Java jest wysokopoziomowym, kompilowanym, obiektowym językiem programowania z silną kontrolą typów. Oznacza to, że każda zmienna musi zostać zadeklarowana przed użyciem.

1 Typy danych

Zmienne w javie dzielą się na trzy rodzaje typów

1.1 typy proste - prymitywy

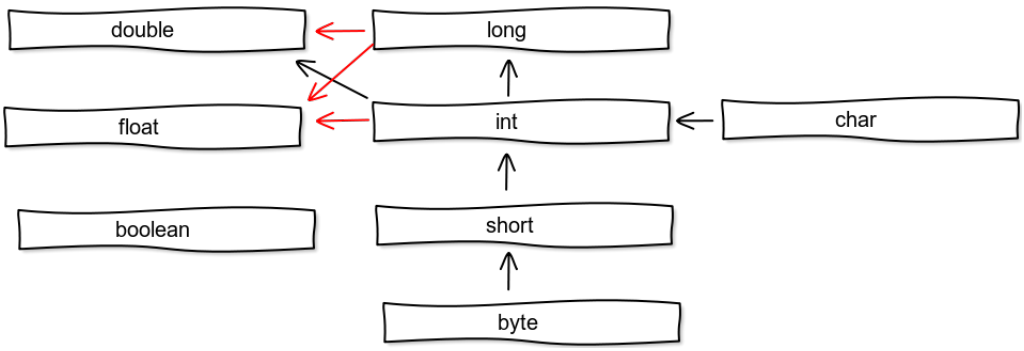
Typy prymitywne służa do reprezentacji danych, nie będących obiektami[1]. Są one wbudowane w język. Ważną cechą jest, że ich wielkość (rozmiar) nie zależy od konkretenej implementacji maszyny wirtualnej. Wyróżniamy osiem typów prostych, które można dodatkowo pogrupować ze względu na *charakter*: typy całkowiete, zmiennoprzecinkowe, znakowe oraz logiczne.

typ wbudowany	**rozmiar w bitach**	**zakres**
typy logiczne		
boolean	8	true / false
typy całkowite		
byte	8	-128 do 127
short	16	-2^{15} do $2^{15}-1$
int	32	-2^{31} do $2^{31}-1$
long	64	-2^{63} do $2^{62}-1$
typy znakowe		
char	16	0 do $2^{16}-1$
typy zmiennoprzecinkowe		
float	32	$1.4e^{-45}$ do $3.4028235e^{38}$
double	64	$4.9e^{-324}$ do $1.797e^{308}$

1.2 Konwersja typów numerycznych

W przypadku konwersji typów należy rozpatrzeć dwa przypadki - niejawne - nie wymagające użycia operatora rzutowania oraz wymagające jawnego rzutownania. Rzutowanie automatyczne zachodzi w przypadku przypisywania typu od mniejszej „dokładności" do typu o większej (na diagramie są możliwe operacje, które obrazują czarne strzałki). Przykładowe konwersje, które nie wymagają jawnego rzutowania, oraz nie powodują utraty danych.

[1] istnieją również typy obiektowe reprezentujące typy proste



Rysunek 1: Hierarchia typów

```
int x = 3;
long y = x;
double z = 3;
```

Możliwe jest też niejawne rzutowania, które powoduje utradę danych

```
int x = 123456789;

float f = n; // f ma wartosc 1.234567892E8
```

W przykładzie powyżej kompilator nie zgłosi żadnego błędu.

W przypadku „poruszanie się" po hierarchii typów w kierunku niezgodnym z kierunkiem strzałek, wymagane jest jawne rzutowanie. Konwesja ta wiąże się z ryzykiem utraty informacji. Przykłady:

```
double x = 9.997;
int nx = (int)x; //nx ma wartosc 9
```

Aby wykonać rzutowanie, należy przez nazwą rzutowanej zmiennej postawić nazwę typu docelowego w okrągłych nawiasach.

1.3 typy obiektowe

1.4 typy tablicowe

Tablica jest rodzajem struktury danych będąca zestawieniem elementów tego samego typu. Dostęp do każdego z tych elementów można uzyskać za pomocą indeksu w postaci liczby typu int. Przykładowo, jeżeli *a* jest tablicą liczb całkowitych, to *a[i]* jest *i-tym* elementem tej tablicy.

