Programowanie obiektowe

Java

Autor: dr inż. Zofia Kruczkiewicz

Literatura:

- **L. Lemay, R. Cadenhead** Java 2 dla każdego
- **P. Naughton** Podręcznik Języka Programowania Java
- **Krzysztof Barteczko** JAVA, wykłady i ćwiczenia
- **Boone Barry** Java dla programistów C i C++

Linki do oprogramowania Java SDK1.4.2

Java 2 Software Development Kit, Standard Edition, version 1.4.2 –

http://java.sun.com/j2se/1.4.2/download.html

Środowiska programistyczne np:

- Borland Jbuilder
- Symantec Visual Café
- Jcreator http://www.jcreator.com/download.htm
  (wersja free http://www.jcreator.com/download.htm)
- Help (http://www.allimant.org/javadoc/jdk14e.html)

1. Charakterystyka języka Java

1) **Obiektowy język Java** – składnia języka podobna do języka C++.

   Pliki źródłowe: nazwa_klasy_publicznej.java,

   gdzie nazwa musi być nazwą klasy publicznej, zdefiniowanej w tym pliku.

2) **Kompilator** przetwarza program nazwa_klasy_publicznej.java na kod binarny zwany B-kod (bytecode, J-code)

   Pliki po kompilacji: nazwa_klasy_publicznej.class

   B-kod może być zinterpretowana i wykonywana przez maszynę wirtualną Java (JVM Java Virtual Machine), czyli urządzenie logiczne

3) **Maszyna wirtualna Java** (JVM Java Virtual Machine).

   JVM jest abstrakcyjnym komputerem, który wykonuje programy nazwa.class:

   a. interpretator wbudowany w przeglądarkę WWW,

   b. oddzielny program

   c. Just-In-Time (przetworzenie nazwa.class na program wykonalny specyficzny dla danej maszyny)

4) **Biblioteka Javy** – pakiety z oprogramowaniem wspomagającym tworzenie programów działających w sieci np. Internet, umożliwiającym tworzenie interfejsu użytkownika, ogólnego przeznaczenia

2. Tworzenie programu w Javie

- Aplikacja (application) - program interpretujący aplikacje jest uruchamiany w systemie operacyjnym (java.exe)

  Program zawierający między innymi jeden moduł źródłowy, którego klasa publiczna zawiera publiczną metodę klasową o nagłówku

  public static void main(String args[])

- Aplet (applet) – program interpretujący aplety jest wbudowany np. w przeglądarkę www

  Program zawierający między innymi jeden moduł źródłowy, którego klasa publiczna zawiera miedzy 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

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

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:

- wszystkie stałe, zmienne i funkcje są elementami składowymi klas
- nie ma wielkości globalnych, definiowanych poza klasą
- nie deklaruje się metod rozwijalnych (**inline**) – o tym decyduje sam kompilator
- 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*:
  - ✓ zawiera statyczny obiekt składowy typu *PrintStream* o nazwie *out*
  - ✓ wywołanie *System.out.print* oznacza pisanie łańcucha typu *String* do standardowego strumienia wyjściowego, w tym wypadku ekranu
  - ✓ przeciążona metoda *print* generuje jeden wiersz wyjściowy i powraca do metody *main*

3. Java jako język obiektowy

3.1. definicja klasy, dziedziczenie, implementowanie metod interfejsów

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

Klasa:

- ✓ przed słowem class może wystąpić jeden ze specyfikatorów:

  public – klasa dostępna publicznie

  final - klasa ta nie może mieć następcy

  abstract – klasy te 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

- ✓ przed słowem **class** mogą wystąpić dwa specyfikatory:
  public abstract, **public final**

- ✓ 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 równoważne niejawne wystąpienie w tej definicji słów **extends** Object

- ✓ i**mplementowanie**- po nazwie klasy wystąpią słowa: i**mplements** 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 oznacza dziedziczenie wielobazowe w C++

Ciało klasy:

- ✓ zamknięte w nawiasy klamrowe
- ✓ może zawierać zmienne składowe (to jest pola lub zmienne wystąpienia)
- ✓ zmienne klasowe (statyczne, tj. poprzedzone słowem kluczowym **static**)
- ✓ konstruktory (metody o nazwie klasy bez zwracanego typu)
- ✓ metody klasowe (nagłówek poprzedzony słowem kluczowym **static**)
- ✓ 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 może wystąpić jeden ze specyfikatorów dostępu:

  private (dostęp tylko dla elementów klasy - **private int** d;),

  protected (dostęp tylko w podklasie, nawet jeśli podklasa należy do 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

3.2. Tworzenie obiektu

- 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 *Punkt p;* jedynie powiadamia kompilator, że będzie używana zmienna *p*, której typem jest *Punkt* (brak przydzielonego miejsca w pamięci na taki obiekt)

- Do zmiennej p można przypisać dowolny obiekt typu *Punkt* przydzielając mu pamięć za pomocą **new**

  np. Punkt p1 = **new** Punkt();

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

  Punkt p2 = **new** Punkt(1, 7);

  Argumentem operatora **new** jest definiowany przez programistę konstruktor *Punkt(**int** a, **int** b)*, który inicjuje obiekt utworzony przez operator **new**. Operator **new** zwraca referencję do tego obiektu, po czym przypisuje go do zmiennej p.

