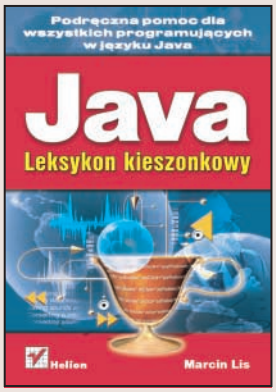
Java. Leksykon kieszonkowy



Autor: Marcin Lis
ISBN: 83-7361-778-7
Format: B6, stron: 140

Popularność Javy rośnie w ogromnym tempie. Jeszcze do niedawna była ona najczęściej kojarzona z internetem i telefonami komórkowymi. Obecnie coraz większe grono programistów docenia ten doskonały, obiektowy język programowania, nadający się do różnorodnych zastosowań: od krótkich apletów do poważnych aplikacji. W oparciu o platformę Java 2 tworzone są ogromne projekty, powstają dla niej doskonałe środowiska programistyczne, a jej znajomość jest wysoko ceniona na rynku pracy.

„Java. Leksykon kieszonkowy" to przewodnik dla programistów Javy. Prezentuje struktury i konstrukcje języka, a także sposoby wykonywania podstawowych zadań programistycznych, takich jak operacje wejścia-wyjścia, tworzenie i obsługa apletów czy wykorzystanie komponentów w aplikacjach z interfejsem graficznym. Książka ta może pełnić rolę podręcznej „ściągi", wykorzystywanej podczas codziennej pracy, jak również materiału uzupełniającego przy nauce Javy.

- Struktura programu
- Komentarze
- Typy danych
- Tablice
- Stosowanie zmiennych
- Operatory
- Konstrukcje sterujące
- Klasy i obiekty
- Tworzenie i korzystanie z pakietów
- Obsługa wyjątków
- Operacje na plikach
- Aplety
- Obsługa grafiki, dźwięku, myszy i klawiatury

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Klasy i obiekty

Tworzenie klas

Klasy są opisami obiektów, czyli bytów programistycznych, które mogą przechowywać dane oraz wykonywać polecone przez programistę zadania. Każdy obiekt jest instancją, czyli wystąpieniem jakiejś klasy. W związku z tym klasa określa także typ danego obiektu. Schematyczny szkielet klasy wygląda następująco:

```
class nazwa_klasy
{
//treść klasy
}
```

W treści klasy są definiowane pola i metody. Pola służą do przechowywania danych, metody do wykonywania różnych operacji. Zatem pola klasy to po prostu zmienne określonych typów, zarówno prostych, jak i obiektowych. Przy nadawaniu nazw klasom występują takie same ograniczenia, jak w przypadku nazewnictwa zmiennych i innych identyfikatorów, czyli nazwa klasy może składać się jedynie z liter (zarówno małych, jak i dużych), cyfr oraz znaku podkreślenia, ale nie może zaczynać się od cyfry. Nie zaleca się również stosowania polskich znaków diakrytycznych, zwłaszcza że nazwa klasy musi być zgodna z nazwą pliku, w którym dana klasa została zapisana.

Aby utworzyć zmienną typu obiektowego (klasowego, referencyjnego), należy skorzystać z konstrukcji:

```
nazwa_klasy nazwa_zmiennej;
```

Do tak zadeklarowanej zmiennej można następnie przypisać obiekt utworzony za pomocą operatora `new`:

```
new nazwa_klasy();
```

Jednoczesna deklaracja zmiennej, utworzenie obiektu i przypisanie go do zmiennej odbywa się za pomocą schematycznej konstrukcji[1]:

```
nazwa_klasy nazwa_zmiennej = new nazwa_klasy();
```

Pola klas

Definicje pól

Pola definiowane są w ciele klasy, w sposób identyczny jak zwykłe zmienne. Najpierw należy podać typ pola, a po nim nazwę pola. Schematycznie wygląda to następująco:

```
class nazwa_klasy
{
  typ_pola1 nazwa_pola1;
  typ_pola2 nazwa_pola2;
  //...
  typ_polan nazwa_polan;
}
```

