# Objektovo orientované programovanie

(Základy jazyka: premenné, pretypovanie, pole, voliteľný počet argumentov, operátory, príkazy riadenia toku)

2. prednáška (1. časť)

Vladislav Novák
FEI STU v Bratislave
25.9.2014

## Obsah

## Premenné

Java sa vyznačuje silnou typovou kontrolou. To znamená, že všetky premenné je nutné pred použitím deklarovať. Súčasťou deklarácie je uvedenie typu a názvu premennej.

```
//deklarácia
int pocet; //premenná typu int s názvom pocet

//deklarácia s inicializáciou
int pocet = 1; //premenná typu int s názvom pocet
               //a počiatočnou hodnotou 1
```

Typ premennej určuje hodnoty, ktoré môže premenná obsahovať a tiež operácie ktoré možno s premennou vykonávať.

Typy premenných delíme na:
- základné typy
  - sú súčasťou jazyka
  - premenné týchto typov obsahujú priamo hodnotu
- referenčné typy
  - triedy, rozhrania, enumeračné typy, polia
  - premenné týchto typov obsahujú referenciu na hodnotu (objekt)
  - okrem špeciálnych prípadov, hodnoty vytvárame pomocou operátora `new`

Základné typy premenných
`byte` – 8 bitové celé číslo v rozsahu -128 až 127 (vrátane)
`short` – 16 bitové celé číslo v rozsahu -32 786 až 32767 (vrátane)
`int` – 32 bitové celé číslo v rozsahu -2 147 483 648 až 2 147 483 647 (vrátane).
U celočíselných hodnôt je to spravidla východzí typ (ak neexistuje dôvod zvoliť iný typ).
`long` – 64 bitové celé číslo v rozsahu -9 223 372 036 854 775 808 až
9 223 372 036 854 775 807 (vrátane)
`float` – číslo s plávajúcou desatinnou čiarkou s jednoduchou presnosťou. Je definované normou IEEE 754. V pamäti zaberá 32 bitov.
`double` - číslo s plávajúcou desatinnou čiarkou s dvojitou presnosťou. Je definované normou IEEE 754. V pamäti zaberá 64 bitov.
U neceločíselných hodnôt je to spravidla východzí typ.
`boolean` – má iba dve možné hodnoty `true` a `false` (pravda, nepravda). Jeho veľkosť nie je presne definovaná.
`char` – umožňuje uložiť jeden 16 bitový znak v kódovaní Unicode. Minimálna hodnota je 0 (‘\u0000’), maximálna hodnota je 65535 (‘\uffff‘).

Pozn.: Všetky základné číselné typy sú znamienkové.

Pozn.: Jazyk Java má špeciálnu podporu pre znakové reťazce pomocou triedy `java.lang.String`. Tento typ nepatrí medzi základné typy, ale ak v programe uzavriete reťazec medzi dvojité úvodzovky, automaticky sa vytvorí nový objekt triedy `String`. Nemusíme používať operátor `new`. Objekty triedy `String` sú *nemenné* (*immutable*), to znamená že po vytvorení nemožno meniť ich hodnotu.

Pozn.: Ak napr. potrebujeme použiť celé číslo, ale typ `long` nám nestačí, tak použijeme napr. triedu `java.math.BigDecimal`.

Druhy premenných
- členské premenné inštancie
- členské premenné triedy (statické premenné) označujú sa slovom `static`
- lokálne premenné
- parametre

Pomenovanie premenných
- rozlišujú sa malé a veľké písmená
- názov sa môže skladať z písmen a číslic v kódovaní unicode s ľubovoľnou dĺžkou
- názov sa musí začínať písmenom, znakom dolár $, alebo podčiarkovníkom _
- konvencia: názov premennej by mal začínať písmenom
- konvencia pre premenné ktoré nie sú konštantou:
  ak sa názov skladá z jedného slova, tak je sú všetky písmena malé,
  ak sa skladá z viacerých slov, tak sú všetky písmená malé okrem začiatočných písmen druhého a nasledujúcich slov
- konvencia pre konštanty: používajú sa iba veľké písmená, ak sa názov skladá z viacerých slov tak tieto slová sú oddelené podčiarkovníkom
-názov premennej nesmie byť kľúčové slovo (vyhradené slovo)

