Programowanie obiektowe

Wstęp do programowania w języku Java

Paweł Rogaliński
Instytut Informatyki, Automatyki i Robotyki
Politechniki Wrocławskiej

pawel.rogalinski@pwr.wroc.pl

Zamiast wprowadzenia

Dla kogo jest dzisiejszy wykład?

- Dla osób, które znają język C/C++ ?
- Dla osób które zaczynają swoją przygodę z językiem Java?
- Dla osób zainteresowanych tworzeniem aplikacji dla systemu Android ?

Czy można nauczyć (nauczyć się) programowania w języku Java w kilka godzin?

Literatura

- Krzysztof Barteczko: *„Java od Podstaw do technologii”*, Tom 1 i 2, Wydaw. MIKOM, 2004.
- Elliot Koffman, Paul Wolfgang: „*Structury danych i techniki obiektowe na przykładzie Javy 5.0*”, Wydaw. HELION, 2006.
- Herbert Schildt: *„Java. Kompendium programisty”*, Wydaw. Helion, 2005.
- Joshua Bloch: „*Java. Efektywne programowanie*”, Wydaw. Helion, 2009.
- W. Frank Ableson, Robi Sen, Chris King: „Android w akcji”, Wyd II, Wydaw. Helion, 2011.

Narzędzia programistyczne

Firma ORACLE udostępnia bezpłatnie wszystkie niezbędne narzędzia do programowania w Javie. Pod adresem

`www.oracle.com/technetwork/java/javase/downloads/index.html`

można pobrać pakiet **Java SE Development Kit**, który zawiera:

- narzędzia do budowania, kompilacji i uruchamiania programu,
- narzędzia do dokumentowania i archiwizacji programów,
- pakiety klas standardowych,
- przykładowe aplety i aplikacje.

Uzupełnieniem pakietu **Java SE JDK** jest znormalizowana dokumentacja:

Java SE 6 Documentation

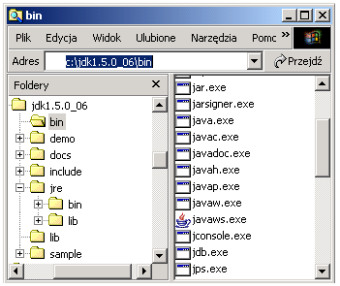
oraz zastaw bezpłatnych podręczników:

The Java™ Tutorials

Narzędzia programistyczne c.d.

Po zainstalowaniu pakietu **Java SE JDK** w katalogu ***jdk1.6.0_nn*** powstają podkatalogi:

- ***bin*** zawierający programy usługowe:
  - ***javac*** – kompilator,
  - ***java*** - interpreter,
  - ***appletviewer*** – przeglądarka apletów,
  - ***javadoc*** - generator dokumentacji,
  - ***jdb*** – debuger,
  - ***jar*** – narzędzie do tworzenia archiwów.



- ***jre*** zawierający środowisko uruchomieniowe:
  - wirtualną maszynę javy,
  - bibliotekę klas,
- ***lib*** zawierający dodatkowe biblioteki klas
- ***demo*** zawierający aplety i aplikacje demonstracyjne
- ***include*** zawierający pliki nagłówkowe języka C
- ***docs*** zawierający znormalizowana dokumentacja pakietu w postaci stron HTML.

Zintegrowane środowiska programistyczne Javy

Pakiet **Java SE JDK** nie zawiera żadnego edytora kodu źródłowego. Dlatego do pisania programów w języku Java należy używać innych narzędzi. Najpopularniejsze zintegrowane środowiska programistyczne Javy to:

NetBeans – ***http://www.netbeans.org/***

Eclipse – ***http://www.eclipse.org/***

JCreator – ***http://www.jcreator.com/***

Zintegrowane środowiska programistyczne Javy

Do programowania w systemie Android jest zalecane zainstalowanie:

- Pakiet **Java JDK 5** lub **Java JDK 6**,
  (Uwaga: środowisko uruchomieniowe Java JRE nie jest wystarczające)

- Środowisko **Eclipse** w jednej z poniższych wersji:
  - Eclipse IDE for Java Developers
  - Eclipse Classic
  - Eclipse IDE for Java EE Developers

Narzędzia programistyczne dla systemu Android

Do tworzenia aplikacji dla systemu Android niezbędna jest zainstalowanie pakietu **Android SDK** udostępnianego bezpłatnie pod adresem:

`http://developer.android.com/sdk/index.html`

Szczegółowa instrukcja instalacji „krok po kroku” jest pod adresem:

`http://developer.android.com/sdk/installing.html`

Tutoriale:

1. „The Java SE Tutorials” (on line version) - praktyczny przewodnik dla programistów tworzących aplikacje w języku Java

   http://docs.oracle.com/javase/tutorial/

2. Mark Dexter: „Using the Eclipse Workbench” (video tutorial)

   http://eclipsetutorial.sourceforge.net/workbench.html

3. Mark Dexter: „Eclipse and Java for Total Beginners” (video tutorial)

   http://eclipsetutorial.sourceforge.net/totalbeginner.html

4. Mark Dexter: „Eclipse and Java: Introducing Persistence” (video tutorial)

   http://eclipsetutorial.sourceforge.net/persistence.html

5. Mark Dexter: „Eclipse and Java: Using the Debugger” (video tutorial)

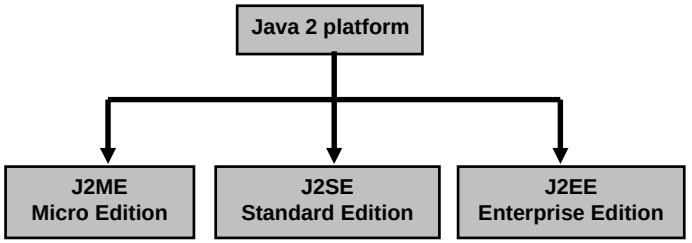
   http://eclipsetutorial.sourceforge.net/debugger.html

Java jako uniwersalne środowisko programowania

Platforma programistyczna Java 2 zawiera:

- uniwersalny obiektowy język programowania,
- kompilator, debuger, narzędzia tworzenia dokumentacji, itp.,
- bogaty zestaw standardowych bibliotek umożliwiających między innymi:
  - jednolity sposób tworzenia graficznych interfejsów użytkownika (AWT, Swing),
  - dostęp do baz danych (JDBC API),
  - działania w sieci (aplikacje typu klient-serwer, aplety, serwelety),
  - programowanie multimedialne (Java 3D, Java Media Framework),
  - przetwarzanie dokumentów HTML oraz XML.
- środowisko programowania w systemach rozproszonych i heterogenicznych (JNI - Java Native Interface, RMI - Remote Metod Invocation),
- środowisko budowania programu z gotowych komponentów (JavaBeans)

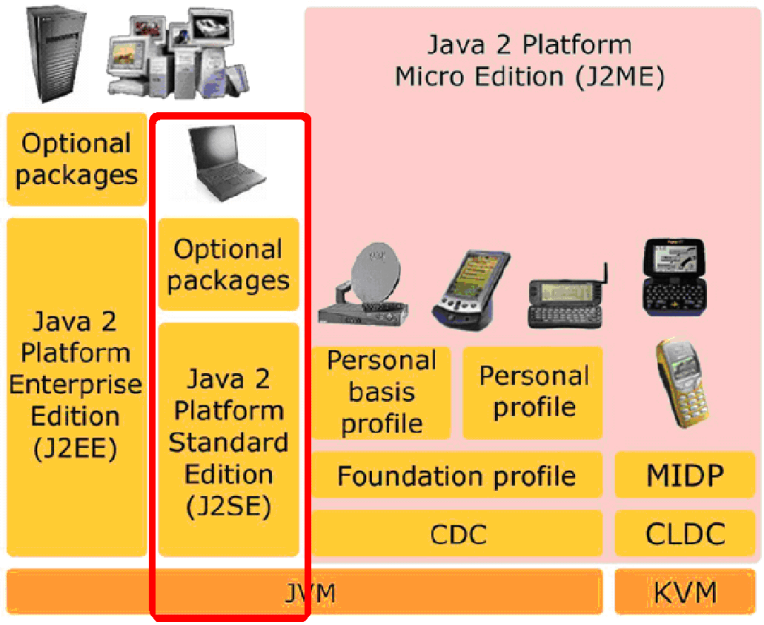
Aktualne edycje Javy



Java 2 Micro Edition (J2ME) - uproszczona wersja platformy do programowania urządzeń elektronicznych o bardzo ograniczonych zasobach, takich jak telefony komórkowe lub palmtopy.

