Metody programowania

Wprowadzenie do platformy Java

Dr inż. Andrzej Grosser

Częstochowa, 2013

Spis treści

1. Podstawy wyrażeń i instrukcji

1.1. Wstęp

Niniejszy rozdział dostarcza najważniejszych informacji związanych z typami podstawowymi Javy. Zaprezentowano także sposoby budowania wyrażeń z użyciem operatorów. Omówiono również instrukcji sterujące w tym języku. Do przygotowania tego rozdziału użyto następujących pozycji literaturowych [1], [2] i [3].

1.2. Typy podstawowe

Oprócz typów obiektowych Java wprowadza dodatkowo zestaw typów podstawowych (prymitywnych). Powodem ich wprowadzenia były kwestie wydajnościowe. Zmienne obiektowe są alokowane na stercie, taka strategia zarządzania pamięcia nie miałaby większego sensu dla zmiennych, które zawierają liczby lub znaki, powodowałaby dodatkowe narzuty związane z alokacją i dostępem do zmiennej.

Tabela 1.1. Typy podstawowe

Typ danych	Opis	Wartości
byte	liczba całkowita o długości bajtu	-2^7 do 2^7-1
short	krótka liczba całkowita	-2^{15} do $2^{15}-1$
int	liczba całkowita	-2^{31} do $2^{31}-1$
long	długa liczba całkowita	-2^{63} do $2^{63}-1$
float	liczba rzeczywista pojedynczej precyzji	IEE754
double	liczba rzeczywista podwójnej precyzji	IEE754
char	znak	Unicode
boolean	wartość logiczna	true, false

Tabela 1.1 zawiera podsumowanie informacji o podstawowych typach. Proszę zwrócić uwagę, że w Javie występują jedynie liczby całkowite ze znakiem.

1.3. Komentarze

Oczywiście nie trzeba przypominać, że dobrą praktyką programistyczną jest komentowanie własnego kodu. Ułatwia to przecież późniejsza analizę, przypomina także, nawet autorowi, co kod robi.

Język Java przejmuje po C++ komentarze. W Javie również da się zapisać komentarze na dwa sposoby:

- komentarze jednolinijkowe - zaczynające się od `//` i kończące się znakiem końca linii,
- komentarze, które można zapisać w wielu liniach - zaczynające się do znaków `/*` i kończące się znakami `\*`.

Na przykład komentarz linijkowy:

```
1 int x = 0; //Przechowuje wartosc zmiennej x
```

Na przykład komentarz drugiego rodzaju:

```
1 /* Metoda suma jest  odpowiedzialna za sumowanie argumentow */
2 int suma(int x, int y)
3 {
4     return x + y;
5 }
```

1.4. Zmienne

Java jest językiem statycznie typowanym, więc każda zmienna musi mieć swój typ. Dlatego też definicja każdej zmiennej musi być poprzedzona nazwą typu. Nowej zmiennej można też nadać wartość początkową (bez wartości początkowej przyjmuje wartość domyślną dla typu). W nazwach są też rozróżniane wielkości liter. W jednej definicji można podać więcej niż jedną nazwę zmiennej. Na przykład:

```
1 int x;
2 double y;
3 int i, j;
4 char zn = 'x';
```

Reguły definiowania są, więc takie same jak w C++, ale zasady nazewnictwa zmiennych są już nieco inne, gdyż dozwolone jest używanie w nazwach zmiennych znaków Unicode (pod warunkiem, że dany znak jest literą w jakimś alfabecie). Nazwa zmiennej musi zaczynać się literą lub znakiem podkreślenia `_`, po którym mogą występować litery, cyfry i znaki podkreślenia. Przykładami poprawnych nazw zmiennych są

- `x`
- żaba
- `_y`
- x121

Natomiast niepoprawnym są:

- 1x - nazwa zaczyna się od cyfry,
- x y - zawiera spację w nazwie,
- x+y - zawiera znak niebędący literą, cyfrą lub znakiem podkreślenia.

1.5. Stałe

Java nie używa słowa kluczowego `const` (jest tylko słowem zarezerwowanym). Do definiowania stałych wymagane jest słowo kluczowe `final`, po którym następuje typ, nazwa stałej i opcjonalna część zaczynająca się od operatora `=` powodująca nadanie wartości stałej (bez tego przyjmuje wartość domyślną). Na przykład:

```
1 final int x = 5;
2 final int y;
3 final double y = 12.0;
```

Reguły nazewnictwa są takie same jak dla zmiennych.

