



Príručka Java

Úvod do Javy



IT ACADEMY

Obsah:

I. Inštalácia Javy	3
II. Nastavenie systémovej premennej (Windows).....	5
III. Vytvorenie a spustenie programu v konzole	7
IV. Inštalácia NetBeans	8
V. Vytvorenie programu v NetBeanse	9
VI. Premenné	11
VII. Literály (doslovné hodnoty)	15
VIII. Pole.....	18
IX. Viacrozmerné polia.....	20
X. Voliteľný počet argumentov.....	21
XI. Operátory	23
XII. Výrazy	33
XIII. Príkazy	34
XIV. Bloky.....	35
XV. Príkazy riadenia toku	36
XVI. Deklarácia triedy	46
XVII. Metódy.....	48
XVIII. Preťažovanie metód (overloading methods)	50
XIX. Odporúčaná literatúra a zdroje	52

Túto príručku môžete využiť ako pomôcku pri práci s programovacím jazykom Java. **Príručka podlieha autorským právam a jej vlastníkom je spoločnosť IT Academy s.r.o.**

I. Inštalácia Javy

- prejsť na stránku:
<http://www.oracle.com/technetwork/java/index.html>
- v sekcii Software Downloads zvoliť **Java SE**



Obr. 1 Inštalácia Javy

Na ďalšej stránke v tabuľke Java Platform, Standard Edition zvoliť napr. **Download JDK**.

Java SE Development Kit 8u25

You must accept the [Oracle Binary Code License Agreement for Java SE](#) to download this software.

☒ Accept License Agreement
 ☐ Decline License Agreement

Product / File Description	File Size	Download
Linux x86	135.24 MB	jdk-8u25-linux-i586.rpm
Linux x86	154.88 MB	jdk-8u25-linux-i586.tar.gz
Linux x64	135.6 MB	jdk-8u25-linux-x64.rpm
Linux x64	153.42 MB	jdk-8u25-linux-x64.tar.gz
Mac OS X x64	209.13 MB	jdk-8u25-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	137.01 MB	jdk-8u25-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	97.14 MB	jdk-8u25-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	137.11 MB	jdk-8u25-solaris-x64.tar.Z
Solaris x64	94.24 MB	jdk-8u25-solaris-x64.tar.gz
Windows x86	157.26 MB	jdk-8u25-windows-i586.exe
Windows x64	169.62 MB	jdk-8u25-windows-x64.exe

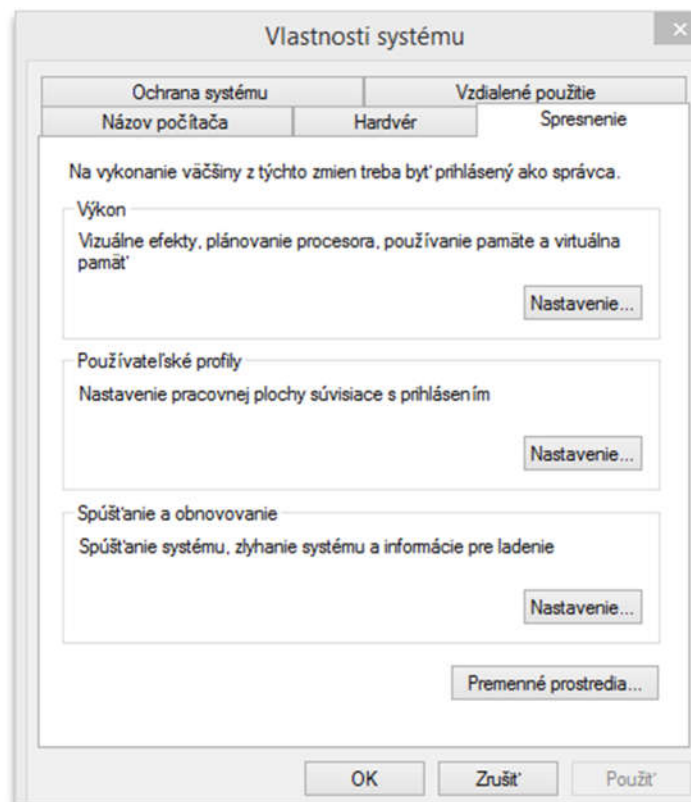
Obr. 2 Java Platform

II. Nastavenie systémovej premennej (Windows)

Nastaviť vo Windows cestu k adresáru obsahujúcemu súbory java, java atď. do premennej Path.

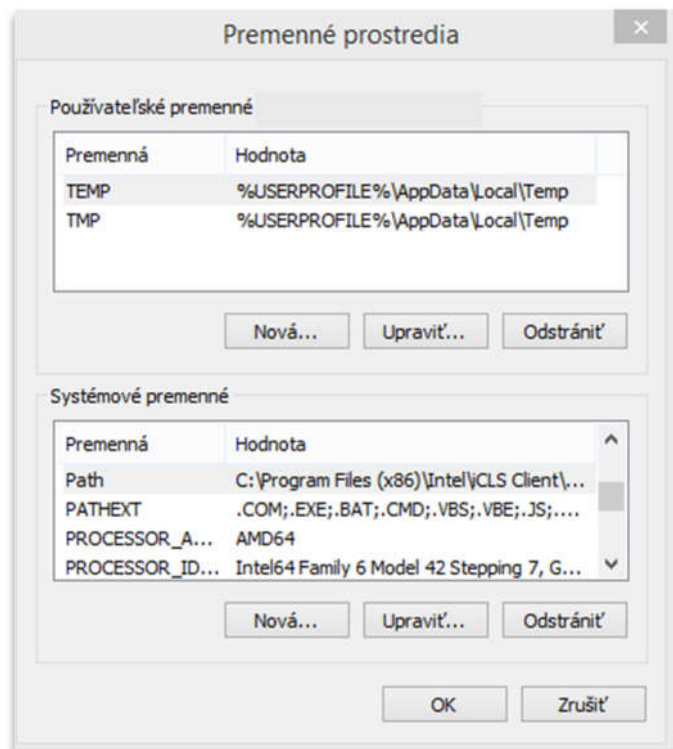
Cesta by mala byť:

