

Výrazy, operátory a řídicí struktury

Jan Faigl

Katedra počítačů
Fakulta elektrotechnická
České vysoké učení technické v Praze

Přednáška 2
A0B36PR1 – Programování 1

Část 2 – Řídicí struktury

Algoritmus a jeho popis

Řídicí struktury

Složený příkaz

Větvení

Cykly

Část 1 – Výrazy a operátory

Číselné typy (připomínka)

Výrazy

Výstup programu (vsuvka)

Přiřazení

Operátory

Matematické funkce

Část I Výrazy a operátory

Výpočet a typy čísel

- Základem výpočtu je práce s čísly
- Čísla mohou být různého typu, liší se **rozsahem a přesností** reprezentace u neceločíselných typů
- Výpočet realizujeme prostřednictvím **výrazů**
- Číselné operace jsou realizovány prostřednictvím **operátorů**
- Java základní číselné typy
 - **int** – 32 bitů (4 bytes)
 - **double** – 64-bit (8 bytes) IEEE 754
- Ostatní Java základní typy

Celočíselné typy

- **byte** – 8 bitů
- **short** – 16 bitů
- **long** – 64-bitů

Neceločíselné typy

- **float** – 32-bit IEEE 754

Logický a znakový typ

- **boolean** – true / false
- **char** – 16-bit Unicode znak

Zápis čísel v Javě

- Přímý zápis hodnoty v programu se nazývá **literál**

Názvosloví?

- Zápis celých čísel je možný i v jiných soustavách

```
int decI = 173; // desítková soustava
int hexI = 0xad; // šestnáctková soustava
int binI = 0b10101101; // dvojková soustava

int sum = 10 + 0xA + 0B10;
```

- Zápis desetinných čísel ve vědeckém formátu
- Příklady zápisu literálů

```
boolean      false      true
char         'a'        '1'        '+'
long         1000000000000L
float        4F         258.52f    1.32e-10f
```

- Pozor na kombinaci typů

```
System.out.println(4.7 - 4.7F);
1.9073486345888568E-7
```

Byte, Bajt a jeho násobky

- Byte (Bajt) – jednotka množství dat v informace
Obvykle nejmenší objem dat, se kterým dokáže procesor pracovat
- Označuje 8 bit, tj. 8-mi ciferné binární číslo
- Rozsah 2^8 hodnot, např. reprezentuje celé číslo od 0 do 255
- V roce 1998 uvedená norma IEC 60027-2 (ČR převzata jako ČSN IEC 60027-2) zavádí nový systém označování násobků.

V souladu se soustavou SI (Le Système International d'Unités)

- | | |
|--|--|
| ■ Kilobajt – kB je 1000 Bajtů (B) | ■ Kibibajt – KiB je 1024 bajtů (B), 2^{10} |
| ■ Megabajt – MB je 1000 kB, 10^6 B | ■ Mebibajt – MiB je 2^{20} B, 1024^2 |
| ■ Gigabajt – GB je 10^9 B | ■ Gibibajt – GiB je 2^{30} B, 1024^3 |
| ■ Terabajt – TB je 10^{12} B, 1000^4 B | ■ Tebibajt – TiB je 2^{40} B |
| ■ Petabajt – PB je 10^{15} B, 1000^5 B | ■ Pebibajt – PiB je 2^{50} B |

Proměnné, literály a pojmenované konstanty

- Číselnou hodnotu uloženou někde v paměti odkazujeme jménem **proměnné**
 - Jméno přidělujeme **deklarací** proměnné, zapisuje se jako příkaz (zakončený středníkem) ve tvaru `typ jméno`;
 - Hodnotu proměnné můžeme specifikovat nebo se nastaví na implicitní hodnotu

```
boolean      false
char         \u0000
int, short, byte / long    0 / 0L
float / double    0.0f / 0.0d
reference     null
```

Pozor, ne všechny jazyky nastavují implicitní hodnotu

- Čísla mohou v programu vystupovat jako **literály**
- Kromě literálů můžeme použít také **pojmenované konstanty**
 - Deklarují se podobně jako inicializované proměnné, ale s klíčovým slovem **final**