1.6. Operatory

Java dostarcza zestaw podstawowych operatorów pozwalający wykonywać najważniejsze działania takie jak dodawanie, odejmowanie, mnożenie, dzielenie, przesunięcie bitowe itp.

Java wyróżnia następujące kategorie operatorów:

- operatory jednoargumentowe - jednoargumentowy + i -, inkrementacja ++ i dekrementacja `--`, logiczne zaprzeczenie `!`, bitowe zaprzeczenie `~`,
- operatory arytmetyczne- dodawanie `+`, odejmowanie `-`, mnożenie `*`, dzielenie `/`, porównanie wartości równy `==` i różny `!=`
- operatory relacyjne - mniejszy `<`, większy `>`, mniejszy lub równy `<=`, większy lub równy `>=` i `instanceof`.
- operatory warunkowe - koniunkcja `&&`, alternatywa `||` i trójargumentowe wyrażenie warunkowe `?:`.
- operatory przesunięcia: przesunięcie bitowe w lewo `<<`, przesunięcie bitowe w prawo `>>` i przesunięcie `>>>`.
- operatory przypisania wersja prosta = i wersje złożone np. `+=`,
- operatory logiczne i bitowe - `&`, `|` i `^`.

Tabela 1.2. Hierarchia operatorów

Rodzaj operatora	Łączność
[].() (wywołanie metody)	od lewej do prawej
`++ -- + - ()` (rzutowanie) `new`	od prawej do lewej
`* / %`	od lewej do prawej
`+ -` (dwuargumentowe)	od lewej do prawej
`<< >> >>>`	od lewej do prawej
`< > <= >= instanceof`	od lewej do prawej
`== !=`	od lewej do prawej
`&`	od lewej do prawej
`^`	od lewej do prawej
`|`	od lewej do prawej
`&&`	od lewej do prawej
`||`	od lewej do prawej
`?:`	od lewej do prawej
`= += -=` itd	od prawej do lewej

Tabela 1.2 przedstawia hierarchię operatorów i ich łączność. Operatory o wyższym priorytecie są zapisywane w tabeli wyżej niż operatory o niższym priorytecie. Łączność oznacza sposób wartościowania:

1. od lewej do prawej oznacza, że będą wartościowane od lewej do prawej np: x + y + z będzie wartościowane jako (x + y) + z - gdzie nawiasy pokazują, które działanie będzie wykonywane najpierw,
2. od prawej do lewej oznacza, że będą wartościowane od prawej do lewej np. x = y = z będzie wartościowane jako x = (y = z) - gdzie nawiasy pokazują, które działanie będzie wykonywane najpierw.

Znaczenie tych operatorów jest w większości przypadków jest takie samo jak w C++. Zostaną omówione jedynie operatory, których nie ma w C++ lub ich działanie jest inne.

1. Operatory `&` i `|` oprócz wykonywania operacji bitowych mogą być także użyte do argumentów typu logicznego. Działają wtedy jak operatory `&&` i `||`, jedyną różnicą jest to, że wyliczając oba argumenty, nie stosują skróconego sposobu wyliczeń (w przypadku `&&`, gdy pierwszy argument jest wartościowany do fałszu nie jest wyliczany drugi argument, w przypadku `||`, gdy pierwszy argument jest wartościowany jest do prawdy nie jest wyliczany drugi argument).
2. Operatory `>>`, `<<`, `>>>` - są operatorami przesunięcia bitowego. Semantyka ich jest następująca:
   a) `>>` - przesuwa bity w prawo, najstarsze bity są wypełniane bitem znaku,
   b) `>>>` - przesuwa bity w prawo, najstarsze bity są wypełniane zerami,
   c) `<<` - przesuwa bity w lewo, najmłodsze bity są wypełniane zerami.
3. Znaczenie `instanceof` zostanie omówione przy okazji klas.

1.7. Podstawowe konwersje

1.7.1. Konwersje pomiędzy typami numerycznymi

Bez operatora rzutowania (wykonywana jest niejawna konwersja) możliwe jest tylko awansowanie typu zmiennej (zakres wartości typu, do którego jest konwertowana zmienna jest większy), dlatego możliwe są jedynie następujące konwersje:

1. z `byte` do `short`,
2. z `short` do `int`,

3. z `int` do `long`,
4. z `char` do `int`,
5. z typów całkowitych do rzeczywistych (możliwa jest jednak utrata precyzji obliczeń).