Java 2 Standard Edition (J2SE) – platforma przeznaczona głównie do standardowych zastosowań dla komputerów personalnych i serwerów.

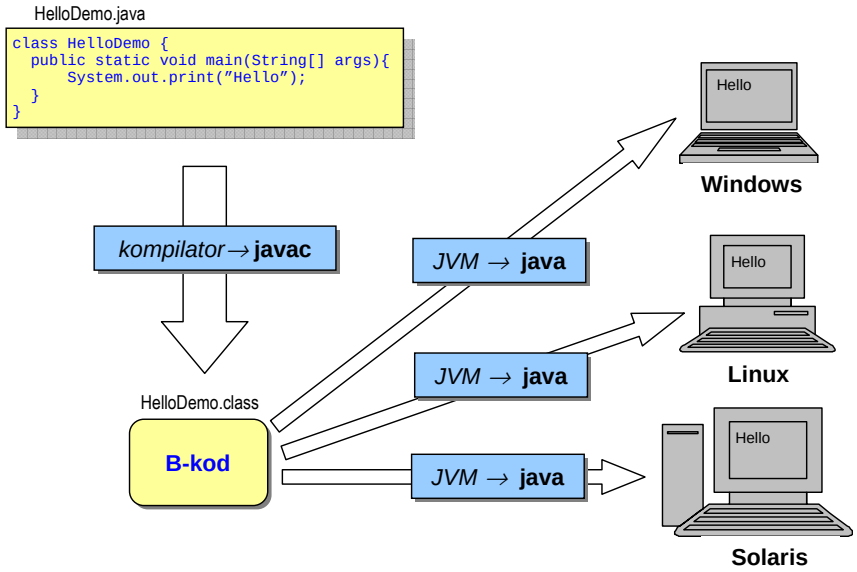
Java 2 Enterprise Edition (J2EE) – platforma przeznaczona głównie do tworzenia rozbudowanych i zaawansowanych aplikacji biznesowych opartych na architekturze wielowarstwowej.



Wirtualna maszyna Javy (JVM)

- **Java Virtual Machine** to rodzaj wirtualnego komputera, który ma swój zestaw rejestrów, zestaw instrukcji, stos i pamięć dla programów.
- Dzięki standaryzacji maszyny wirtualnej, programy napisane w Javie są **uniwersalne**, tzn. wykonują się identycznie w każdym systemie operacyjnym.
- Programy napisane w Javie są kompilowane do poziomu kodu pośredniego, nazywanego **kodem bajtowym Javy** (bytecode).
- Kod bajtowy jest **interpretowany** przez wirtualną maszynę JVM do postaci programu wykonywalnego dla danego systemu operacyjnego.

Wieloplatformowość programów w Javie

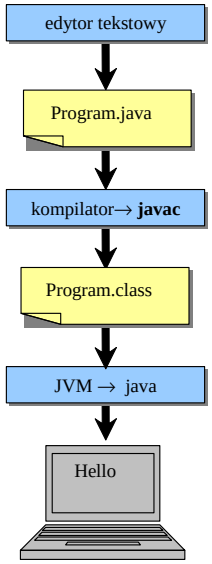


Program w Javie

- Każdy program w Javie jest zestawem klas.
- Klasa jest podstawową jednostką enkapsulacji (nie można pisać kodu poza definicją klasy).
- Pisany przez nas program może być zapamiętany w jednym lub wielu plikach źródłowych o rozszerzeniu "java".
- Należy przestrzegać następującej konwencji dotyczącej nazewnictwa – nazwa klasy powinna być zgodna z nazwą pliku, który przechowuje program.

Kompilacja

- Kompilator Javy wymaga, aby pliki źródłowe miały rozszerzenie „**`java`**”.
- Pliki źródłowe są kompilowane za pomocą kompilatora Javy (**`javac.exe`**) do postaci kodu bajtowego (pośredniego), a nie kodu maszynowego.
- Polecenie kompilacji pliku źródłowego ma postać:
  `javac nazwa_pliku.java`
- Wynikiem kompilacji są pliki z rozszerzeniem „**`class`**”, które mogą być wykonane przez maszynę wirtualną Javy.
- Podczas kompilacji pliku źródłowego każda klasa zostaje przeniesiona do swojego własnego pliku o nazwie właściwej zgodnej z nazwą klasy i rozszerzeniu „**`class`**”