- c:\Program Files\Java\jdk1.8.0_25\bin\
- stlačiť klávesy: **Win + pause/break**
- vybrať kartu **Spresenie**
- a potom tlačidlo **Premenné prostredia** („Environment Variables“)



Obr. 3 Vlastnosti systému

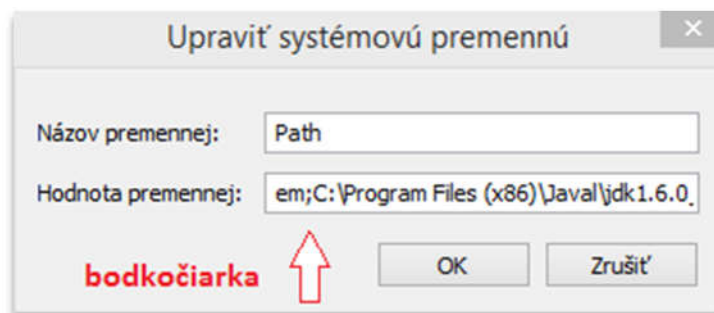
1. Vybrať premennú **Path** (dvojklik)



Obr. 4 Premenné prostredia

2. pridať k nej cestu

napr.: c:\Program Files(x86)\Java\jdk1.8.0_25\bin\



Obr. 5 Úprava systémovej premennej

Programy, ktoré boli spustené pred zmenou premennej **Path**, nerozpoznajú túto zmenu (to platí aj pre procesy, ktoré sú potomkami predtým spustených procesov).

III. Vytvorenie a spustenie programu v konzole

Súbor Prvy.java:

```
class Prvý {  
    public static void main (String[] args) {  
        System.out.println("spustil sa prvy  
program");  
    }  
}
```

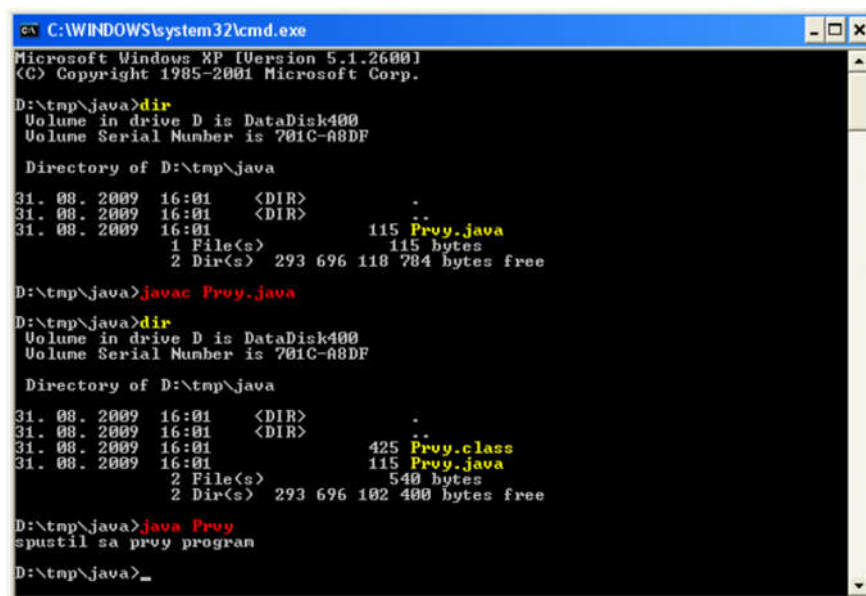
Kompilácia:

javac Prvy.java

Spustenie programu:

java Prvy

(spustí funkciu main, v triede Prvy)



```
C:\WINDOWS\system32\cmd.exe  
Microsoft Windows XP [Version 5.1.2600]  
(C) Copyright 1985-2001 Microsoft Corp.  
  
D:\tmp\java>dir  
Volume in drive D is DataDisk400  
Volume Serial Number is 701C-A8DF  
  
Directory of D:\tmp\java  
  
31. 08. 2009 16:01 <DIR> .  
31. 08. 2009 16:01 <DIR> ..  
31. 08. 2009 16:01 115 Prvy.java  
1 File(s) 115 bytes  
2 Dir(s) 293 696 118 784 bytes free  
  
D:\tmp\java>javac Prvy.java  
  
D:\tmp\java>dir  
Volume in drive D is DataDisk400  
Volume Serial Number is 701C-A8DF  
  
Directory of D:\tmp\java  
  
31. 08. 2009 16:01 <DIR> .  
31. 08. 2009 16:01 <DIR> ..  
31. 08. 2009 16:01 425 Prvy.class  
31. 08. 2009 16:01 115 Prvy.java  
2 File(s) 540 bytes  
2 Dir(s) 293 696 102 400 bytes free  
  
D:\tmp\java>java Prvy  
spustil sa prvy program  
  
D:\tmp\java>
```

Obr. 6 System properties

IV. Inštalácia NetBeans

Pred inštaláciou NetBeans musí byť nainštalovaný **JDK**.