1.7.2. Rzutowanie

W sytuacji, gdy istnieje potrzeba rzutowania w dół (z typu o szerszym zakresie do typu o węższym zakresie) programista musi jawnie poinformować kompilator o swoich intencjach poprzez użycie operatora rzutowania. Rzutowanie jest zapisywane w postaci nawiasów okrągłych, w których jest typ, a po nawiasie zamykającym nazwa zmiennej.

```
1 double z = 4.65;
2 int m = (int)z;
```

Tego rodzaju konwersja jest potencjalnie niebezpieczna - może po prostu wartość nie zmieścić się w wartościach typu węższego, nastąpi obcięcie najstarszych bitów. Na przykład po wykonaniu poniższych instrukcji yy będzie miało wartość -127:

```
1 int xx = 129;
2 byte yy = (byte)xx;
```

1.8. Instrukcje sterujące

Zestaw instrukcji sterujących Java ma praktycznie identyczny jak C++. Java nie pozwala jednak na szereg konstrukcji, jakie dozwolone są w C++. Dla przykładu w warunkach instrukcji sterujących muszą być zapisywane instrukcje, które są wartościowane do typu logicznego. Nie można, więc użyć w warunku wyrażenia, które jest wartościowane do typu całkowitego (co jest dopuszczalne w C++). Na przykład:

```
1 while(1)
2 {}
```

musi być zapisane w postaci:

```
1 while(true)
2 {}
```

1.8.1. Blok instrukcji

Podobnie jak w C++ blok instrukcji (instrukcję złożoną) zapisuje się pomiędzy nawiasami klamrowymi. Dozwolone jest definiowanie wewnątrz bloku zmiennych, zasięg ich widoczności ogranicza się od miejsca zdefiniowania aż do końca bloku. Na przykład:

```
1 {
2     int x = 10;
3     y = x + 5;
4 }
```

Zdefiniowana powyżej zmienna x jest widoczna tylko i wyłącznie w bloku instrukcji.

W przeciwieństwie do C++ w Javie zabronione jest przesłanianie zmiennych (ponowne definiowanie zmiennych w bloku o takiej samej nazwie, jak zdefiniowana wcześniej na zewnątrz bloku zmienna). Na przykład poniższy kod jest niepoprawny w Javie:

```
1 int x = 5;
2 {
3     int x = 10;
4 }
```

1.8.2. Instrukcja warunkowa

Instrukcja warunkowa w Javie wygląda składniowo identycznie jak w C++. To znaczy:

- `if (warunek) instrukcja`,

  Jeśli warunek jest spełniony to wykonywana jest instrukcja. Na przykład dla kodu zamieszczonego poniżej, jeżeli x jest równy zero, to zostanie wypisany w konsoli napis X jest równe 0:

  ```
  1 if(x == 0) {
  2     System.out.println("X jest rowne zero");
  3 }
  ```

- `if (warunek) instrukcja1 else instrukcja2`,

  Jeśli warunek jest spełniony wykonywana jest instrukcja1, w przeciwnym przypadku wykonywana jest instrukcja2. Na przykład dla poniższego kodu, gdy x będzie większy od zera, y przyjmie wartość 1, w przeciwnym przypadku wartość -1:

```
1 if(x > 0) {
2     y = 1;
3 }
4 else {
5     y = -1;
6 }
```

1.8.3. Instrukcja wyboru

Składnia instrukcji wyboru w Javie jest identyczna jak w C++. To znaczy:

```
switch (wyrazenie) {
  case wartosc1:
               instrukcje1
               break;
               ...
  case wartoscn:
              instrukcjen
  default:
              instrukcjaD
}
```

Na początku obliczanie jest wyrażenie występujące po słowie kluczowym `switch`. Wyrażenie to musi być wartościowane do typu całkowitego, dopuszczalne jest też wartościowanie do typu łańcuchowego (`String`) i wyliczeniowego. Następnie wybierana jest etykieta, której wartość odpowiada wartości wyrażenia obliczonego na samym początku. Potem obliczane są kolejne instrukcje, aż do napotkania słowa kluczowego `break`, `return` lub końca instrukcji wyboru. Dopuszczalne jest także używanie klauzuli `default`, która jest wybierana w sytuacji, gdy żadna z etykiet nie pasowała do obliczonej wartości wyrażenia. Na przykład dla kodu poniżej:

```
1 switch(x)
2 {
3   case 1:
```

```
4         y = 1;
5     case 2:
6         y = 2;
7         break;
8     case 3:
9         y = 3;
10     default:
11        y = 0;
12 }
```

jeżeli x jest równe 1 lub 2, to y przyjmie wartość 2, zaś dla każdej innej wartości x, y przyjmie 0.