Příklady inicializace proměnných a konstant

Proměnné

```
int intValue = 10; //alokace 4 bytu v pameti pro ulozeni ceheho
    cisla a nastaveni hodnoty na 10

double doubleValue; //alokace 8 bytu v pameti pro ulozeni
    desetinného čísla, hodnota je nastavena na 0.0d

int step = 1; //jméno promenne vyjadruje její ucel
int numberOfSteps = 10; //volíme název co nejlépe vystihující
    ucel promenne
```

Konstanty

```
final int MAX = 100; //konstanty zapisujeme jako promenne s
    klicovym slovem final (modifikator typu promene)

MAX = 10; //hodnotu konstanty nelze zmenit, nastane chyba
    prekladu

final int MAX_NUMBER_OF_STEPS = 100; //jména píšeme velkými
    písmeny a slova spojujeme podtržítkem
```

Proměnné, konstanty a kódovací konvence

■ Proměnné

- Podstatná jména
- Malá písmena
- Volíme co nejlépe vystihující účel proměnné
- V případě více slov píšeme dohromady a další slova „zvýrazňujeme“ velkým písmenem a v případě více slov píšeme dohromady

■ Konstanty

- Zapisujeme velkými znaky
- Slova spojujeme podtržítkem

<http://docs.oracle.com/javase/tutorial/java/nutsandbolts/variables.html>

Příklad

```
int stepCounter;
final int MAX_NUMER_OF_STEPS = 5;
```

Kódovací konvence

- Kódovací konvence je soubor pravidel jak psát čitelné a dobře pochopitelné zdrojové kódy
- Vymyslet odpovídající jméno je velmi **těžké**
Pokud vymyslíte lepší, nebojte se jej změnit.
- Dobré jméno **odhaluje autorův záměr**
- Dlouhé jméno je lepší než dlouhý komentář
- Pojmenované konstanty jsou přehlednější než magická čísla uprostřed programu

<http://www.iwombat.com/standards/JavaStyleGuide.html>

<http://www.oracle.com/technetwork/java/codeconvtoc-136057.html>

Výrazy

- **Výraz** předepisuje výpočet hodnoty určitého vstupu
- Struktura výrazu obsahuje *operandy*, *operátory* a *závorky*
- Výraz může obsahovat
 - literály
 - unární a binární operátory
 - proměnné
 - volání funkcí
 - konstanty
 - závorky
- Pořadí operací předepsaných výrazem je dáno **prioritou** a **asociativitou** operátorů.

Příklad

```
10 + x * y    poradi vyhodnoceni 10 + (x * y)
10 + x + y    poradi vyhodnoceni (10 + x) + y
```

* má vyšší prioritu než +
+ je asociativní zleva

Základní výstup programu

- Každý znakově orientovaný program komunikuje s uživatelem nebo ostatními částmi operačního systému prostřednictvím vstupního a dvou výstupních „znakových komunikačních kanálů“
 - Standardní vstup
Vstup uživatele z klávesnice nebo výstup jiného programu
 - Standardní výstup
Zpravidla tisk na obrazovku nebo do souboru
 - Standardní chybový výstup
Slouží k rozlišení informativních výpisů od očekávaných výstupů programu
- V Javě lze výstup programu tisknout prostřednictvím systémové knihovny System

Příklad

```
System.out.println("Print to the standard system output");
System.err.println("Print to the system error output");
```

Formátovaný výstup programu

- Číselný výstup lze formátovat příkazem printf


```
System.out.printf("%+.1f %n", 12.345);
System.out.printf("%+8.2f %n", -12.345);
System.out.printf("%+05d %n", 12);
```

```
javac Printf.java && java Printf
+12.3
-12.35
+0012
```
- Specifikace formátu
`%[index_parametru$][modifikátor][šířka][.přesnost]konverze`
 - **konverze** - číselná soustava pro celé číslo (dekadická, oktalová, šestnáctková) nebo zápis desetinného čísla [d|o|x|f|e|E]
 - **přesnost** - počet desetinných míst
 - **šířka** - počet sázených míst, zarovnání vpravo
 - **modifikátor** – tisk znaménka '+', zarovnání vlevo '-', doplnění nulami '0' na požadovanou šířku