Na stránke <http://www.netbeans.org/downloads/index.html> vybrať verziu, ktorá obsahuje JavaSE download napr. súbor: **netbeans-8.0.2-ml-windows.exe** a spustiť inštaláciu.

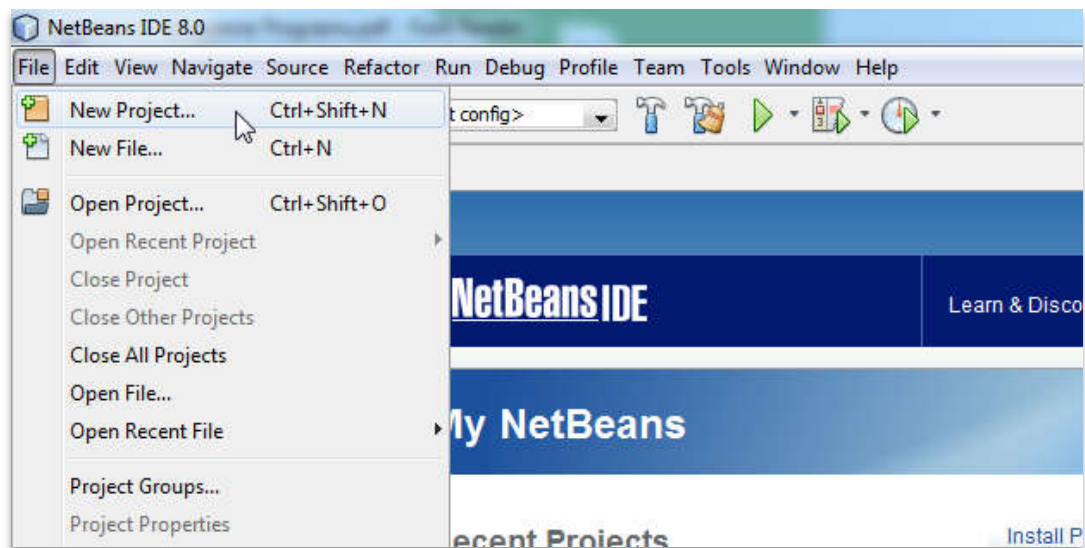


Obr. 7 Inštalácia NetBeans

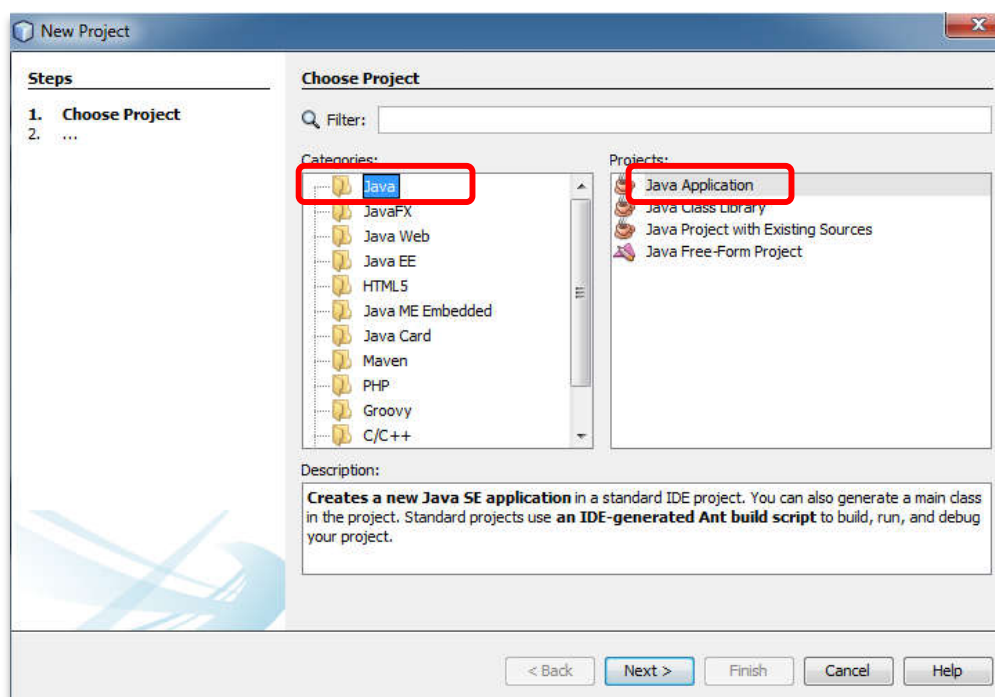
NetBeans IDE Download Bundles					
Supported technologies *	Java SE	Java EE	C/C++	HTML5 & PHP	All
NetBeans Platform SDK	•	•			•
Java SE	•	•			•
Java FX	•	•			•
Java EE		•			•
Java ME					•
HTML5		•		•	•
Java Card™ 3 Connected					•
C/C++			•		•
Groovy					•
PHP				•	•
Bundled servers					
GlassFish Server Open Source Edition 4.1		•			•
Apache Tomcat 8.0.9		•			•
	Download	Download	Download	Download	Download
	Free, 90 MB	Free, 185 MB	Free, 63 MB	Free, 63 MB	Free, 204 MB