- Dostęp do elementów klasy uzyskuje się za pomocą operatora kropkowego

Przykłady

1) Drukowanie na ekranie w pętli wartości typu całkowitego bez tworzenia obiektu

```
public class Napis              //klasa publiczna niefinalna, nie abstrakcyjna
{
  static int ile;      //składowa klasowa (istnieje niezależnie od istnienia obiektu tej klasy)

  public static void main(String args[])
   {
    ile=10;                          //ile musi być składową typu static!
   for (int j=0; j<ile; j++)         //definicja zmiennej sterującej w bloku instrukcji for
    {
      System.out.println(j);         //konwersja zmiennej typu całkowitego na łańcuch
    }                                //i przejście do następnej linii
  }
}
```

---

2) Drukowanie na ekranie w pętli wartości typu całkowitego z tworzeniem obiektu

```
public class Napis_
{
  int ile;                                  //zmienna składowa klasy

  public static void main(String args[])
   { Napis_ p = new Napis_();               //wywołanie domyślnego konstruktora podczas
                                            //przydziału pamięci na obiekt klasy Napis_
     p.ile=10;                              //p jest referencją do obiektu klasy Napis_
     for (int j=0; j<p.ile; j++)            //odwołanie do obiektu p
      {System.out.println("petla "+j);}     //dodawanie łańcucha pętla do łańcucha
                                            //znaków (cyfry) uzyskanego za pomocą
                                            //konwersji z wartości typu całkowitego j
   }
}
```

3) Wywołanie programu z listą parametrów

java argi Jan Kowalski

```
public class argi                    // klasa publiczna, nie abstrakcyjna i niefinalna
{
  static int ile;                    //składowa klasowa

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

---

4) Wywołanie programu z listą parametrów

java argi_ Jan Kowalski

```
public class argi_
{
   int ile;

   public static void main(String args[])
   {
    argi_ p = new argi_();
    p.ile=args.length;
    for (int j=0; j<p.ile; j++)
    { System.out.println(args[j]);}
  }
}
```

Dodatek do wykładu
Pliki źródłowe i pakiety

- Program języka Java może się składać z wielu niezależnie kompilowanych modułów źródłowych, w których umieszcza się definicje klas oraz interfejsów

- Moduły źródłowe są przechowywane w plikach o nazwie *Nazwa_klasy.java*, gdzie Nazwa_klasy jest nazwą klasy publicznej – pliki do kompilacji

- Jeżeli w pliku Nazwa_klasy.java zdefiniowano jedną klasę, to w wyniku kompilacji tego pliku powstaje plik wynikowy *Nazwa_klasy.class*.

- Jeżeli program jest aplikacją, to w zestawie modułów źródłowych musi się znaleźć dokładnie jeden moduł źródłowy (moduł główny aplikacji) z klasą publiczną, która zawiera publiczną i statyczną funkcję main (każdy inny moduł źródłowy może zawierać klasę z funkcją main, jeżeli nie jest to klasa publiczna).

- Moduł źródłowy, w którym definicje klas oraz interfejsów poprzedzono deklaracją pakietu o postaci ***package*** *nazwa_pakietu*; staje się pakietem. Deklaracja pakietu rozszerza przestrzeń nazw programu i pozwala na lepsze zarządzanie programem wielomodułowym.

- Jeżeli moduł źródłowy nie zawiera deklaracji pakietu, to należy on do tzw. pakietu domyślnego (pakietu bez nazwy). Np. zadeklarowana wcześniej klasa *Witaj*, umieszczona w pliku *Witaj.java* należy do pakietu domyślnego.

- Pakiety są ściśle powiązane z katalogami, w których umieszcza się moduły źródłowe i pliki wynikowe.

Przykład tworzenia i korzystania z pakietu

Np. w katalogu nowak\myprog\pakiet1 umieszczono główny plik źródłowy aplikacji punkt1.java o postaci:

```
package nowak.myprog.pakiet1;  //klasa punkt1 należy do pakietu nowak.myprog.pakiet1
                               //tylko taka kolejność: package...; import...;
import nowak.myprog.pakiet1.Punkt;
                               //można używać nazwy skrócone składowych klas importowanych
                               //tutaj klasy Punkt
public class punkt1
{
 public static void main (String[] args)
  { Punkt p1 = new Punkt(1, 7);
          //zamiast nowak\myprog\pakiet1\Punkt p1= new nowak\myprog\pakiet1\Punkt (1,7);
    //.......
     p1.zmien(2,5);
 }
}
```

Definicję klasy *Punkt* umieszczono w pliku *Punkt.java* , która ma postać:

```
package nowak.myprog.pakiet1;   //klasa Punkt należy do pakietu nowak.myprog.pakiet1

class Punkt
 {
   // definicja ciała klasy jak w przykładzie 5
  }
```

- **Przebieg kompilacji**

Plik punkt1.java zawiera definicję klasy *punkt1*, poprzedzoną deklaracją **pakietu** oraz deklaracją **importu** klasy *Punkt*. Plik ten może zostać skompilowany (wywołaniem kompilatora javac z katalogu nadrzędnego w stosunku do katalogu nowak).

```
javac nowak\myprog\pakiet1\punkt1.java
```

W wyniku kompilacji zostaną utworzone dwa pliki wynikowe:

punkt1.class , Punkt.class

- **Wykonanie programu**

```
java nowak\myprog\pakiet1\punkt1
```

spowoduje wyprowadzenie na ekran napisów tworzonych prze funkcje *main* publicznej klasy *punkt1*