# Programowanie w języku Java - Klasy, obiekty, inicjalizacja

mgr inż. Maciej Lasota <m.lasota@tu.kielce.pl>

Version 1.0, 27-02-2017

# Spis treści

Zajęcia **laboratoryjne nr 1** z programowania w języku Java - dwa zajęcia **3h**.

# Zmienne, typy danych, tablice

**Zmienna** jest symbolem w programie, oznaczającym *obszar w pamięci komputera*, w którym mogą być zapisywane różne dane:

- zmienna ma nazwę (użyty w programie symbol nazywa się nazwą zmiennej),
- przez tę nazwę odwołujemy się do konkretnego obszaru pamięci, w którym chcemy przechowywać wartości jakiejś danej,
- zawartość tego obszaru (wartość zmiennej) możemy zmieniać w trakcie wykonania programu.

Definiowanie zmiennych:

```
typ danej zmienna [ = wartość][, zmienna [ = wartość] ...];
```

## Typy proste:

*logiczne*

- boolean (true/false)

*znakowe*

- char (16 bitów)

*numeryczne całkowite*

- byte (8 bitów)
- short (16 bitów)
- int (32 bity)
- long (64 bity)

*numeryczne zmiennoprzecinkowe*

- float (32 bity)
- double (64 bity)

*puste*

- void (typ pusty)

Typy numeryczne są typami ze **znakiem** *(signed)*. W języku Java konieczne jest **zainicjowanie zmiennej** przed użyciem

## Typy obiektowe (opakowujące typy proste):

- Boolean (boolean)

- Character (char)
- Byte (byte)
- Short (short)
- Integer (int)
- Long (long)
- Float (float)
- Double (double)
- Void (void)
- BigInteger - typ całkowity dowolnej precyzji
- BigDecimal - typ stałoprzecinkowy dowolnej precyzji
- String (char[]) - łańcuch tekstowy

## Typy złożone (tablice):

**Tablice** są zestawami elementów (wartości) tego samego typu, ułożonych na określonych pozycjach. Do każdego z tych elementów mamy bezpośredni (swobodny - nie wymagający przeglądania innych elementów zestawu) dostęp poprzez nazwę tablicy i pozycję elementu w zestawie, określaną przez **indeks** lub **indeksy tablicy**.

Deklaracja tablicy składa się z:

- nazwy typu elementów tablicy,
- pewnej liczby par nawiasów kwadratowych (liczba par określa liczbę wymiarów tablicy),
- nazwy zmiennej, która identyfikuje tablicę.

Tworzenie tablicy jednowymiarowej typu int

```
int[] tab = new int[10];
```

Tworzenie tablicy wielowymiarowej

```
int[][] tab = new int[10][4];
```

Tworzenie i inicjalizacja zawartości tablicy

```
int[] tab = { 5, 3, 8, 2, 7 };
```

Liczba elementów tablicy

```
tab.length
```

Odwołanie do i-tego elementu tablicy

```
tab[i] lub tab[i][j]
```

Tablica typów **prostych** jest zerowana. Tablica **obiektów** jest tablicą *referencji* pustych (null)

# Klasy, definicja klasy

**Klasa** jest *szablonem*, czy też *projektem* na podstawie którego tworzone są obiekty, które posiadają pewne cechy i funkcje. Klasę winniśmy traktować jako swoisty **wzorzec**, szablon opisujący powstawanie obiektów (konstruktory), ich cechy (pola) oraz sposób komunikowania się z obiektami (metody).

Definicja klasy określa:

- zestaw cech (atrybutów) obiektów klasy,
- zestaw operacji, które można wykonywać na obiektach klasy,
- specjalne operacje, które pozwalają na inicjowanie obiektów przy ich tworzeniu.

W Javie do definiowania klas używa się słowa kluczowego `class`. Samą definicję umieszcza się w następujących po nim **nawiasach klamrowych**. Kod definicji (pomiędzy nawiasami klamrowymi) nazywa się ciałem klasy.

*Przykład 1. Definiowanie klasy*

```
[ public ] class NazwaKlasy  {
    // definicje pól
    // definicje konstruktorów
    // definicje metod
}
```

**Pola klasy** określają (zazwyczaj) z jakich elementów będą składać się obiekty tej klasy.

*Przykład 2. Definiowanie pól klasy*

```
[public] class NazwaKlasy {
    [ specyfikator_dostępu ] [static] nazwa_typu nazwa_zmiennej [ inicjator ];
    //....
}
```

**Metoda** tak samo jak *funkcja* to wyodrębniony zestaw czynności, zapisywany jednorazowo w postaci fragmentu kodu, który może być wywoływany wielokrotnie z innych miejsc programu. Wodróżnieniu od funkcji metody zwykle wywoływane są na rzecz konkretnych obiektów.

*Przykład 3. Definiowanie metod klasy*

```
[public] class NazwaKlasy {
    // ...
    [specyfikator_dostępu] [static]  typ_wyniku nazwa_metody( lista_parametrów ) {
        // ... instrukcje wykonywane po wywołaniu metody
    }
}
```

**Konstruktor** służy (głównie) do inicjowania pól obiektów. O konstruktorze można myśleć jako o *specjalnej metodzie*, która zawsze ma **nazwę taką samą jak nazwa klasy**, nie ma żadnego typu wyniku (nawet void!), ma listę parametrów (w szczególności może być pusta).