Počiatočné hodnoty členských premenných inštancii a členských premenných tried
Ak nie je členskej premennej inštancie, alebo triedy priradená počiatočná hodnota, tak jej kompilátor priradí nulovú hodnotu, alebo hodnotu `null`.
Tabuľka počiatočných hodnôt:

| | |
|---|---|
| `byte` | `0` |
| `short` | `0` |
| `int` | `0` |
| `long` | `0L` |
| `flaot` | `0.0f` |
| `double` | `0.0d` |
| `char` | `'\u0000'` |
| `boolean` | `false` |
| `String`, alebo iný objekt | `null` |

Počiatočné hodnoty lokálnych premenných
Neinicializovanej lokálnej premennej kompilátor nepriraďuje počiatočnú hodnotu. Ak lokálnej premennej nebola nastavená počiatočná hodnota v mieste deklarácie, tak je nutné jej ju priradiť pred prvým použitím. Prístup k neinicializovanej lokálnej premennej spôsobí chybu pri kompilácii.

Literály (doslovné hodnoty)
Literál reprezentuje pevnú hodnotu v zdrojovom kóde. Literál nie je nutné vypočítavať.
Literál možno priradiť premennej základného typu (nepoužíva sa `new`).

Celočíselné typy (`byte`, `short`, `int`, `long`) môžu byť vyjadrené v dvojkovej, osmičkovej, desiatkovej, alebo šestnástkovej sústave.
Pri zadaní čísla v dvojkovej sústave treba použiť prefix `0b` (nula b).
Pri zadaní čísla v osmičkovej sústave treba použiť prefix `0` (nula).
Pri zadaní čísla v šestnástkovej sústave treba použiť prefix `0x` (nula x)
príklady:

```
int bin = 0b11001; //číslo 25 v dvojkovej sústave
int osm = 031;     // číslo 25 v osmičkovej sústave
int dec = 25;      // číslo 25 v desiatkovej sústave
int hex = 0x19;    // číslo 25 v šestnáskovej sústave
```

Číslo typu `long` sa označuje postfixom `l` alebo `L`. Odporúča sa použiť veľké `L`, pretože malé `l` môže byť ťažko rozlíšiteľné od čísla `1`.
príklady:

```
long a = 123456789O123456789L;
long a = 12345678901234567890; <- chyba (mimo rozsahu typu int)
```

Pri zápise čísiel s plávajúcou desatinnou čiarkou (`float`, `double`) možno použiť postfix:
`f` alebo `F` pre číslo typu float,
`d` alebo `D` pre číslo typu double.
Ak sa neuvedie ani jeden z postfixov `f`, `F`, `d`, `D`, tak typ čísla bude `double`.
príklady:

```
float  d1 = 123.4f; //hodnota typu float
double d2 = 123.4d; //hodnota typu double
double d3 = 123.4;  //hodnota typu double
```

Pre zadanie čísla v exponenciálnom tvare môžeme použiť písmeno `e` alebo `E`
príklady:

```
float  d4 = 1.234e2f; //hodnota 123.4 typu float
double d5 = 1.234e2d; //hodnota 123.4 typu double
double d6 = 1.234e2;  //hodnota 123.4 typu double
```

Pre zlepšenie čitateľnosti kódu môžeme pri zápise čísla používať podčiarkovník.
Podčiarkovník môže byť umiestnený iba medzi číslicami (v 16-kovej sústave aj ABCDEF).
príklady:

```
int a = 12_345_678;
int b = 0x12E0_A487;
double c = 3.141_592_653_589_793;
```

Literáli typu `char` a `String` môžu obsahovať znaky v kódovaní unicode (UTF-16).
Znaky a reťazce možno písať priamo v editore, alebo pomocou kódu (v reťazcoch možno obidva spôsoby kombinovať).
Pri písaní znakov pomocou ich kódu treba použiť prefix `\u`.
Znaky typu `char` sa uzatvárajú do jednoduchých úvodzoviek.
Reťazce typy `String` sa uzatvárajú do dvojitých úvodzoviek.

príklady:

```
char z1 =  'á';
char z2 =  '\u00E1'; //znak dlhé á
String r1 = "Sí Señor";
String r2 = "S\u00ED Se\u00F1or"; //Sí Señor
```

Unicode escape sekvencia môže byť použitá na ľubovoľnom mieste programu, nie len v znakoch a reťazcoch.