Przykładowa klasa o nazwie `Punkt`, zawierająca trzy pola typu *int* o nazwach `x`, `y` i `z`, będzie miała następującą postać:

```
class Punkt
{
  int x;
  int y;
  int z;
}
```

Odwołania do pól obiektu

Po utworzeniu obiektu do jego pól można odwoływać się za pomocą operatora `.` (kropka), schematycznie:

```
nazwa_obiektu.nazwa_pola;
```

Przykład:

```
Punkt punkt1 = new Punkt();
punkt1.x = 100;
```

[1] Zapis `nazwa_klasy()` to nic innego, jak wywołanie bezargumentowego konstruktora danej klasy.

```
punkt1.y = 200;
```

Wartości domyślne pól

Każde niezainicjowane pole klasy otrzymuje wartość domyślną, zależną od jego typu. Wartości te zaprezentowane zostały w tabeli 13.

Tabela 13. Wartości domyślne pól

Typ	Wartość domyślna
byte	0
short	0
int	0
long	0
float	0,0
dubloe	0,0
char	\0
boolean	false
obiektowy	null

Metody klas

Definicje metod

Metody definiowane są w ciele klasy pomiędzy nawiasami klamrowymi. Każda metoda może przyjmować argumenty oraz zwracać wynik. Schematyczna deklaracja metody wygląda następująco:

```
typ_wyniku nazwa_metody(parametry_metody)
{
//instrukcje metody
}
```

Po umieszczeniu w ciele klasy deklaracja taka będzie miała postać:

```
class nazwa_klasy
{
  typ_wyniku nazwa_metody(parametry metody)
  {
        //instrukcje metody
  }
}
```

Jeśli metoda nie zwraca żadnego wyniku, jako typ wyniku należy zastosować słowo `void`, jeśli natomiast nie przyjmuje żadnych argumentów, pomiędzy nawiasami okrągłymi nie należy nic wpisywać.

Odwołania do metod

Po utworzeniu obiektu do jego metod można odwoływać się analogicznie jak do pól, czyli za pomocą operatora . (kropka), schematycznie:

```
nazwa_obiektu.nazwa_metody();
```

Zakładając, że istnieje klasa `Punkt` zawierająca bezargumentową metodę o nazwie `wyswietlWspolrzedne` oraz zmienna referencyjna `punkt1` wskazująca na obiekt tej klasy, wywołanie metody będzie miało postać:

```
punkt1.wyswietlWspolrzedne();
```

Argumenty metod

Argumenty metody to inaczej dane, które można jej przekazać. Metoda może mieć dowolną liczbę argumentów umieszczonych w nawiasach okrągłych za jej nazwą. Poszczególne argumenty oddzielane są od siebie znakiem przecinka. Schematycznie wygląda to następująco:

```
typ_wyniku nazwa_metody(typ_parametru_1 nazwa_parametru_1,
↳typ_parametru_2 nazwa_parametru_2, ... , typ_parametru_n
↳nazwa_parametru_n)
```

Przykład:

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

Argumentami mogą być zarówno typy proste, jak i typy obiektowe.

Przeciążanie metod

W każdej klasie mogą istnieć dwie lub więcej metod, które mają takie same nazwy, o ile tylko różnią się argumentami. Mogą — ale nie muszą — również różnić się typem zwracanego wyniku. Technika ta nazywa się przeciążaniem metod. Przykładowa klasa zawierająca dwie przeciążone metody wygląda następująco:

```
public
class Punkt
{
  int x;
  int y;
  void ustawXY(int wspX, int wspY)
  {
    x = wspX;
    y = wspY;
  }
  void ustawXY(Punkt punkt)
  {
    x = punkt.x;
    y = punkt.y;
  }
}
```