<http://docs.oracle.com/javase/8/docs/api/java/util/Formatter.html>

Příklady výstupu

```
System.out.print("1. Print without end of line");
System.out.print(" allows to concat output\n");
System.out.println("2. We can use standard end-of-line");
System.err.println("ERR: Print to standard error output");
```

```
javac Print.java
java Print
1. Print without end of line allows to concat output
2. We can use standard end-of-line
ERR: Print to standard error output
```

```
java Print 2>/dev/null
1. Print without end of line allows to concat output
2. We can use standard end-of-line
```

```
java Print 1>out
ERR: Print to standard error output
```

```
java Print 1>out 2>err
```

Přřazení

- Nastavení hodnoty proměnné, tj. uložení definované hodnoty na místo v paměti, kterou proměnná reprezentuje
- Tvar přiřazovacího operátoru

$$\langle \text{proměnná} \rangle = \langle \text{výraz} \rangle$$

Výraz je literál, proměnná, volání funkce, ...
- Java je silně typovaný (a také staticky typovaný) jazyk
 - Kombinace typů nejsou povoleny
Java je typově bezpečný jazyk
 - Proměnné lze přiřadit hodnotu výrazu pouze identického typu
Jinak je nutné provést typovou konverzi

```
boolean b = 1;
int i = 1.4;
double d = true;
```

Zkrácený zápis přřazení

- Zápis

$\langle \text{proměnná} \rangle = \langle \text{proměnná} \rangle \langle \text{operátor} \rangle \langle \text{výraz} \rangle$

- Lze zapsat zkráceně

$\langle \text{proměnná} \rangle \langle \text{operátor} \rangle = \langle \text{výraz} \rangle$

Příklad

<code>int i = 10;</code>	<code>int i = 10;</code>
<code>int j = 12.6;</code>	<code>int j = 12.6;</code>
<code>i = i + 1;</code>	<code>i += 1;</code>
<code>j = j / 0.2;</code>	<code>j /= 0.2;</code>

- Přřazení je výraz

`int x, y;`

`x = 6;`
`y = x = x + 6;`

„syntactic sugar“

Základní rozdělení operátorů

- Operátory jsou vyhrazené znaky (nebo posloupnost znaků) pro zápis výrazů
- Můžeme rozlišit čtyři základní typy binárních operátorů
 - Aritmetické operátory – sčítání, odčítání, násobení, dělení
 - Relační operátory – porovnání hodnot (menší, větší, ...)
 - Logické operátory – logický součet a součin
 - Operátor přřazení - na levé straně operátoru `=` je proměnná
- Unární operátory
 - indikující kladnou/zápornou hodnotu: `+` a `-`
operátor – modifikuje znaménko výrazu za ním
 - modifikující proměnou: `++` a `--`
 - logický operátor doplněk: `!`
- Ternární operátor – podmíněně přřazení hodnoty

<http://docs.oracle.com/javase/tutorial/java/nutsandbolts/op1.html>

Výraz a přřkaz

- Přřkaz provádí akci a je zakončen středníkem

```
robotHeading = -10;
robotHeading = Math.abs(robotHeading);
System.out.println("Robot heading: " + robotHeading);
```

- Výraz má určený typ a hodnotu

<code>23</code>	typ <code>int</code> , hodnota 23
<code>14+16/2</code>	typ <code>int</code> , hodnota 22
<code>y=8</code>	typ <code>int</code> , hodnota 8

- Přřazení je výraz a jeho hodnotou je hodnota přřazená levé straně

- Z výrazu se stává přřkaz, je-li ukončen středníkem

Aritmetické operátory 1/2

- Pro operandy typu `int` a `double` jsou definovány operátory
 - unární operátor změna znaménka `-`
 - binární sčítání `+` a odčítání `-`
 - binární násobení `*` a dělení `/`
 - binární zbytek po dělení `%`
- Jsou-li oba operandy stejného typu je výsledek aritmetické operace stejného typu
- Je-li jeden operand typu `int` převede se implicitní konverzí na hodnotu typu `double` a výsledek operace je hodnota typu `double`
- Dělení operandů typu `int` je celá část podílu
Např. 7/3 je 2 a -7/3 je -2
- Pro zbytek po dělení platí $x \% y = x - (x/y) * y$
Např. 7 % 3 je 1 -7 % 3 je -1 7 % -3 je 1 -7 % -3 je -1

