Typy danych, zmienne i tablice

Tomasz Borzyszkowski

Silne typy Javy

Java jest językiem wyposażonym w **silny system typów**. Wywodzi się stąd siła i bezpieczeństwo tego języka.

Co to znaczy silny system typów?

- Każda zmienna i każde wyrażenie posiada typ i każdy typ jest ściśle zdefiniowany
- Każde przypisanie wartości (podstawienie lub przesłanie wartości do metody) jest sprawdzane pod kątem poprawności typów; w przypadku niezgodności nie ma automatycznych rzutowań, czy konwersji (jak w niektórych językach: C, C++, ...)
- Kompilacja programu kończy się powodzeniem dopiero, gdy typowanie wszystkich wyrażeń kończy się powodzeniem

Konieczność pisania ściśle utypowionego kodu może wydawać się uciążliwa, jednak pozwala uniknąć wielu sytuacji błędnych.

Typy proste

W języku Java zdefiniowano osiem **typów prostych**, które mogą być podzielone na cztery grupy:

- **Typy całkowite**: **byte**, **short**, **int** i **long**; wszystkie one są liczbami ze znakiem o różnym zakresie
- **Typy rzeczywiste**: **float** i **double**; reprezentują liczby rzeczywiste
- **Typ znakowy**: **char**; reprezentuje pojedyncze znaki
- **Typ logiczny**: **boolean**; reprezentuje wartości logiczne (tj. typu prawda-fałsz) **Zobacz**: **BoolDemo.java**

Typy proste są reprezentują pojedyncze wartości, a nie złożone obiekty. Chociaż Java jest językiem zorientowanym obiektowo, to typy proste nie bazują na modelu obiektowym. *Jakie są powody takiego rozwiązania?* Odpowiedź: *efektywność obliczeń.*

Typy całkowite

Zobacz: **LightTravel.java**

Typy całkowite to:

Nazwa	Ilość bitów	Zakres
`long`	64	-9 223 372 036 854 775 808 9 223 372 036 854 775 807
`int`	32	-2 147 483 648 2 147 483 647
`short`	16	-32 768 32 767
`byte`	8	-128 127

Wszystkie powyższe typy są typami ze znakiem. Java nie posiada typów całkowitych bez znaku. W językach programowania zawierających typy całkowite ze znakiem służą one głównie do operacji na bitach. Java umożliwia także operacje na bitach lecz w inny sposób (opowiemy o tym później).

Rozmiar danego typu całkowitego nie jest istotny. Istotne jest jego zachowanie. Często typy mniejsze są implementowane za pomocą większej ilości bitów niż potrzeba, np. **short** na 32 bitach.

Typy rzeczywiste

Nazwa	Ilość bitów	Zakres	
`double`	64	1.7e-308	1.7e+308 (15 cyfr)
`float`	32	3.4e-038	3.4e+038 (6-7 cyfr)

`float` liczby tego typu zajmują 32 bity i na niektórych (tradycyjnych) komputerach są obliczane szybciej niż liczby 64 bitowe; liczby tego typu stają się nieprecyzyjne, gdy obliczane wartości są b. małe lub b. duże

`double` liczby tego typu zajmują 64 bity i na niektórych (nowych) komputerach są obliczane szybciej niż liczby 32 bitowe; funkcje matematyczne takie jak **sin**, **cos**, **sqrt** zawracają wynik typu `double` jako wynik najbardziej dokładny

Zobacz: **Area.java**

Typy rzeczywiste nie liczby

W Java zgodnie ze standardem IEEE 754 zdefiniowano specjalne wartości zmiennoprzecinkowe:

Dodatnia nieskończoność: Double.POSITIVE_INFINITY np.: 100/0
Ujemna nieskończoność: Double.NEGATIVE_INFINITY np.: -100/0
NaN - „Not a Number": Double.NaN np.: 0/0 lub Math.sqrt(-1)

Zobacz: **Nan.java**

Typ znakowy

Zobacz: **CharDemo.java**

`char` reprezentuje znaki w języku Java. Nie są one tymi samymi znakami co znaki języków C i C++ (8-bitowe liczby). Java reprezenuje znaki zgodnie ze standardem **Unicode**.

Unicode pełni międzynarodowy zestawem znaków reprezentującym wszystkie znaki występujące w ludzkich językach. Zastaw ten ujednolica znaki alfabetów: łacińskiego, greckiego, katakana, hangul, ... Aby pamiętać tak wiele znaków potrzeba 16 bitów (zakres 0 do 65 536) Standardowe znaki ASCII to zakres 0 od 127. ISO-Latin-1 0 do 255. Patrz: http://www.unicode.org

Znaki specjalne:

\ddd	znak 8kowo	**\r**	początek linii
\uxxxx	znak UNICODE 16tkowo	**\n**	koniec linii
		\f	pusta strona
**\' \" **	znaki ' " i \	**\t\b**	tab i backspace

Zakres zmiennych

Zakres zmiennej jest to obszar programu w którym dana zmienna jest rozpoznawalna. Zwykle zakresem zmiennej jest blok programu w którym jest zdefiniowana (np.: klasa, nawiasy klamrowe, ...)