Ďalšie špeciálne escape sekvencie ktoré možno použiť pri zadávaní znakov a reťazcov:
`\b` backspace (spätný krok o jeden znak) <- štvorček (netbeans 6.7.1)
`\t` tab (tabulátor)
`\n` line feed (kód odriadkovania)
`\f` form feed (posun strany) <- štvorček (netbeans 6.7.1)
`\r` carriage return (návrat na začiatok riadku)<- bez účinku (netbeans 6.7.1)
`\"` double quote (dvojité úvodzovky)
`\'` single quote (jednoduché úvodzovky)
`\\` backslash (spätná lomka)

Literál `null`

Literál `null` možno priradiť ľubovoľnej premennej, ktorá nie je základného typu. Väčšinou sa používa na označenie, že daný objekt nie je prístupný. S hodnotou `null` nemožno vykonávať žiadne operácie okrem testovania (či daná premenná má hodnotu `null`, alebo nie).

Literál triedy `.class`

Literál triedy možno vytvoriť pripojením „`.class`“ k názvu triedy (napr. `String.class`). Tento literál odkazuje na objekt (typu `Class`), ktorý reprezentuje samotný typ. Získaný objekt umožňuje získať informácie o danom type. Je vytváraný automaticky v JVM.

**Pretypovanie základných číselných typov**

Pretypovanie je zmena typu hodnoty (bez zmeny hodnoty, ak je to možné). Napríklad zmena hodnoty typu `int` na `double` (`12` na `12.0`). Pretypovane je napríklad možné medzi základnými číselnými typmi (`byte`, `short`, `int`, `long`, `float`, `double`). Nie je možne medzi ľubovoľnými typmi. Napríklad nie je možne pretypovať hodnoty typu `boolean` na `int`. Existujú dva druhy pretypovania: implicitné a explicitné.

*Implicitné pretypovanie* je pretypovanie vykonávajúce rozširujúcu konverziu. Pri tejto konverzii typu zväčša nedochádza ku strate informácie (napr. konverzia `int` na `long`, alebo `int` na `double`), pretože množina hodnôt, ktorú dokáže reprezentovať vstupný typ je zväčša podmnožinou hodnôt, ktoré dokáže reprezentovať cieľový typ (neskôr si ukážeme kedy množina hodnôt cieľového typu nie je nadmnožinou hodnôt vstupného typu). Implicitné pretypovanie automaticky zabezpečuje kompilátor. Príklady:

```
int a = 10;
long b = a; //implicitné pretypovanie int na long
double c = b; //implicitné pretypovanie long na double
```

*Explicitné pretypovanie* je pretypovanie vykonávajúce zužujúcu konverziu. Pri tejto konverzii typu môže nastať zmena pôvodnej hodnoty (napr. konverzia `long` na `int`, alebo `double` na `int`), pretože cieľový typ nedokáže reprezentovať všetky hodnoty, ktoré dokáže reprezentovať vstupný typ. Na vykonanie explicitného pretypovania je nutné použiť operátor pretypovania. Ten je tvorený cieľovým typom zapísaným v zátvorkách. Príklady:

```
double a = 10.2;
int b = (int) a; //explicitné pretypovanie
                 //strata presnosti 10.2 -> 10

int c = 0x1234;
byte d = (byte) c; //explicitné pretypovanie
                   //strata hodnoty 0x1234 -> 0x34
```

Implicitné pretypovanie so stratou presnosti

Pri konverzii `int` na `float`, alebo `long` na `double` môže dôjsť ku strate presnosti, ako ukazuje príklad:

```
int a = 1000000032;
float b1 = a; //strata presnosti
double b2 = a;
int c1 = (int) b1;
int c2 = (int) b2;
System.out.println("a="+a);   //1000000032
System.out.println("b1="+b1); //1.0E9
System.out.println("b2="+b2); //1.000000032E9
System.out.println("c1="+c1); //1000000000
System.out.println("c2="+c2); //1000000032
```

**Pole**

- je objekt, ktorý uchováva pevný počet hodnôt rovnakého typu
- dĺžka poľa = počet prvkov poľa, je určená pri vytvorení poľa a nedá sa meniť
- položky poľa = prvky poľa, sa v poli identifikujú pomocou celočíselného indexu
- index prvej položky pola je nula

poznámka: pri prístupe k prvku poľa sa vykonáva automatická kontrola indexu

Deklarácia premennej odkazujúcej na pole
typPrvkov[] názovPola;
Deklarácia nevytvára pole!

príklady:

```
int[] vysledky; //ak nejde o lokálnu premennú,
                //tak to isté ako: int[] vysledky = null;
```

```
int vysledky[]; //funguje, ale podľa konvencie sa nepoužíva,
pretože hranaté zátvorky sú súčasťou definície typu
```