Obr. 8 Inštalácia NetBeans

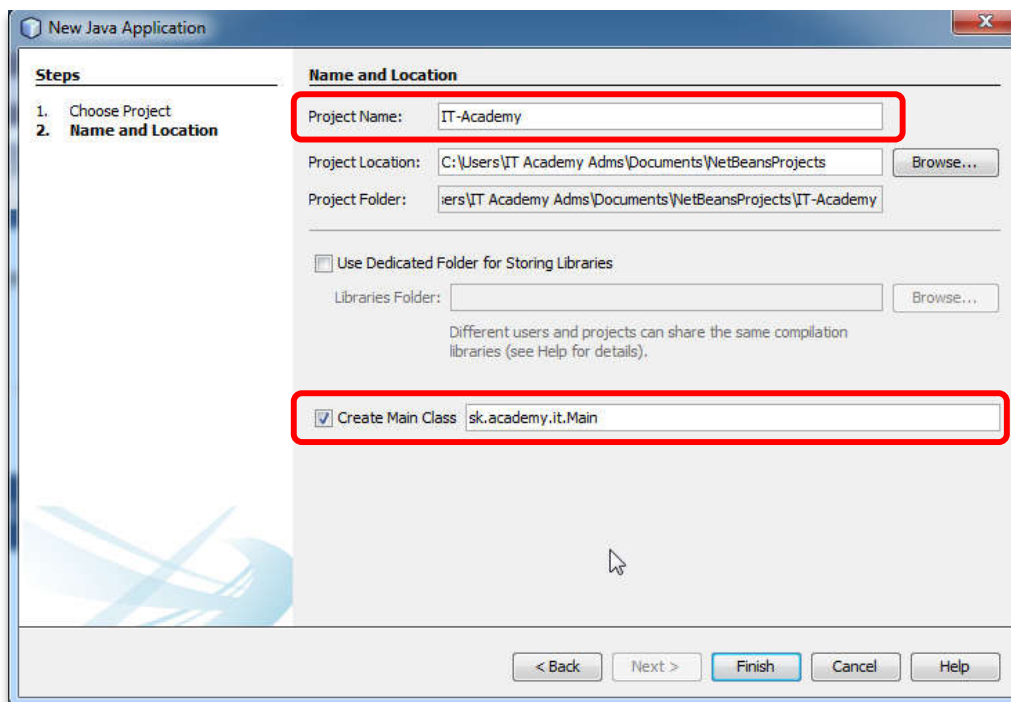
V. Vytvorenie programu v NetBeanse



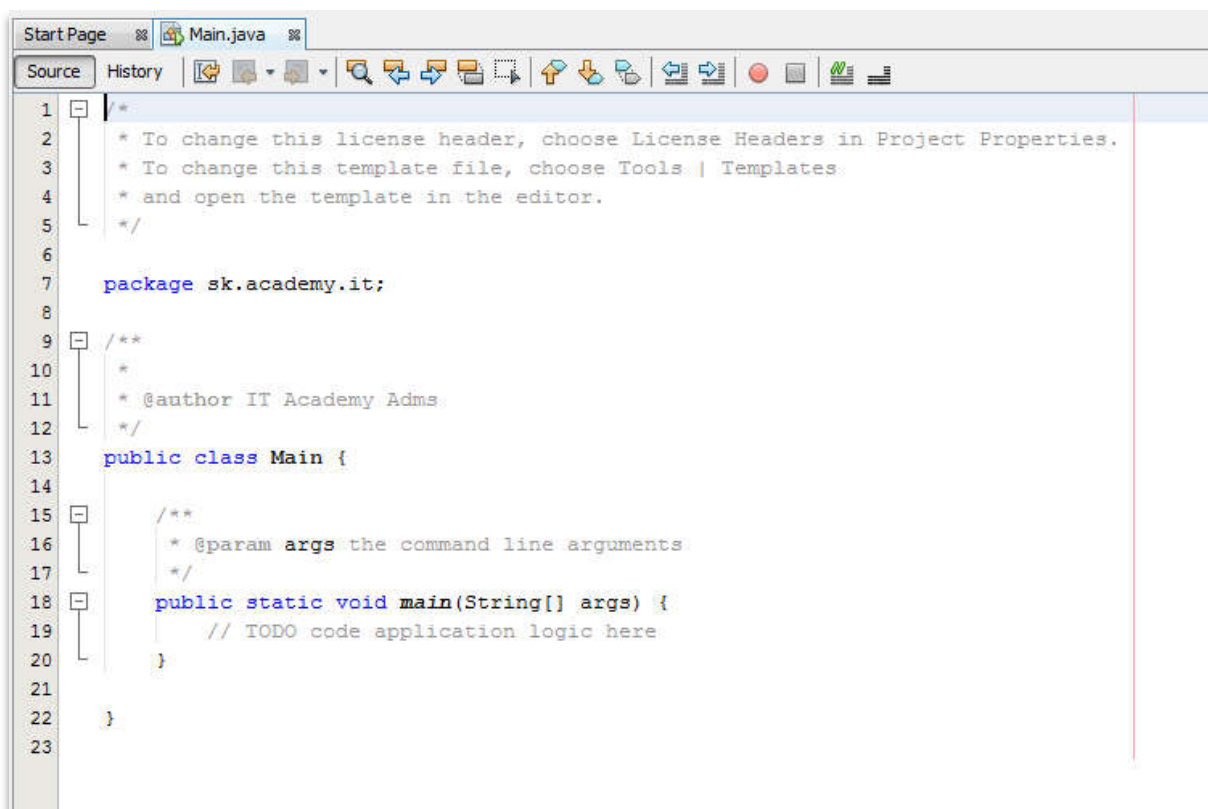
Obr. 9 Vytvorenie nového projektu



Obr. 10 Vytvorenie programu



Obr. 11 Vytvorenie Java projektu



Obr. 12 Kostra zdrojového súboru

VI. 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.

Základné typy premenných:

byte – 8 bitové celé číslo v rozsahu -128 až 127 (vrátane).