W większości języków programowania dostępne są dwa rodzaje zakresów: globalne lokalne (przysłaniające globalne)

W Java wszystkie zmienne zachowują się jak zmienne globalne bloku, w którym są definiowane - brak przesłaniania.

Zasada ogólna: zmienne zdefiniowane wewnątrz zakresu nie są dostępne poza nim. Zakresy mogą być zagnieżdżone. Obiekty zdefiniowane w zakresie zewnętrznym są dostępne w zakresie wewnętrznym, ale nie na odwrót.

Zobacz:
Scope.java **ScopeErr.java**

Konwersje i rzutowania typów

Częstą praktyką jest przypisywanie wartości jednego typu, zmiennym innego typu. W Java, jeżeli typy są **zgodne**, to typ wyrażenia przypisywanego jest **automatycznie przekształcany** na typ zmiennej docelowej, np.: przypisanie zmiennej typu **long** wartości typu **int**.

Automatyczne przekształcenia typów ... ma miejsce gdy:

Oba typy są zgodne

Typ docelowy jest większy niż typ źródłowy

Gdy oba pow. warunki są spełnione typ źródłowy jest rozszerzany do typu docelowego w sposób automatyczny.

Przykłady typów zgodnych:

`int` i `float`

`long` i `float`

Przykłady typów niezgodnych:

`int` i `char`

`int` i `boolean`

`char` i `boolean`

Konwersje i rzutowania typów cd

Rzutowania typów niezgodnych

Czasem zachodzi konieczność przypisania wartości typu *większego*, np. `int`, zmiennej typu *mniejszego* np. `byte`. Nazywa się to **konwersją zawężającą**. W programie efekt ten uzyskuje się przez **jawne rzutowanie**:

```
(typ-docelowy) wartość
```

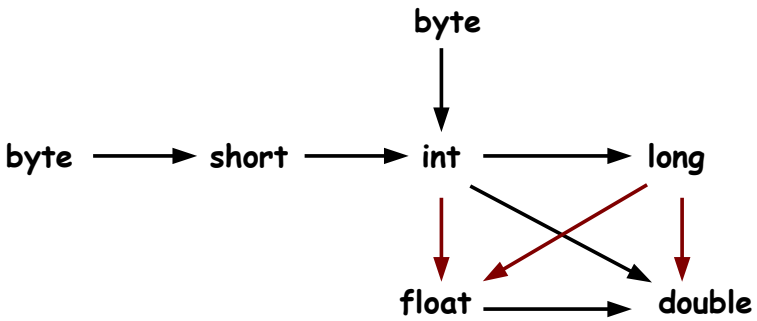
Reguły konwersji zawężającej:

Konwersja typów całkowitych: wartość typu większego jest zredukowana modulo maksymalny zakres typu docelowego, np. `(byte) i`, dla `i = 257` to `1`.

Konwersja typów rzeczywistych na całkowite: wartość typu rzeczywistego traci część dziesiętną i następnie otrzymana wartość całkowita jest przekształcana jak liczby całkowite

Zobacz: **Conversion.java**

Legalne konwersje i rzutowania



Tablice jednowymiarowe

Tablice w Java są realizowane inaczej niż w C i C++, chociaż stosuje się podobną składnię.

Tworzenie tablicy przebiega w **dwóch etapach**:

Deklaracja zmiennej typu tablicowego:

```
typ zmienna_tablicowa[];
```

Deklaracja taka tworzy zmienną tablicową z przypisaną wartością `null` reprezentującą tablicę bez elementów.

Alokacja pamięci niezbędnej do przechowywania elementów tablicy:

```
zmienna_tablicowa = new typ[rozmiar];
```

`typ` jest typem elementów tablicy, `rozmiar` jest liczbą elementów. Taki rodzaj alokacji nazywamy dynamicznym. W Java wszystkie tablice są dynamicznie alokowane.

Zobacz: **Array.java**

Tablice jednowymiarowe cd

Deklarację i alokację tablicy można zapisać w jednej linii. Np. w programie **Array.java** można zapisać tak:

```
int month_days[] = new int[12];
```

Deklarację tablicy można również połączyć podając po przecinku jej elementy, bez używania **new**:

```
int month_days[] = {31, 28, 31, 30, ..., 31};
```

Tablica zostanie utworzona automatycznie z zaalokowanym miejscem wystarczającym do przechowania wszystkich elementów.