Aritmetické operátory 2/2

- Zbytek po dělení platí obdobně i pro typ **double**

Např. $3.8 \% 1.6$ je 0.6

- Unární operátory **++** a **--** mění hodnotu svého operandu
Operand musí být l-hodnota, tj. výraz, který má adresu kde je uložena hodnota výrazu (např. proměnná)
 - Lze zapsat prefixově např. **++x** nebo **--x**
 - nebo postfixově např. **x++** nebo **x--**
 - v obou případech se však **liši výsledná hodnota výrazu!**

<code>int i; int a;</code>	hodnota i	hodnota a
<code>i = 1; a = 9;</code>	1	9
<code>a = i++;</code>	2	1
<code>a = ++i;</code>	3	3
<code>a = ++(i++);</code>	nelze, hodnota i++ není l-výrazem	

Logické operátory

- Logické operátory jsou definovány pro hodnoty typu **boolean**
 - unární operátor negace **!**
 - binární operátor logického součinu **&&**
 - binární operátor logického součtu **||**

x	y	!x	x && y	x y
false	false	true	false	false
false	true	true	false	true
true	false	false	false	true
true	true	false	true	true

- Negace má stejnou prioritu jako změna znaménka
- Logický součin má nižší prioritu než relační operátory
- Operace **&&** a **||** se vyhodnocují zkráceným způsobem, tj. druhý operand se nevyhodnocuje, lze-li výsledek určit již pouze z hodnoty prvního operandu

Relační operátory

- Hodnoty všech základní typů jsou uspořádané a lze je porovnávat relačními operátory
- Priorita vyhodnocení je menší než priorita aritmetických operátorů
- Výsledek relační operace je hodnota typu **boolean**
 - true** – relace označená operátorem platí
 - false** – v opačném případě
- Relační operátory
 - >** – větší
 - >=** – větší nebo rovno
 - ==** – rovná se
 - <** – menší
 - <=** – menší nebo rovno
 - !=** – nerovná se

Asociativita priorit operátorů

- Binární operace op na množině **S** je **asociativní**, jestliže platí $(x \text{ op } y) \text{ op } z = x \text{ op } (y \text{ op } z)$, pro každé $x, y, z \in S$
- U **neasociativních operací** je nutné řešit v jakém pořadí jsou operace implicitně provedeny
 - asociativní zleva – operace jsou seskupeny zleva
 Např. výraz $10 - 5 - 3$ je vyhodnocen jako $(10 - 5) - 3$
 - asociativní zprava – operace jsou seskupeny zprava
 Např. $3 + 5^2$ je 28 nebo $3 \cdot 5^2$ je 75 vs. $(3 \cdot 5)^2$ je 225
- Přřazení je asociativní zprava
 Např. $y=y+8$
 Vyhodnotí se nejdříve celá pravá strana operátoru **=**, která se následně přřadí do proměnné na straně levé.
- Priorita binárních operací vyjadřuje v algebře pořadí, v jakém jsou binární operace prováděny
- Pořadí provedení operací lze definovat důsledným **závorkováním**

Přehled operátorů a jejich priorit 1/2

Priorita	Operátor	Typ operandu	Asociativita	Operace
1	()	jméno	L	volání metody
2	++	aritmetický	P/L	pre/post inkrementace
	--	aritmetický	P/L	pre/post dekrementace
	- +	aritmetický	P	unární minus (plus)
	!	logický	P	logická negace
	()	výraz	P	přetypování
3	*,/,%	aritmetický	L	násobení, dělení, zbytek
4	-	aritmetický	L	odečítání
	+	aritmetický	L	sčítání
		řetězový	L	zřetězení
6	<, >, <=, >=	aritmetický	L	porovnání

Informativní

Matematické funkce

- Základní matematické funkce poskytuje v Javě třída `Math`
- Příklad poskytovaných funkcí
 - Goniometrické funkce: *sin*, *cos*, *tan*, *acos*, *atan2*, ...
 - Logaritmické funkce: *log*, *log10*
 - Mocnina, odmocnina: *sqrt*, *pow*
 - Minima, maxima, absolutní hodnoty: *min*, *max*, *abs*
 - Zaokrouhlovací funkce: *rint*, *ceil*, *floor*, *round*
 - Generování pseudo-náhodných čísel: *random*
- Třída poskytuje také konstanty:
 - `Math.PI`
 - `Math.E`