short – 16 bitové celé číslo v rozsahu -32 768 až 32 767 (vrátane).

int – 32 bitové celé číslo v rozsahu -2 147 483 648 až 2 147 483 647 (vrátane).

U celočíselných hodnôt je to spravidla počiatočný 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 počiatočný 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').

Poznámka: Všetky základné číselné typy sú znamienkové.

Poznámka: 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**.

Objekty triedy String sú **nemenné** (immutable), to znamená že po vytvorení nemožno meniť ich hodnotu.

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

Druhy premenných

- premenné inštancie
- premenné triedy (statické premenné) sa označujú slovom static
- lokálne premenné
- parametre

Pomenovanie premenných

- rozlišujú sa **malé** a **veľké** premenné
- názov sa môže skladať s **písmen**, **číslic** v kódovaní unicode s ľubovoľnou dĺžkou
- názov sa musí začínať **písmenom**, znakom dolar \$, alebo podtrhovníkom _
- **konvencia**: názov premennej by mal začínať písmenom
- konvencia pre premenné, ktoré nie sú konštantou: ak sa názov skladá z viacerých slov, tak prvé písmena každého slova okrem prvého sú veľké, ostatné sú malé
- prvé slovo je iba **malými** písmenami
- konvencia pre konštanty: používajú sa iba veľké písmená, ak sa názov skladá z viacerých slov tak tieto slová sú oddelené **podtrhovníkom**
- názov premennej **nesmie** byť kľúčové slovo (vyhradene slovo)

Počiatkové hodnoty premenných inštancií a premenných tried

Ak nie je premennej inštancii alebo premennej triede priradená **počiatková** hodnota, tak jej

kompilátor priradí nulovú hodnotu, alebo hodnotu **null**.

Tabuľka počiatkových hodnôt:

byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	„\u0000“
boolean	false
String, alebo iný objekt	null

Počiatkové hodnoty lokálnych premenných

Neinicializovanej lokálnej premennej kompilátor **nepriraduje** počiatkovú hodnotu. Ak lokálnej premennej nebola nastavená počiatková hodnota v mieste deklarácie, tak je nutné jej ju priradiť pred prvým použitím. Prístup k neinicializovanej lokálnej premennej spôsobí chybu pri kompilácii.

VII. 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 osmičkovej, desiatkovej alebo v šestnástkovej sústave.

Pri zadaní čísla v **osmičkovej** sústave treba použiť prefix **0** (nula).

Pri zadaní čísla v **šestnástkovej** sústave treba použiť prefix **0x**.

Príklad:

```
int osm = 031; // číslo 25 v osmičkovej sústave
int dec = 25; // číslo 25 v desiatkovej sústave
int hex = 0x19; // číslo 25 v šestnástkovej sústave
```

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**.

Príklad:

```
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**.

Príklad:

```
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
```

Literály 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 typu **String** sa uzatvárajú do dvojitéch úvodzoviek.

Príklad:

```
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

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**.

VIII. 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: automatická kontrola indexu pri prístupe k prvku poľa.

Deklarácia premennej odkazujúcej na pole

typPrvkov[] názovPola;

Deklarácia nevytvára pole (podobne ako pri premenných iného typu).

Príklad:

```
int[] vysledky; // to iste ako: int[] vysledky =  
null;  
int vysledky[]; // funguje, ale podľa konvencie sa  
nepoužíva,  
pretože hranaté zátvorky sú súčasťou definície typu
```

Vytvorenie a inicializácia poľa

dve možnosti:

1. možnosť:

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

2. možnosť:

```
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";
```

Dĺžka poľa

Dĺžku poľa môžem zistiť pomocou vlastnosti (premennej) **length**.

Príklad:

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

IX. Viacrozmerné polia

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

Príklad:

1. možnosť:

```
int[][] cisl3 = {{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[][] cisl4 = new int[2][3];  
cisl4[0][0] = 11;  
cisl4[0][1] = 12;  
cisl4[0][2] = 13;  
cisl4[1][0] = 21;  
cisl4[1][1] = 22;  
cisl4[1][2] = 23;
```