Deklaracja zmiennej tablicowej polega na określeniu typu tablicy (czyli podaniu typu elementów i nawiasów kwartatowych *[]*) i nazwy zmiennej

```
int [] a;
```

Powyższa instrukcja tylko deklaruje zmienną *a*. Nie inicjuje jej jednak tablicą (użycie tablicy, przypisanie lub pobranie jakiejś wartości, spowoduje błąd). Do utworzenia tablicy potrzebny jest operator *new*

```
int [] a = new int[100];
```

Powyższa instrukcja tworzy tablicę, w której można zapisać 100 elementów typy *int*.
Tablice indeksowane są od zera. Oznacza to, że pierwszy element w tablicy znajduje się pod indeksem 0, natomiast ostatni po indeksem równym *długości tablicy - 1*

2 Przepływ sterowania

2.1 Instrukcje warunkowe

Instrukcje warunkowe służą do podejmowania decyzji, czy dany kod ma być wykonany. Najprostszy warunek ma postać

```
if (warunek_logiczny)
        instrukcja
```

Warunek zawsze musi zwracać (być ewaluowany do) wartości logicznej true / false.

```
int x = 3;
int y = 4;

if (x < y) {
        y = x;
}
```

Po *if* mogą, opcjonalnie, wystąpić bloki wykonujące kod w przypadku gdy nie zostanie spełniony właściwy warunek.

```
if (warunek_logiczny)
        instrukcja1
else
        instrukcja2
```

oraz

```
if (warunek_logiczny)
        instrukcja1
else if (warunek_logiczny)
        instrukcja2
else
        instrukcja3
```

Bloki *if(){ }*, *if() { } else { }*, *if() { } else if() { } else { }* można dowolnie zagnieżdżać. Przykład

```
int x = 10;

if (x > 10) {
        if (x < 20) {
                System.out.println("x <10, 20>");
        } else {
                System.out.println("x > 20");
        }
} else if (x < 10) {
        System.out.println("x < 10");
} else {
        System.out.println("x == 10");
}
```

2.2 Pętle

Pętla *while* wynonuje blok instrukcji do póki zadany warunek ma wartość *true*. Ogólna postać to

```
while (warunek_logiczny) {
        instrukcja
}
```

Jeżeli *warunek_logiczny* będzie fałszywy przed rozpoczęciem wykonywania bloku *while*, instrukcje wewnątrz bloku nie zostaną wykonane. Pętla *while* sprawdza warunek na samym początku działania. W związku z tym jej instrukcje mogą nie zostać wykonane ani razu. Aby mieć pewność, że instrukcje zostaną wkonane co najmniej raz, sprawdzanie warunku trzeba przenieść na sam koniec. Do tego służy pętla *do-while*. Jej składnia jest następująca

```
do {
        instrukcje
} while( warunek_logiczny )
```

Najpierw wykonywany jest blok instrukcji, a następnie sprawdzany jest warunek.
Ostatnią przedstawioną będzie instrukcja wyboru *switch* (jest ona generalizacją instrukcji if). Składania jest następująca

```
switch (warunek_wyboru) {

        case wartosc1:
                instrukcja1
        case wartosc2:
                instrukcja2
        default:
                instrukcja3

}
```

W instrukcji *switch warunek_wyboru* nie jest wyrażeniem logiczny, które ewaluuje do wartości logicznej (true / false). *Warunek_wyboru* powinien być liczą całkowitą (lub char)[2]. Natępnie instrukcje z *case*, który spełnia warunek, są wykonywane kaskadowo w dół.

```
switch (x) {
        case 0:
                System.out.println("0");
                break;
        case 1:
                System.out.println("1");
        case 2:
        case 3:
                System.out.println("2, 3");
                break;
        case 4:
                System.out.println("4");
                break;
        default:
                System.out.println("x > 4");
}
```

W powyższym przykładzie, jeżeli wartość wejściowa będzie równa 0, zostanie wyświetlone '0'. W kolejnej linijce znajduje się instrukcja *break*, która przerywa działanie *switch*. Jeżeli, wartość wejściowa będzie równa 1, zostaną

[2]lub enum

wyświetlone następujące napisy: ’1’, ’2,3’. ’4’ nie zostanie wyświetlone ponieważ, poprzedni blok konczy się instrukcją *break*. Jeżeli wartość wejściowa będzie równa, np. 1000, nie zostanie dopawowany żaden warunek i wykona się instrukcja z *default*