<http://docs.oracle.com/javase/8/docs/api/java/lang/Math.html>

Přehled operátorů a jejich priorit 2/2

Priorita	Operátor	Typ operandu	Asociativita	Operace
7	==, !=	primitivní	L	rovno, nerovno
11	&&	logický	L	logické AND
12		logický	L	logické OR
14	=, +=, -=			
	*=, /=, %=	libovolný	P	přřazení
	&=, =			

Informativní

Knihovna funkcí – třída `Math`

```
double value = 10.3;
double dx = 1.0;
double dy = 1.0;

System.out.println("Ceil of " + value + " is " + Math.ceil(value));
System.out.println("Floor of " + value + " is " + Math.floor(value));

System.out.println("\nMath.PI: " + Math.PI);
System.out.println("Rounded PI: " + Math.round(Math.PI));
System.out.printf("Formatted print with %4.2f: %4.2f\n", Math.PI);
System.out.printf("Formatted print with %6.4f: %6.4f\n", Math.PI);

System.out.println("\nMath.E: " + Math.E + "\n");

double angle1 = Math.atan2(dy, dx);
double angle2 = Math.atan2(-dy, dx);
System.out.printf("Angle on positive side %4.2f\n", Math.toDegrees(angle1));
System.out.printf("Angle on negative side %4.2f\n", Math.toDegrees(angle2));
```

Typ double – nekonečno a nečíslo

- V rozsahu reprezentace typu double jsou vyhrazeny hodnoty pro nekonečno a pro nedefinovanou hodnotu.

```
double infinity = 5.0 / 0.0;
double max = Double.MAX_VALUE;
double maxTwoTimes = max * 2;
double notDefined = 0.0 / 0.0;

/* tisk hodnot promenných na obrazovku */
if (Double.isInfinite(maxTwoTimes)) {
    System.out.println("Value of the ... ")
}
if (Double.isNaN(notDefined)) {
    System.out.println("Value of the ... ")
}

javac Infinity.java && java

value: 5.0
infinity: Infinity
max: 1.7976931348623157E308
2*max: Infinity
notDefined: NaN

Value of the variable maxTwoTimes is infinity
Value of the variable notDefined is not a number
```

Program jako algoritmus

- Program je implementací (realizací) algoritmu
- Algoritmus je posloupnost kroků vedoucí k řešení určité třídy úloh, je to syntetický postup řešení obecných úloh
- Vlastnosti algoritmu:
 - **hromadnost** a **univerzálnost** – řešení třídy úloh
Měnitelná vstupní data
 - **determinovanost** – každý krok jednoznačně definován
 - **konečnost** – pro přípustná data v konečné době skončí
 - **rezultativnost** – vždy vrátí výsledek (třeba chybu)
 - **korektnost** – výsledek je správný
 - **opakovatelnost** – stejný vstup vede na stejný výstup
- Prostředky pro zápis algoritmu
 - Přirozený jazyk
 - Vývojové diagramy
 - Strukturogramy, pseudojazyk, programovací jazyk

Část II

Řídicí struktury

Příklad úlohy – slovní popis

- Úloha:

Najděte největšího společného dělitele čísel 6 a 15.

Co platí pro společného dělitele čísel?
- Řešení

Návrh postupu řešení pro dvě libovolná přirozená čísla

Definice vstupu a výstupu algoritmu

 - Označme čísla x a y
 - Vyberme menší z nich a označme jej d
 - Je-li d společným dělitelem x a y končíme
 - Není-li d společným dělitelem pak zmenšíme d o 1 a opakujeme test až d bude společným dělitelem x a y
- Symboly x , y a d reprezentují **proměnné** (paměťové místo), ve kterých jsou uloženy hodnoty, které se v průběhu výpočtu mohou měnit.

Příklad algoritmu – slovní popis

■ Úloha:

Najít největší společný dělitel přirozených čísel x a y .

■ Popis řešení

- **Vstup:** dvě přirozená čísla x a y
- **Výstup:** přirozené číslo d – největší společný dělitel x a y
- **Postup**
 1. Je-li $x < y$, pak d má hodnotu x , jinak má d hodnotu y
 2. Pokud d není dělitelem x nebo d není dělitelem y opakuj krok 3, jinak proved' krok 4
 3. Zmenši d o 1
 4. Výsledkem je hodnota d

Návrh algoritmu se vlastně skládá z definice typu vstupních dat a výstupních dat (případně pomocných dat pro výpočet) spolu s postupem výpočtu.