Vytvorenie a inicializácia poľa

tri možnosti:

1. možnosť (príklad):

```
int[] cisla1 = {10,20,30,40,50};
String[] retazce1 = {"autobus", "elektricka", "trolejbus"};
```

2. možnosť (príklad):

```
int[] cisla2 = new int[5];
cisla2[0] = 10;
cisla2[1] = 20;
cisla2[2] = 30;
cisla2[3] = 40;
cisla2[4] = 50;
```

```
String[] retazce2 = new String[3];
retazce2[0] = "autobus";
retazce2[1] = "elektricka";
retazce2[2] = " trolejbus";
```

3. možnosť (príklad):

```
int[] cisla1 = new int[] {10,20,30,40,50};
String[] retazce1 = new String[] {"autobus","elektricka",
                                  "trolejbus"};
```

Pomocou tejto možnosti môžeme naraz vytvoriť a definovať hodnoty prvkov poľa v ľubovoľnom mieste programu.

Dĺžka poľa

Dĺžku poľa môžeme zistiť pomocou vlastnosti (premennej) `length`
príklad:

```
int[] pole = {10, 20, 30, 40, 50};
int dlzka = pole.length;
```

Viacrozmerné polia

Viacrozmerné pole je pole, ktorého prvky sú polia. Preto môžu mať „vnútorné“ polia navzájom rôznu dĺžku.

príklady:

1. možnosť:

```
int[][] cisla3 = { {11,12,13,14,15},
                   {21,22},
                   {31,32,33,34,35,36,37},
                 };

String[][] retazce3 ={{"ret11", "ret12", "ret13", "ret14", "ret15"},
                      {"ret21", "ret22"},
                      {"ret31", "ret32", "ret33", "ret34"},
                     };
```

2. možnosť:

```
int[][] cisla4 = new int[2][3];
cisla4[0][0] = 11;
cisla4[0][1] = 12;
cisla4[0][2] = 13;
cisla4[1][0] = 21;
cisla4[1][1] = 22;
cisla4[1][2] = 23;
```

3. možnosť:

```
int[][] cisla5 = new int[][] { {1,2}, {3,4,5} };
```

**Voliteľný počet argumentov**

Pomocou konštrukcie pomenovanej *varargs* možno metóde predať ľubovoľný počet argumentov. Pri použití tejto konštrukcie sa za typom posledného argumentu uvedie výpustka (tri bodky) nasledovaná názvom parametra.
Táto konštrukcia môže byť použitá v metóde, aj v konštruktore.

príklad:

```
public class VolitelnyPocetArgumentov {
    public static int suma(int ... scitance) {
        int vysledok = 0;
        for( int i=0; i<scitance.length; i++) {
            vysledok += scitance[i];
        }
        return vysledok;
    }

    public static int mocninaSumy(int exponent,
                                  int ... scitance){
        return (int) Math.pow(suma(scitance), exponent);
    }

    public static void main(String[] args){
        int vysledok1 = suma(1,2,3,4,5);
        System.out.println(vysledok1); //vytlaci: 15

        int pole[] = {1,2,3,4,5};
        int vysledok2 = suma(pole);
        System.out.println(vysledok2); //vytlaci: 15

        int vysledok3 = suma();
        System.out.println(vysledok3); //vytlaci: 0

        int vysledok4 = suma(5);
        System.out.println(vysledok4); //vytlaci: 5

        int vysledok5 = mocninaSumy(4,1,2,3,4);
        System.out.println(vysledok5); //vytlaci: 10000
    }
}
```

Vo vnútri metódy sa s parametrom `scitance` pracuje ako s poľom.
Metódu možno volať buď s poľom, alebo sekvenciou parametrov.

V príklade môže byť za formálny parameter `scitance,` dosadený ľubovoľný počet prvkov typu `int` (nula, jeden, alebo viac), alebo pole prvkov typu `int`.

Voliteľný počet argumentov využíva napr. metóda `printf`

```
//vytlaci: formatovany retazec podobne ako v C s cislom 1234
System.out.printf("formatovany %s %s ako v C s cislom %d%n",
                  "retazec", "podobne", 1234);
```

## Operátory

Operátory sú špeciálne symboly, ktoré vykonávajú konkrétne operácie s jedným, dvoma, alebo troma operandmi a potom vracajú výsledok.

