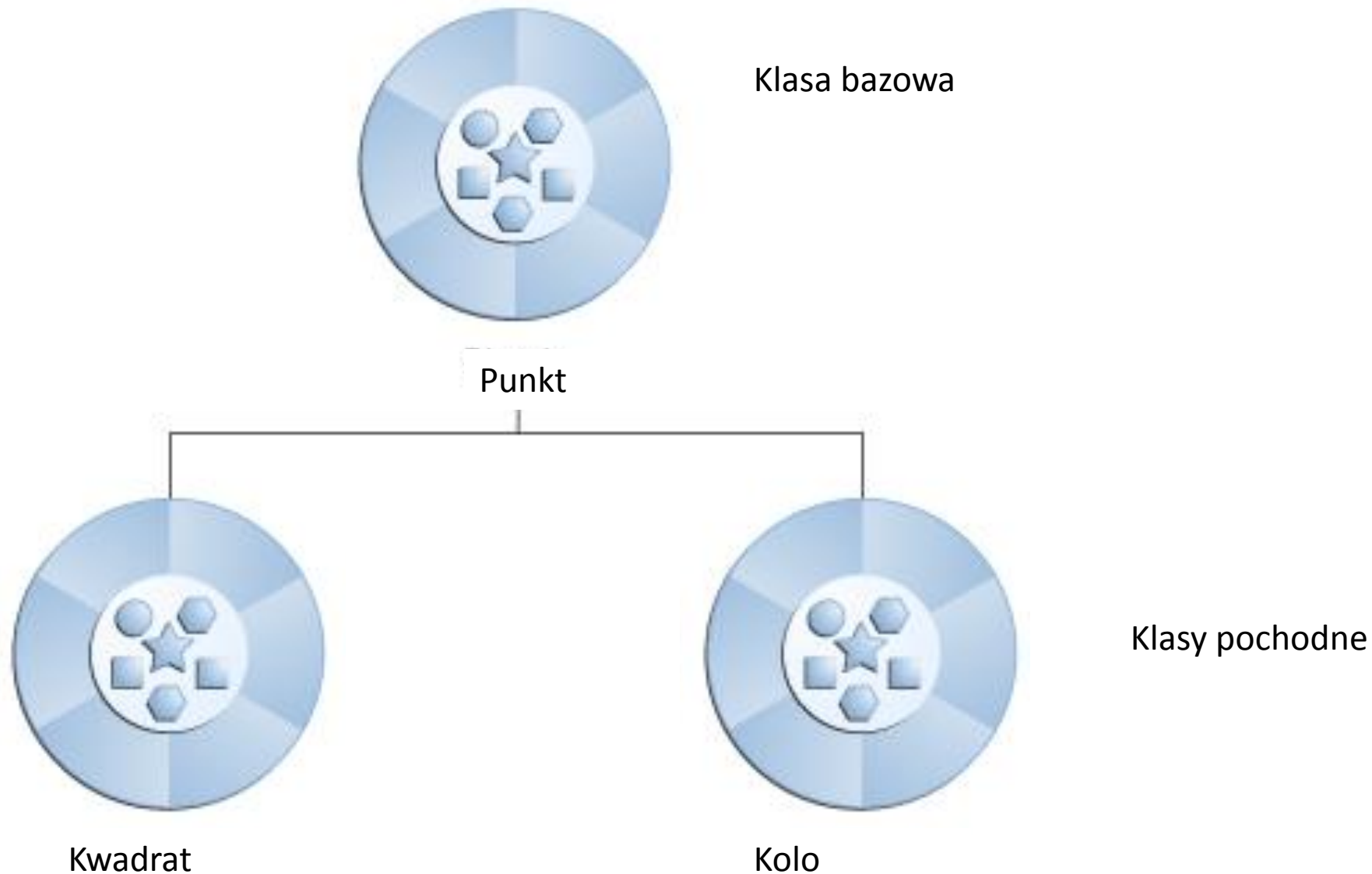
## Przykład obiektu typu Koło



Zalety (przy zachowaniu zasad poprawnego programowania):

- **Modularność** (kod obiektu może być przenoszony i uruchamiany niezależnie od położenia kodu źródłowego)
- **Hermetyzacja** kodu obiektu
- **Wieloużywalność** 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**