X. Voliteľný počet argumentov

Pomocou konštrukcie pomenovanej **varargs** možno funkcii 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 obyčajnej funkcii, 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); // vytlačí:
15
    int vysledok3 = suma();
    System.out.println(vysledok3); // vytlačí:
0
    int vysledok4 = suma(5);
    System.out.println(vysledok4); // vytlačí:
5
    int vysledok5 = mocninaSumy(4,1,2,3,4);
    System.out.println(vysledok5); // vytlačí:
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. funkcia **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);
```

XI. Operátory

Operátory sú špeciálne **symbols**, ktoré vykonávajú konkrétne operácie s jedným, dvoma alebo troma operandami 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 majú **rovnakú** prioritu.
- čím **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 zretazenia 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

Príklad:

`(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

Poznámka: 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.

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:

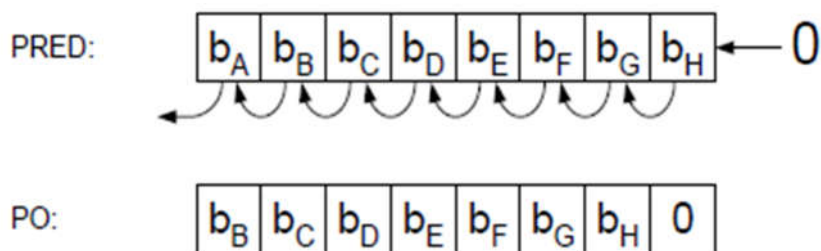
25 ^ 83 // hodnota výrazu je 74

v dvojkovej sústave:

```
0001 10012
^ 0101 00112
-----
```

```
0100 10102
```

<< bitový posun doľava



Obr.13 Bitový posun doľava

Príklad pre hodnoty typu byte:

45 << 1 // hodnota výrazu je 90
v dvojkovej sústave:

00101101₂

<< 1₁₀

01011010₂

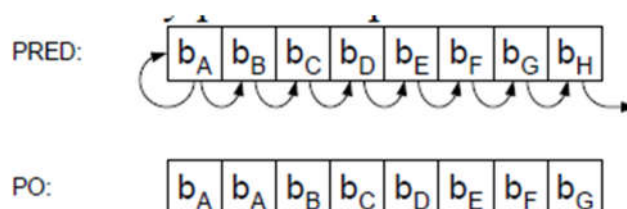
45 << 2 // hodnota výrazu je 180
v dvojkovej sústave:

00101101₂

<< 2₁₀

10110100₂

>> bitový posun doprava so znamienkom



Obr.14 Bitový posun doprava so znamienkom

Príklad pre hodnoty typu byte:

12 >> 1 // hodnota výrazu je 6

v dvojkovej sústave:

000011002

>> 110

000001102

12 >> 2 // hodnota výrazu je 3

v dvojkovej sústave:

00001100₂

>> 2₁₀

00000011₂

11

-13 >> 1 // hodnota výrazu je -7

v dvojkovej sústave:

11110011₂

>> 1₁₀

11111001₂

-13 >> 2 // hodnota výrazu je -4

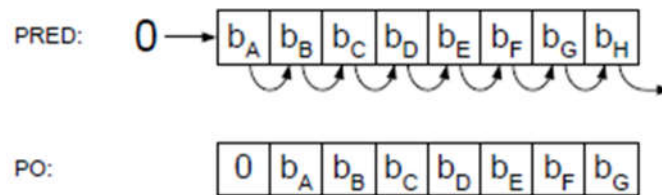
v dvojkovej sústave:

11110011₂

>> 2₁₀

11111100₂

>>> bitový posun doprava bez znamienka



Obr.15 Bitový posun doprava bez znamienka

Príklad pre hodnoty typu byte:

12 >>> 1 // hodnota výrazu je 6

v dvojkovej sústave:

00001100₂

>>> 1₁₀

00000110₂

12 >>> 2 // hodnota výrazu je 3

v dvojkovej sústave:

00001100₂

>>> 2₁₀

00000011₂

-13 >>> 1 // hodnota výrazu je 121

v dvojkovej sústave:

11110011₂

>>> 1₁₀

01111001₂

-13 >>> 2 // hodnota výrazu je 124

v dvojkovej sústave:

11110011₂

>>> 2₁₀

01111100₂

Operátory zloženého priradenia

Umožňujú skrátenejší zápis (menšia pravdepodobnosť chyby).

Príklad:

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

XII. Výrazy

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

Príklad:

$7+4*5$ // 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.

XIII. Príkazy

Príkaz tvorí úplnú jednotku vykonávaného programu. 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

XIV. 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
```

XV. 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**.

```
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("B");  
        break;  
    case 5:  
    case 6:  
    case 7:  
    case 8:
```

```
System.out.println("F");  
break;  
default:  
System.out.println("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 ; podmienka Opakovania ; vykonane Na Konci Cyklu) príkaz ;

Príklad:

```
for (pocet = 1; pocet <= 10; pocet++){  
    System.out.println(pocet);  
}
```

```
pocet=1;  
while (pocet<= 10 ){  
    System.out.println(pocet);  
    pocet++;  
}
```

V inicializácii môžeme deklarovať **premennú**. Platnosť takejto premennej je od jej deklarácie až po koniec cyklu. Mimo cyklu táto premenná nie je deklarovaná (rozdiel oproti C++).

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 ;

- znižuje 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 návěstím** (ukončuje príkaz pred ktorým je zadané návestie)

Príklad použitia break bez návestia

```
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 návěstím:

```
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;
            }
        }
    }
}
```

System.out.println("uz sme tu");

Príkaz **break** môže ukončovať len príkaz v rámci ktorého je uvedený.

Príklad:

```
break navestie; // CHYBA: nedefinované návěstie  
navestie:
```

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

```
break navestie; // CHYBA: nedefinované návěstie  
navestie2:
```

```
{  
    break navestie2; //OK  
}
```

Príkaz continue

Použitie v cykle **for**, **while**, **do-while**.