Tabuľka operátorov zoradených podľa priority:

| | |
|---|---|
| postfix | výraz++ výraz-- |
| unary | ++výraz --výraz +výraz -výraz ~ ! |
| multiplicative | * / % |
| additive | + - |
| shift | << >> >>> |
| relational | < > <= >= instanceof |
| equality | == != |
| bitwise AND | & |
| bitwise exclusive OR | ^ |
| bitwise inclusive OR | \| |
| logical AND | && |
| logical OR | \|\| |
| ternary | ? : |
| assignment | = += -= *= /= %= &= ^= \|= <<= >>= >>>= |

- operátory na jednom riadku tabuľky majú rovnakú prioritu.
- čim vyššie je operátor, tým ma vyššiu prioritu.
- operátory s vyššou prioritou sú vyhodnocované skôr, ako operátory s nižšou prioritou.
- všetky binárne operátory okrem operátorov priradenia sa vyhodnocujú zľava doprava
- operátory priradenia sa vyhodnocujú sprava doľava.

Druhy operátorov

Operátor jednoduchého priradenia
= operátor jednoduchého priradenia

Aritmetické operátory
+ operátor súčtu
- operátor odčítania
* operátor násobenia
/ operátor delenia
% operátor celočíselného zvyšku po delení

príklad:
12 % 5 // hodnota výrazu je 2

Unárne operátory
+ označuje kladnú hodnotu
- mení znamienko hodnoty
++ zvyšuje hodnotu o 1
(operátor možno použiť ako prefixový, alebo postfixový -> rôzne hodnoty výrazu)
-- znižuje hodnotu o 1
(operátor možno použiť ako prefixový, alebo postfixový -> rôzne hodnoty výrazu)
! operátor logického doplnku, invertuje hodnotu typu `boolean`

int *i* =10;
++*i* //hodnota výrazu je 11, hodnota premennej *i* je 11

int *j* = 10;
*j*++ //hodnota výrazu je 10, hodnota premennej *j* je 11

int *k* =10;
--*k* //hodnota výrazu je 9, hodnota premennej *k* je 9

int *l* = 10;
*l*-- //hodnota výrazu je 10, hodnota premennej *l* je 9

Operátor nad objektmi triedy `String`
+ operátor zreťazenia objektov triedy `String`

```
"aa" + "bb"     //hodnota výrazu "aabb"
```

Operátory rovnosti a relačné operátory
= = rovná sa
!= nerovná sa
> väčší ako
>= väčší, alebo rovný
< menší
<= menší alebo rovný

Podmienkové operátory
- vykonávajú operácie nad dvoma logickými výrazmi
- nemožno aplikovať na čísla
- najprv sa vyhodnotí ľavý operand a pravý operand sa vyhodnotí iba v prípade potreby

&& a zároveň
|| alebo

príklady:
(1 = = 1 ) && ( 2 = = 2 ) // hodnota výrazu je true, vyhodnocujú sa obidve strany
(1 = = 2 ) && ( 2 = = 2 ) // hodnota výrazu je false, vyhodnocuje sa iba ľavá strana
(1 = = 2 ) || ( 3 = = 4 ) // hodnota výrazu je false, vyhodnocujú sa obidve strany
(1 = = 1 ) || ( 1 = = 2 ) // hodnota výrazu je true, vyhodnocujú sa iba ľavá strana
(1 = = 2 ) || ( 1 = = 1 ) // hodnota výrazu je true, vyhodnocujú sa obidve strany

?: termárny operátor – podľa pravdivosti prvého operandu, je hodnotou výrazu druhý, alebo tretí operand

príklad:
true **?** 10 **:** 20 //hodnota výrazu je 10
false **?** 10 **:** 20 //hodnota výrazu je 20
(1= = 1) **?** 10 **:** 20 //hodnota výrazu je 10
(1= = 2) **?** (2*5) **:** (2+5) //hodnota výrazu je 7

Operátor porovnania typu
`instanceof`

príklad:
```
//vyraz ma hodnotu true ak premenna je typu String,
//inak ma vyraz hodnotu false
premenna instanceof String
```

hodnota `null` nie je inštancia žiadnej triedy
```
null instanceof String //hodnotou je false
```

Bitové operátory a operátory bitového posunu
- pre celočíselné typy operandov
- všetky základné číselne typy sú znamienkové

~ bitový doplnok
príklad pre hodnoty typu `byte`:
```
~25 //hodnota výrazu je -26
```
v dvojkovej sústave:
```
~ 0001 1001₂
------------
  1110 0110₂
```