1.8.4. Instrukcje while i do while

Składnia instrukcji `while` i `do while` jest identyczna jak w C++. To znaczy:

– `while(warunek) instrukcja`

Dopóki warunek jest spełniony (wartościowany jest do prawdy) wykonywana jest instrukcja wewnątrz pętli (może być blokiem instrukcji). Na przykład:

```
1 while(x > 0)
2 {
3       x -= 1;
4 }
```

Powyższa pętla `while` jest wykonywana do momentu, gdy x przyjmie wartość zero.

– `do instrukcja while(warunek);`

W tej sytuacji instrukcja wewnątrz pętli wykonywana jest przynajmniej raz, warunek jest sprawdzony po jej wykonaniu. Jeżeli warunek jest wartościowany do prawdy wykonywanie instrukcji wewnątrz pętli powtarza się. Na przykład:

```
1 do {
2     x -= 1;
3 }
4 while(x > 0);
```

Wykonywanie powyższej pętli jest kontynuowanie, aż do momentu gdy x przestanie być większy od zera.

1.8.5. Instrukcja for

Instrukcja `for` występuje w dwóch postaciach:

1. `for(wyrazenie1;wyrazenie2;wyrazenie3) instrukcja`

   Wszystkie wyrażenia są opcjonalne (nie muszą wystąpić w instrukcji, w takiej sytuacji nie są po prostu obliczane). Wykonywana wewnątrz pętli instrukcja może wystąpić w postaci instrukcji złożonej (bloku kodu).

   Instrukcja w zwykłej (z wszystkimi wyrażeniami) postaci będzie wykonywana następująco:

   a) Obliczane jest wyrażenie1 - tylko raz na początku wykonywania pętli (może zawierać definicje zmiennych, których zasięg widoczności będzie się ograniczał do ciała pętli).

   b) Wartościowane jest wyrażenie2, jeżeli zwaraca prawdę wykonywana jest instrukcja w przeciwnym razie będzie koniec wykonywania pętli.

   c) Obliczane jest wyrażenie3, sterowanie wraca do punktu poprzedniego (b).

   Przykład działania instrukcji `for` można zilustrować następującym fragmentem kodu:

```
1 int sum = 0;
2 for(int i = 1; i < 10; ++i)
3 {
4     sum += i;
5 }
```

   W powyższej pętli są sumowane liczby od 1 do 9.

2. `for(zmienna : kolekcja)`

   W czasie pisanie programów, często wykonywaną operacją jest przechodzenie po wszystkich elementach tablicy i wykonywanie określonych operacji. Java (C++ w nowym standardzie też dostał tą pętlę) wprowadza wygodną formę instrukcji `for` pozwalającą na przejście po kolejnych elementach tablicy. W kolejnych iteracjach zmienna będzie przyjmowała wartości kolejnych elementów.

   Działanie tej instrukcji sterującej można przedstawić w sposób następujący:

```
1 double tab[] = {1,2,4};
2
3 for(double e : tab)
4 {
5     System.out.println(e);
6 }
```

W pętli, zapisanej powyżej, są wypisywane wszystkie elementy tablicy tab.

1.8.6. Instrukcje przerywające sterowanie - break, continue

Znaczenie tych instrukcji jest takie samo jak w przypadku C++ - break przerywa wykonywanie bieżącej pętli, natomiast w przypadku continue sterowanie wraca do sprawdzania warunku pętli. W przeciwieństwie do C++ instrukcje te mogą być opatrzone etykietę, do której nastąpi skok. Jest to przydatne w sytuacji, gdy trzeba przerwać od razu bardzo zagnieżdżoną pętlę, na przykład:

```
1 etykieta:
2 while(x > 0)
3 {
4 //...
5     for(int i = 0; i < n; ++i)
6     {
7         if(x > 0)
8             break etykieta;
9     }
10 }
```

Nie można wskoczyć do bloku kodu, można tylko z niego wyskoczyć. Nie jest również możliwy skok do etykiety zdefiniowanej dalej w kodzie (można skoczyć tylko do wcześniejszej).

Literatura

[1] B. Eckel. *Thinking in Java*. Helion, 2006.

[2] C. S. Horstmann and G. Cornell. *Core Java Volume I–Fundamentals*. Prentice Hall, 2011.

[3] B. Kurniawan. *Java 7: A Beginner's Tutorial*. Brainy Software, 2011.