Dve verzie:

- **bez návestia** (vynechá zvyšok tela najvnútornejšieho cyklu for, while, do-while)
- **s návěstím** (vynechá zvyšok tela cyklu ktoré je označené príslušným návěstím)

Príklad (bez návestia) – 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 (s návěstím) – 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' } };
```

Hľadanie riadkov:

```
for (char[] podpole : udaje) {
    for (char pismeno : podpole){
        // ak nie velke pismeno
        if (!Character.isUpperCase(pismeno)) {
            continue hladanieRiadkov;
        }
    }
    System.out.println(podpole);
}
```

Príkaz return

Príkaz return zaistí opustenie **funkcie** a vráti riadenie toku tam, odkiaľ bola funkcia volaná.

Dve verzie:

- bez návratovej hodnoty
- s návratom hodnoty

Príklad:

```
void funkcia1(int parameter) {  
    // .....  
    return;  
}  
  
int funkcia2(int parameter) {  
    // .....  
    return 10;  
}
```

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

XVI. Deklarácia triedy

minimálna deklarácia triedy:

```
class NazovTriedy {  
    // deklarácia atribútov, konštruktorov a metód  
}
```

atribúty – implementácia pomocou premenných – ukladanie stavu

konšuktory – funkcie pre inicializácia nových objektov

metódy – funkcie implementujúce operácie triedy, alebo jej inštanciami (objektmi)

Názov triedy (konvencia)

- jednoslovné názvy - prvé písmeno je veľké ostatné písmena sú malé
- viacslovné názvy – podobne ako jednoslovné názvy, ale prvé písmeno každého slova je vždy veľké

Príklady:

Bicykel

Auto

HorskyBicykel

OsobnéAuto

Členské premenné

Modifikátor Prístup typ Premennej názov Premennej;

Modifikátor prístupu určuje ktoré triedy majú prístup k danej členskej premennej. Ako typ členskej premennej môže sa môže použiť niektorý zo základných typov (int, double, char, atď), alebo referenčný typ (trieda, rozhranie, pole).

Názov premennej – **identifikácia**.

XVII. Metódy

Definícia metód

modifikátorPrístupu

návratový Typ

názov Funkcie

(typParametra nazovParametra, typParametra2 nazovParametra2)

zoznam výnimiek

```
{  
    // definícia implementácie  
}
```

Signatúra metódy

Signatúra metódy slúži na **identifikáciu** metódy (vzájomné rozlíšenie metód). Signatúra metódy sa skladá z názvu a typov parametrov.

Príklad:

funkcia:

`public double` vypocitaj(`double` sirka, `double` vyska, `int` pocet)

signatúra:

vypocitaj(`double`, `double`, `int`)

Pomenovanie metód (konvencia)

- prvé slovo názvu triedy sa píše **malými** písmenami, ďalšie slová sa začínajú veľkým písmenom a zvyšok je malými písmenami
- prvé slovo názvu je **sloveso**

Príklad:

bez

bezRychlo

získajPozadie

získajVysledneUdaje

jePrazdne

XVIII. Preťažovanie metód (overloading methods)

Preťažovanie metód – vytvorenie viacerých metód s rovnakým názvom, ale odlišným typom alebo počtom vstupných parametrov.

Pri vzájomnom rozlišovaní metód **nerozhoduje** typ návratovej hodnoty, takže nemožno v 1 triede vytvoriť dve funkcie s rovnakou signatúrou, ale rôznymi návratovými hodnotami.

Pri volaní preťaženej metódy sa vhodná verzia metódy vyberie podľa **typu** a **počtu parametrov** uvedených pri jej volaní.

Príklad:

používanie `System.out.println(.....)`

```
class PrintStream {  
    .....  
    public void println() {.....}  
    public void println(boolean x) {.....}  
    public void println(char x) {.....}  
    public void println(int x) {.....}  
    public void println(long x) {.....}  
    public void println(float x) {.....}  
    public void println(double x) {.....}  
    public void println(char[] x) {.....}  
    public void println(String x) {.....}  
    public void println(Object x) {.....}  
    .....  
}
```

volanie metódy:

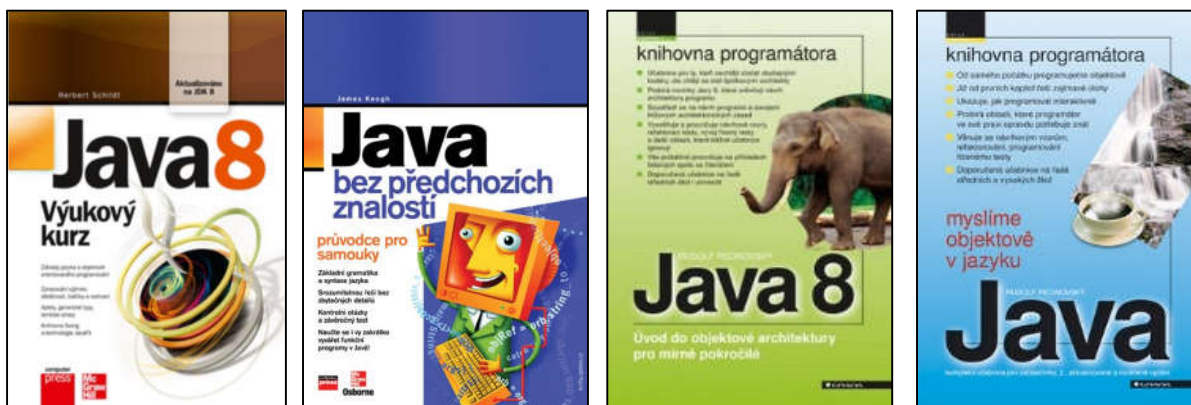
```
int cislo = 10;
String retazec = "ahoj";
System.out.println(cislo); // volanie println(int x)
System.out.println(retazec); // volanie println(String x)
```

Príklad 2: rôzne počty a typy parametrov:

```
public class KresliacePlatno {
    public void vytlac(String retazec) {..... }
    public void vytlac(String retazec, int poziciaX, int poziciaY) { }
    public void vytlac(int cislo) {.....}
    public void vytlac(int cislo, int poziciaX, int poziciaY) {...}
}
```

XIX. Odporúčaná literatúra a zdroje

1. Java 8 - Herbert Schildt
2. Java bez předchozích znalostí - James Keogh
3. Java 8 - Rudolf Pecinovský
4. Myslíme objektově v jazyku Java - Rudolf Pecinovský



5. Mistrovství Java - Herbert Schildt
6. Java 7 - Rudolf Pecinovský
7. 1001 tipů a triků pro jazyk Java - Bogdan Kiszka
8. Učebnice jazyka Java 5 - Pavel Herout



Zaujímavé odkazy:

1. Download JRE, JDK¹
2. Dokumentácia JavaDocs²
3. Vždy aktuálna verzia Java – JavaRa³
4. IDE Netbeans⁴

¹www.oracle.com/technetwork/articles/javase/index-jsp-138363.html

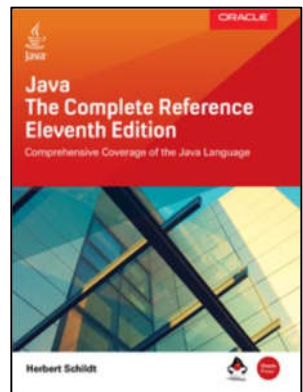
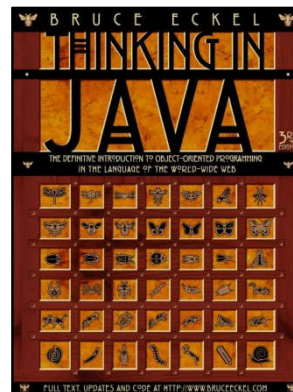
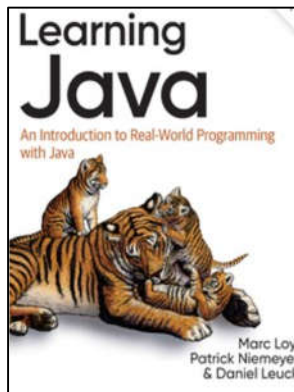
²docs.oracle.com/javase/7/docs/api/

³singularlabs.com/software/javara/

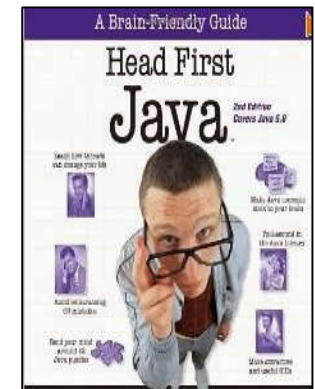
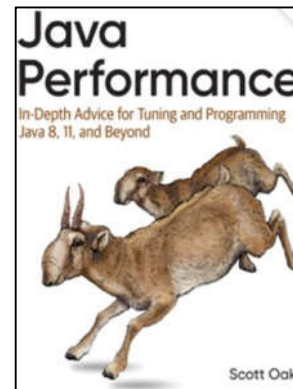
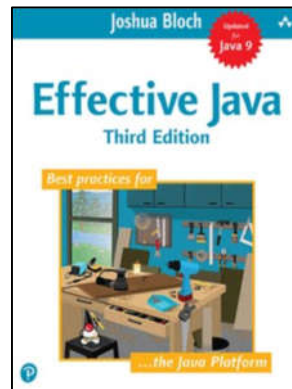
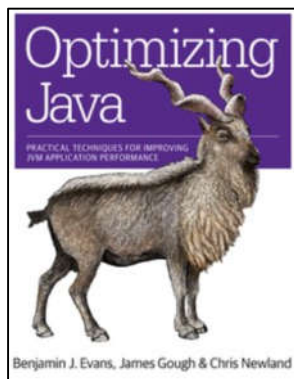
⁴netbeans.org

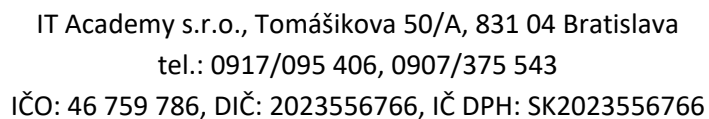
Zahraničná literatúra

1. Learning Java - Marc Loy
2. Java for beginners - Manuj Aggarwal
3. Thinking in Java - Bruce Eckel
4. Java: The Complete Reference - Herbert Schildt



5. Optimizing Java - Benjamin J Evans
6. Effective Java - Joshua Bloch
7. Java Performance - Scott Oaks
8. Head First Java - Kathy Sierra, Bert Bates





IČO: 46 759 786, DIČ: 2023556766, IČ DPH: SK2023556766