& bitový súčin (AND)
príklad pre hodnoty typu `byte`:
25 & 83 `//hodnota výrazu je 17`
v dvojkovej sústave:
```
  0001 1001₂
& 0101 0011₂
------------
  0001 0001₂
```

| bitový súčet (inkluzívny OR)
príklad pre hodnoty typu `byte`:
```
25 | 83 //hodnota výrazu je 91
```
v dvojkovej sústave:
```
  0001 1001₂
| 0101 0011₂
------------
  0101 1011₂
```

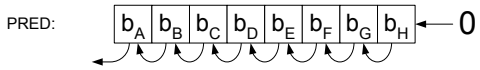
^ bitová neekvivalencia (vyhradný OR, exclusive OR, XOR)
príklad pre hodnoty typu `byte`:

```
25 ^ 83 //hodnota výrazu je 74
```

v dvojkovej sústave:

$$\begin{array}{l} 0001\ 1001_2 \\ \wedge\ 0101\ 0011_2 \\ \hline 0100\ 1010_2 \end{array}$$

<< bitový posun doľava




príklady pre hodnoty typu `byte`:

```
45 << 1 //hodnota výrazu je 90
```

v dvojkovej sústave:

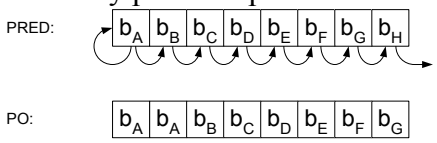
$$\begin{array}{l} 00101101_2 \\ <<\ \ \ \ \ \ 1_{10} \\ \hline 01011010_2 \end{array}$$

```
45 << 2//hodnota výrazu je 180
```

v dvojkovej sústave:

$$\begin{array}{l} 00101101_2 \\ <<\ \ \ \ \ \ 2_{10} \\ \hline 10110100_2 \end{array}$$

>> bitový posun doprava so znamienkom




príklady pre hodnoty typu `byte`:

```
12 >> 1 //hodnota výrazu je 6
```

v dvojkovej sústave:

$$\begin{array}{l} 00001100_2 \\ >>\ \ \ \ \ \ 1_{10} \\ \hline 00000110_2 \end{array}$$

```
12 >> 2//hodnota výrazu je 3
```

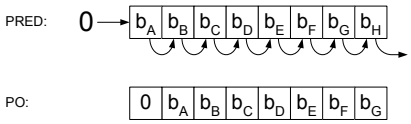
v dvojkovej sústave:

$$\begin{array}{l} 00001100_2 \\ >>\ \ \ \ \ \ 2_{10} \\ \hline 00000011_2 \end{array}$$

```
-13 >> 1 //hodnota výrazu je -7
```
v dvojkovej sústave:
$$11110011_2$$
$$>> \quad 1_{10}$$
$$11111001_2$$

```
-13 >> 2 //hodnota výrazu je -4
```
v dvojkovej sústave:
$$11110011_2$$
$$>> \quad 2_{10}$$
$$11111100_2$$

>>> bitový posun doprava bez znamienka




príklady pre hodnoty typu `byte`:

```
12 >>> 1 //hodnota výrazu je 6
```
v dvojkovej sústave:
$$00001100_2$$
$$>>> \quad 1_{10}$$
$$00000110_2$$

```
12 >>> 2 //hodnota výrazu je 3
```
v dvojkovej sústave:
$$00001100_2$$
$$>>> \quad 2_{10}$$
$$00000011_2$$

```
-13 >>> 1 //hodnota výrazu je 121
```
v dvojkovej sústave:
$$11110011_2$$
$$>>> \quad 1_{10}$$
$$01111001_2$$

```
-13 >>> 2 //hodnota výrazu je 124
```
v dvojkovej sústave:
$$11110011_2$$
$$>>> \quad 2_{10}$$
$$01111100_2$$

Operátory zloženého priradenia
- umožňujú skrátený zápis (menšia pravdepodobnosť chyby)
príklady:

```
a += 5    // to isté ako a = a + 5
a -= 5    // to isté ako a = a - 5
a *= 5    // to isté ako a = a *5
a /= 5    // to isté ako a = a / 5
a %= 5    // to isté ako a = a % 5
a &= 5    // to isté ako a = a & 5
a |= 5    // to isté ako a = a | 5
a ^= 5    // to isté ako a = a ^ 5
a <<= 5   // to isté ako a = a << 5
a >>= 5   // to isté ako a = a >> 5
a >>>= 5 // to isté ako a = a >>> 5
```