Aplikacje i Aplety

- Wyróżniamy dwa rodzaje programów: aplikacje (standalone programs) i aplety (applets).
- Aplikacje mogą działać zarówno w trybie graficznym jak i tekstowym.
- Aplety działają jedynie w środowisku graficznym.
- Aby zobaczyć działanie aplikacji musimy mieć zainstalowaną w naszym komputerze wirtualną maszynę Javy – JVM.
- Aplety są wykonywane przez środowisko przeglądarek; są one widoczne wtedy, gdy przeglądarka posiada zintegrowaną wirtualną maszynę Javy.

Aplikacja

- Aby aplikacja mogła zostać uruchomiona, główna klasa musi zawierać metodę
  `public static void main(String args[])`
- Maszyna wirtualna Javy jest wywoływana za pomocą polecenia **`java`** z argumentami: nazwa pliku o rozszerzeniu „**`class`**” zawierającego metodę **`main()`** oraz argumenty wywołania tej metody, np.:
  `java nazwa_pliku arg1 arg2`
- Po załadowaniu klasy przez JVM sterowanie zostaje przekazane do metody **`main()`** i tu zaczyna się właściwe działanie programu: tworzenie obiektów, odwołania do innych klas aplikacji.

Aplet

- Aplety w odróżnieniu od aplikacji nie posiadają metody **`main()`**. Główna klasa każdego apletu (np. **`MyApps`**) musi być klasą publiczną, dziedziczyć z predefiniowanej klasy **`Applet`** z pakietu **`java.applet`**, albo **`JApplet`** z pakietu **`javax.swing`**.
- Nazwa pliku źródłowego w którym znajduje się główna klasa musi być taka jak nazwa klasy, a jego rozszerzenie musi być „**`java`**”.
- Do uruchomienia apletu trzeba utworzyć plik HTML zawierający znacznik wywołania tej klasy, np.:

```
<applet code = "MyApps.class"
 width = "300" height = "300">
</applet>
```

- Po napotkaniu tego znacznika przeglądarka ładuje plik **`MyApps.class`**, następnie wywoływany jest konstruktor tej klasy oraz metoda inicjalizacyjna, itd.

Pierwszy program

- Tworzymy plik *PierwszyProgram.java* i wprowadzamy następujący kod:

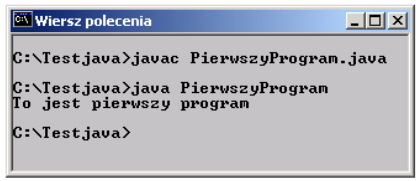
```
class PierwszyProgram
{
    public static void main(String arg[]){
        System.out.println("To jest pierwszy program");
    }
}
```

- Następnie kompilujemy program źródłowy za pomocą kompilatora *javac*:

  javac PierwszyProgram.java

- Na końcu wykonujemy program, korzystając z interpretera *java*:

  java PierwszyProgram



Program z argumentami

- Tworzymy plik *Argumenty.java* i wprowadzamy następujący kod:

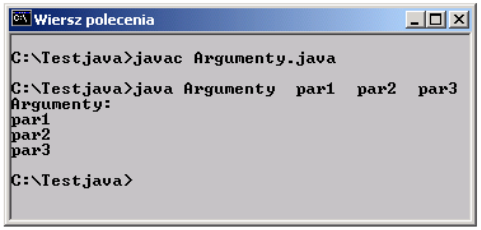
```
class Argumenty
 {
   public static void main(String[] args)
   {
      int n = args.length;
      System.out.println("Argumenty: ");
      for (int i=0; i<n; ++i)
        System.out.println(args[i]+ "\t");
   }
 }
```

- Następnie kompilujemy program źródłowy za pomocą kompilatora *javac*:

  javac Argumenty.java

- Na końcu wykonujemy program, korzystając z interpretera *java*:

  java Argumenty par1 par2 par3



Pierwszy aplet

- Tworzymy plik *PierwszyAplet.java* i wprowadzamy następujący kod:

```
import java.awt.*;
import java.applet.*;

public class PierwszyAplet extends Applet
{
   public void paint(Graphics g)
   {
      g.drawString("To jest pierwszy aplet ", 20, 20);
   }
}
```

- Następnie kompilujemy program źródłowy za pomocą kompilatora *javac*:

  javac PierwszyAplet.java

- Co się stanie, gdy aplet zostanie uruchomiony jako samodzielna aplikacja? Interpreter Javy wyświetli wtedy komunikat o błędzie, że nie znalazł metody *main*:

  Exception in thread "main" java.lang.NoSuchMethodError: main

Pierwszy aplet c.d.