Java dokładnie sprawdza w czasie wykonania programu, czy program nie próbuje dostępu do elementu tablicy o indeksie spoza dopuszczalnego zakresu. Jest to kolejna różnica w implementacji tablic Javy i C.

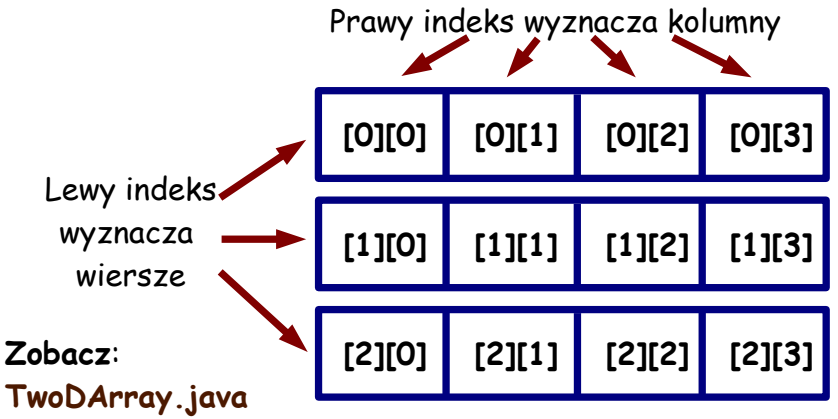
Zobacz: **Average.java**

Tablice wielowymiarowe

W Java tablice wielowymiarowe są tablicami tablic. Deklaracja przykładowej tablicy wygląda następująco:

```
int twoD[][] = new int[3][4];
```

Koncepcyjnie tablica ta wygląda następująco:



Zobacz: **TwoDArray.java**

Tablice wielowymiarowe cd

Alokując pamięć na tablicę wielowymiarową trzeba co najmniej określić pierwszy (najbardziej lewy - liczba kolumn) wymiar. Resztę, tj. liczbę elementów w każdym wierszu można alokować oddzielnie. Nie wszystkie wiersze muszą posiadać taką samą liczbę elementów.

Zobacz: **TwoDAgain.java**

Podobnie jak w przypadku tablic jednowymiarowych można alokować tablicę wielowymiarową podając jej elementy.

Zobacz: **Matrix.java**

Tablice mogą mieć dowolną liczbę wymiarów.

Zobacz: **ThreeDMatrix.java**

Tablice wielowymiarowe cd

Alternatywna składnia deklaracji tablic:

```
int a[] = new int[3];  int b[][] = new int[3][4];
int[] a = new int[3];  int[][] b = new int[3][4];
```

Zobacz: **WyborLosowy.java**

Zobacz: **PrzyrostOdsetek.java**

Zobacz: **TablicaLoterii.java**

Typy String i Pointer

W językach programowania zwykle przy okazji omawiania tablic omawia się ***typy napisowe***, które są implementowane jako ***tablice znaków***. W Java jest inaczej.

Typ **String** ***nie jest typem prostym*** lecz klasą, której instancje (obiekty) dopiero implementują napisy.

Omawiając tablice, równie często jak typy napisowe, omawia się związane z nimi ***typy wskaźnikowe***. I tu zaskoczenie:

Java nie wspiera ani nie pozwala na używanie wskaźników.

Powodem braku wskaźników jest bezpieczeństwo. Gdyby zezwolić na ich używanie program lub applet Javy mógłby z łatwością przekroczyć barierę dzielącą środowisko uruchomieniowe Javy (JVM) i stać się niebezpieczny dla systemu operacyjnego.

Brak wskaźników i **implementacja napisów jako obiektów** to główne powody (choć nie jedyne) ***poprawy bezpieczeństawa*** w stosunku do C czy C++.

Formatowanie wyjścia

Klasa `NumberFormat` z pakietu `java.text` posiada metody formatowania:

Liczb
`getNumberInstance()`

Wartości walutowych
`getCurrencyInstance()`

Wartości procentowych
`getPercentInstance()`

Każda z powyższych metod występuje w również z parametrem Locale.NAZWA. Np.:

```
getNumberInstance(Locale.US)
```

Zobacz: NumFormat.java

Wielkie liczby

Jeżeli nie wystarczają nam zakresy dostarczane przez typy proste możemy skorzystać z klas pakietu `java.math`:

```
BigInteger
BigDecimal
```

Przykład:

```
BigInteger a = BigInteger.valueOf(10000);
```

Zobacz: TestWielkichLiczb.java
Epsilon.java