**Výrazy**

Výraz je konštrukcia vytvorená z premenných, operátorov a volania metód. Výsledkom vyhodnotenia výrazu je jediná hodnota.

príklady:

```
7+4*5      //hodnota výrazu je 27
7+(4*5)    //hodnota výrazu je 27
(7+4)*5    //hodnota výrazu je 55
```

```
int frekvencia;
frekvencia = 2 * 5; //hodnota výrazu je 10 a je typu int
```

Najprv sa vykoná násobenie (operátor násobenia má vyššiu prioritu, ako operátor priradenia). Potom sa hodnota 10 priradí premennej `frekvencia`. Hodnota celého výrazu je 10 a typu `int`, pretože operátor priradenia(=) vracia hodnotu, ktorá sa priradí jeho ľavému operandu. Ľavý operand tiež udáva typ hodnoty.

**Príkazy**

Príkaz tvorí úplnú jednotku vykonávaného programu. Príkazy sú oddelené bodkočiarkou.
Typy príkazov:
- výrazové príkazy
  napr.:
  ```
  hodnota = 10;
  System.out.println("ahoj");
  Bicykel mojBicykel = new Bicykel();
  ```
- deklaračné príkazy
  napr.:
  ```
  double hodnota = 10;
  ```
- príkazy riadenia toku
  if, if-else, switch, while, do-while, for, break, continue, return

**Bloky**

Blok je skupina žiadneho, alebo viacerých príkazov medzi zloženými zátvorkami.
príklad:

```
if (podmienka) { // zaciatok bloku 1
     int a = 10;
     System.out.println("podmienka splnena");
} //koniec bloku 1
else { //zaciatok bloku 2
     System.out.println("podmienka nesplnena");
} //koniec bloku 2
```

**Príkazy riadenia toku**

Príkazy v zdrojovom súbore sa obvykle vykonávajú zhora nadol za sebou. Príkazy riadenia toku menia poradie vykonávania príkazov na základe aktuálnych podmienok počas vykonávania programu.

Príkaz **if**

`if` (podmienka) príkaz;

príklad:

```
a  = 5;
if ( b < 10 )
     c++
     ;
d++;
```

príklad:

```
a  = 5;
if ( b < 10 ) {
     c++;
     e++;
}
d++;
```

Príkaz **if-else**

`if` ( podmienka ) príkazA ; `else` príkazB ;

príklad:

```
a = 5;
if( b < 10 )
     c++
     ;
else
     d++
;
e++;
```

príklad:

```
a = 5;
if( b < 10 ) {
     c++ ;
}
else {
     d++ ;
}
e++;
```

Príkaz **switch**

Príkaz switch dokáže testovať hodnoty typu `byte`, `short`, `char`, `int`, enumeračné typy a tiež objektov typu `Character`, `Byte`, `Short`, `Integer`, `String`.

```
switch ( testovanaHodnota ) {
      case hodnota1: prikazA; break;
      case hodnota2: prikazB; prikazC; break;
      case hodnota3:
      case hodnota4: prikazD; break;
      default: prikazE; break;
}
```

príklad:

```
int a = 1; //2, 3, 4, 5, 6, 7, 8, 9, 10
switch( a ) {
    case 1:
        System.out.println("A");
        break;
    case 2:
        System.out.println("B");
        System.out.println("C");
        break;
    case 3:
        System.out.println("D");
    case 4:
        System.out.println("E");
        break;
    case 5:
    case 6:
    case 7:
    case 8:
        System.out.println("F");
        break;
    default:
        System.out.println("G");
}
```

výsledky pre rôzne hodnoty premennej `a`:

| a | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| výsledok | A | B<br>C | D<br>E | E | F | F | F | F | G | G |

Príkaz **while**
`while` ( podmienka ) príkaz ;

príklad pre výpis čísiel od 1 do 10:
```
int pocet = 1;
while( pocet <= 10) {
    System.out.println(pocet);
    pocet++;
}
```

Príkaz **do-while**
`do` príkaz `while` ( podmienka ) ;

príklad pre výpis čísiel od 1 do 10:
```
int pocet = 1;
do{
    System.out.println(pocet);
    pocet++;
}while(pocet < 11);
```

Príkaz **for**
dve verzie:
**1)**
`for` ( inicializácia ; podmienkaOpakovania ; vykonaneNaKonciKazdehoCyklu ) príkaz ;

príklad (obidva kódy vykonajú to isté)
```
for( pocet=1; pocet <= 10; pocet++){
    System.out.println(pocet);
}
```
```
pocet=1;
while( pocet<=10 ) {
      System.out.println(pocet);
      pocet++;
}
```