- Musimy jeszcze przygotować prostą stronę WWW z odpowiednio zapisanym znacznikiem *<APPLET>*. Wymaganymi parametrami znacznika *<APPLET>* są:
  - *width* i *height*, które definiują rozmiar obszaru wykorzystywanego przez aplet na stronie WWW,
  - *code* określający nazwę pliku „*class*” z kodem bajtowym apletu.

```
<html>
<body>
<applet code = "PierwszyAplet.class"
        width = "300"
        height = "200">
</applet>
</body>
</html>
```

- Po otwarciu strony WWW przeglądarka utworzy okno o rozmiarach *width* x *height*, a następnie uruchomi i wyświetli w utworzonym oknie aplet, którego kod bajtowy znajduje się w pliku „*PierwszyAplet.class*”.

Aplet uruchamiany jako aplikacja

Istnieje możliwość napisania apletu, który będzie mógł pracować zarówno jako aplet, jak i aplikacja. Wystarczy dodać do niego metodę *main*, która utworzy okno (obiekt klasy *Frame*) i egzemplarz apletu.

```
import java.awt.*;
import java.applet.*;

public class Hello extends Applet
{
   public void paint(Graphics g)
   {
      g.drawString("Hello World", 20, 20);
   }

   public static void main(String args[])
   {
     Frame frame = new Frame("Pogram Hello");
     Hello aplet = new Hello();
     frame.setSize(300, 300);
     frame.add(aplet);
     frame.show();
     aplet.init();
   }
}
```


Okna dialogowe do wczytywania i wyświetlania tekstów

- Tekst (łańcuch znaków) można zapamiętać w obiektach klasy **`String`**

```
String text;      // deklaracja obiektu klasy String
```

- Tekst można wczytywać w graficznym oknie dialogowym tworzonym przez metodę **`showInputDialog`** z klasy **`JOptionPane`**

```
text = JOptionPane.showInputDialog("tekst zachęty:");
```

- Text można wyświetlić w oknie konsoli za pomocą metody **`print`** lub **`println`**

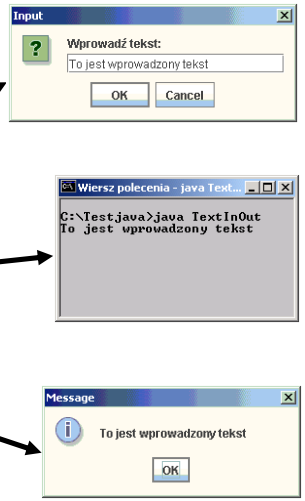
```
System.out.print(text);
```

- Tekst można wyświetlać w oknie graficznym tworzonym przez metodę **`showInputDialog`** z klasy **`JOptionPane`**

```
JOptionPane.showMessageDialog(text);
```


Okna dialogowe do wczytywania i wyświetlania tekstów

```
import javax.swing.JOptionPane;

class TextInOut
{
    public static void main(String arg[])
    {
        String text;

        text = JOptionPane.showInputDialog(null,
                                "Wprowadź tekst:");

        System.out.println(text);

        JOptionPane.showMessageDialog(null, text);

        System.exit(0);
    }
}
```



Wczytywanie i wyświetlanie tekstów - przykład

```
import javax.swing.JOptionPane;

class Wizytowka
{
    public static void main(String[] arg)
    {
        String nazwisko, imie;

        // wcztytanie danych
        nazwisko = JOptionPane.showInputDialog(null, "Podaj nazwisko:");
        imie = JOptionPane.showInputDialog(null, "Podaj imie:");

        // wyswietlanie w oknie konsoli
        System.out.println("Imie: " + imie);
        System.out.println("Nazwisko: " + nazwisko);

        // wyswietlanie w oknie graficznym
        String text;
        text  = "Imie: " + imie + "\n";
        text += "Nazwisko: " + nazwisko;
        JOptionPane.showMessageDialog(null, text);

        // zakonczenie aplikacji
        System.exit(0);
    }
}
```