*Przykład 4. Definiowanie konstruktorów klasy*

```
[public] class NazwaKlasy {
    // Definicja konstruktora
    [ specyfikator_dostępu ] nazwa_klasy(lista_parametrów) {
        // czynności wykonywane przez konstruktor
    }
}
```

*Przykład 5. Przykład klasy*

```
public class NazwaKlasy {
  //pole (zmienna/stala/referencja)
  public int i;
  //konstruktor (domyslny/sparametryzowany)
  public NazwaKlasy() {

  }
  //metoda
  public void nazwaMetody() {

  }
  //metoda main - punkt wejscia programu
  public static void main(String[] args) {

  }
}
```

- klasa publiczna determinuje nazwę pliku java
- nazewnictwo:
  - klasa – każde słowo w nazwie klasy z dużej litery
  - konstruktor – identycznie jak klasa
  - pole i metoda – pierwsze słowo z małej litery, każde następne z dużej
  - stała – całość dużymi literami
  - pakiet – całość małymi literami

# Obiekty, inicjalizacja, tworzenie, usuwanie obiektów

Klasa jest tylko definicją typu, a dopiero **instancja klasy**, tj. *obiekt*, to **konkretny byt w pamięci komputera**. To dla instancji klasy (tj. dla obiektu) możemy wywoływać metody, które pozwalają operować na polach (atrybutach) składowych danego obiektu. Obiekty tworzymy przy pomocy operatora `new`. Argumentem tego operatora jest *klasa*.

Składnia instrukcji tworzenia obiektu ma następującą postać:

```
new [nazwa klasy]([argumenty inicjalizacji obiektu]);
```

Kolejność inicjalizacji obiektu:

- pola statyczne
- deklaracje
- konstruktor
- instrukcje

Pola są **zerowane**, zmienne lokalne **nie są**.

Tworzenie obiektu:

```
Object o = new Object();
```

Każda klasa automatycznie dziedziczy po klasie **Object** (otrzymuje jej właściwości)

- **this** - wskazanie na aktualny obiekt
- **super** – wskazanie na obiekt klasy bazowej (nadrzędnej)

Klasa Object:

- `.clone()` - tworzy i zwraca kopię aktualnego obiektu

- `.equals(Object)` - sprawdza, czy dany obiekt jest równy aktualnemu
- `.finalize()` - metoda wywoływana przez Garbage Collector, gdy nie istnieje żadna referencja do aktualnego obiektu
- `.toString()` - zwraca reprezentację tekstową obiektu

Usuwanie obiektu:

- brak destruktorów – pamięć odśmiecana automatycznie (Garbage Collector)
- gdy klasa rezerwuje pamięć w sposób niestandardowy, należy zwolnić ją ręcznie

```
protected void finalize() throws Throwable {
  //...
  super.finalize();
}
```

```
System.gc()
System.runFinalization()
```

# Referencje

**Referencja** to wartość, która oznacza *lokalizację (adres)* obiektu w pamięci. Wszystkie zmienne deklarowane z nazwą klasy w miejscu nazwy typu są zmiennymi **typu referencyjnego**. Zmienne te mogą zawierać referencje do obiektów lub nie zawierać żadnej referencji (nie wskazywać na żaden obiekt). Obiekty, na które w programie nie wskazuje już żadna referencja są automatycznie usuwane z pamięci. Nazywa się to **automatycznym odśmiecaniem** *(garbage collecting)*.

- obiekty musimy tworzyć za pomocą wyrażenia `new`
- na obiektach operujemy za pomocą **referencji** (które na nie wskazują) i poleceń (metod) zdefiniowanych w klasie obiektów
- referencje **nie są obiektami** - są adresami obiektów
- zmienne typów referencyjnych musimy (tak samo jak zmienne innych typów) deklarować przed ich użyciem w programie
- deklaracja zmiennej-referencji nie tworzy obiektu

# Przykładowy program

*Przykład 6. Przykładowy program*

```
class Samochod {

  public String marka;
  public int rok;

  //konstruktor
  public Samochod(String marka, int rok) {
    this.marka = marka;
    this.rok = rok;
  }

  //przeslonieta metoda
  public String toString() {
    return marka + " rocznik " + rok;
  }
}

public class Test {
  public static void main(String[] args) {
    System.out.println(new Samochod("BMW", 2005));
    //automatyczna konwersja typu - wywolanie metody toString
  }
}
```

```
BMW rocznik 2005
```

Instrukcja **return** – natychmiastowy powrót z metody.

# Bibliografia


- **Bruce Eckels**, *"Thinking in Java. Edycja polska. Wydanie IV"*, wydawnictwo Helion.
- **Cay S. Horstmann, Gary Cornell**, *"Java. Podstawy. Wydanie IX"*, wydawnictwo Helion.
- **Cay S. Horstmann, Gary Cornell**, *"Java. Techniki zaawansowane. Wydanie IX"*, wydawnictwo Helion.
- **Krzysztof Barteczko**, *"Podstawy programowania w języku Java, PJWSTK"*, http://edu.pjwstk.edu.pl/wyklady/ppj/scb/
- **Konrad Kurczyna**, *"Laboratorium Java"*, Politechnika Świętokrzyska w Kielcach.
- **Mariusz Lipiński**, *"Nauka Javy"*, http://www.naukajavy.pl/