V inicializácii cyklu `for` môžeme deklarovať premennú. Platnosť takejto premennej je od jej deklarácie až po koniec cyklu. Mimo cyklu tato premenná nie je deklarovaná.
príklad:
```
for( int pocet=1; pocet <= 10; pocet++) {
    System.out.println(pocet);
}
pocet = 5; //CHYBA
```

Výrazy v cykle `for` nie sú povinné
príklad:
```
for( ;  ; ) { //nekonecny cyklus
     //príkazy
}
```

**2)** nazýva sa rozšírený príkaz for
`for`( premenná : poleAleboKolekcia ) príkaz ;
- zmenšuje pravdepodobnosť chyby

```
int[] pole = {1,2,3,4,5,6,7,9,10};
for( int polozka: pole ) {
    System.out.println(polozka);
}
```

Príkaz **break**
dve formy
- bez návestia (ukončuje najvnútornejší príkaz switch, for, while, do-while)
- s návestím (ukončuje príkaz pred ktorým je zadané návestie)

Príklad použitia `break` bez návestia (hľadanie pozície prvku v poli):

```
int[] udaje = {10,50,40,15,80};
for(int i=0; i<udaje.length; i++) {
     System.out.println("i = " + i);
     if( udaje[i] == 15 ) {
          System.out.println("index cisla 15 je " + i);
          break;
    }
}
System.out.println("uz sme tu");
```

Príklad použitia `break` s návestím hľadanie pozície prvku v poli):

```
int[][] udaje = { {10, 20}, {11, 22, 33} };
hladanie:
     for(int i=0; i<udaje.length; i++) {
          for(int j=0; j<udaje[i].length; j++) {
             System.out.println("i = "+i+", j = "+j);
             if( udaje[i][j] == 11 ) {
                    System.out.println("index cisla 11 je
                    [" + i + "][" + j + "]");
                    break hladanie;//ukončí sa cyklus pred ktorým je návestie
             }
          }
     }
System.out.println("uz sme tu");
```

Príkaz `break` môže ukončovať len príkaz v rámci ktorého je uvedený
príklad:

```
break navestie; //CHYBA: nedefinované návestie
navestie:
     for( int[] polozka: udaje ) {
          break navestie; // OK
     }
break navestie; //CHYBA: nedefinované návestie
navestie2:
     {
         break navestie2; //OK
     }
```

Príkaz **continue**
Použitie v cykle for, while, do-while
Dve verzie:
- bez návestia (vynechá zvyšok tela najvnútornejšieho cyklu `for`, `while`, alebo `do-while`)
- s návestím (vynechá zvyšok tela cyklu ktoré je označené príslušným návestím)

Príklad použitia `continue` bez navestia (výpočet sumy párny čísiel):

```
int[] udaje = { 4, 7, 5, 2};
int suma = 0;
for( int polozka: udaje) {
     if( polozka % 2 != 0 ) { //ak nie je parne
          continue;
     }
     suma += polozka;
}
System.out.println("suma = " + suma);
```

príklad použitie `continue` s návestím (výpis riadkov (podpolí), ktoré obsahujú iba veľké písmená):

```
char[][] udaje = { {'A', 'B', 'C', 'D' },
                   {'E', 'f', 'G', 'H' },
                   {'i', 'j', 'k', 'l' },
                   { 'M', 'N', 'O', 'P'} };
hladanieRiadkov:
     for( char[] podpole: udaje) {
          for( char pismeno: podpole ) {
               //ak nie velke pismeno
               if( ! Character.isUpperCase(pismeno)) {
                    //vynechá zvyšok cyklu, ktorý je označený návestím
                    continue hladanieRiadkov;
               }
           }
           System.out.println(podpole);
     }
```

Príkaz **return**

Príkaz `return` zaistí opustenie funkcie vráti riadenie toku tam, odkiaľ bola funkcia volaná.
Dve verzie:
- bez návratovej hodnoty
- s návratovou hodnotou

príklady:

```
void funkcia1(int parameter) {
     //.....
     return;
}
```

```
int funkcia2(int parameter) {
     //.....
     return 10;
}
```

Ak sa za `return` nachádzajú príkazy, ktoré sa nikdy nevykonajú (kôli `return`), tak kompilátor vyhlási chybu.