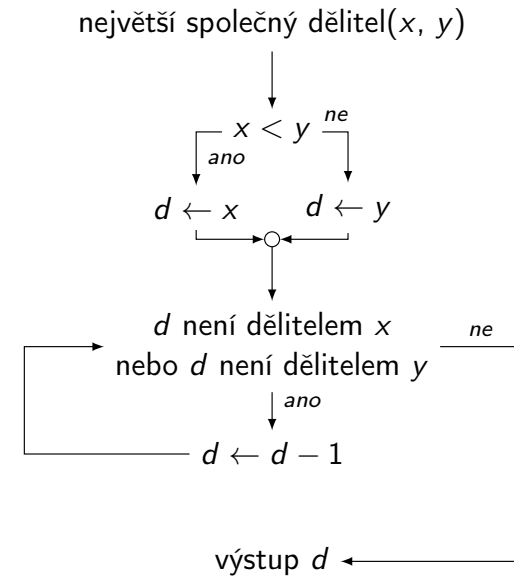
Řídicí struktury

- Řídicí struktura je programová konstrukce, která se skládá z dílčích příkazů a předepisuje pro ně způsob provedení

■ Tři základní druhy řídicích struktur

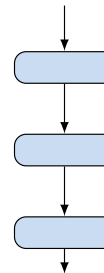
- **Posloupnost** – předepisuje **postupné provedení** dílčích příkazů
- **Větvení** – předepisuje provedení dílčích příkazů v závislosti na **splnění určité podmínky**
- **Cyklus** – předepisuje **opakované provedení** dílčích příkazů v závislosti na splnění určité podmínky

Příklad algoritmu – vývojový diagram

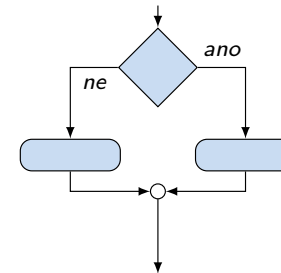


Typy řídicích struktur 1/2

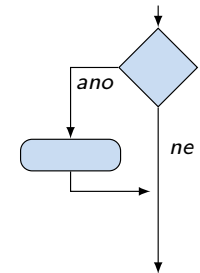
■ Sekvence



■ Podmínka If

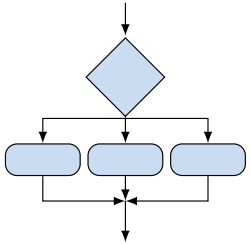


■ Podmínka If

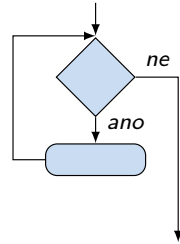


Typy řídicích struktur 2/2

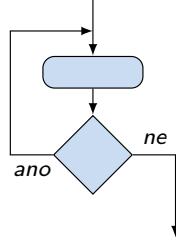
■ Větvení **switch**



■ Cyklus **for** a **while**



■ Cyklus **do**



Větvení **if**

- Příkaz **if** umožňuje větvení programu na základě podmínky
- Má dva základní tvary
 - **if** (*podmínka*) příkaz₁
 - **if** (*podmínka*) příkaz₁ **else** příkaz₂
- podmínka je logický výraz, jehož hodnota je typu **boolean**
tj. *true* nebo *false*
- příkaz je příkaz, složený příkaz nebo blok
příkaz je zakončen středníkem ;
- Ukázka zápisu na příkladu zjištění menší hodnoty z *x* a *y*:

Varianta zápisu 1

```
int min = y;
if (x < y) min = x;
```

Varianta zápisu 2

```
int min = y;
if (x < y)
    min = x;
```

Varianta zápisu 3

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

Která varianta splňuje kódovací konvenci a proč?

Složený příkaz a blok

- Řídicí struktury mají obvykle formu strukturovaných příkazů v Javě to jsou

- **Složený příkaz** – posloupnost příkazů
- **Blok** – posloupnost deklarací a příkazů

```
{
    //blok je vymezen složenými závorkami
    int steps = 10;

    System.out.println("No. of steps" + steps);
}

steps += 1; //nelze - mimo rozsah platnosti bloku
```

Deklarace – alokace paměti podle konkrétního typu proměnné. Rozsah platnosti deklarace je lokální v rámci bloku.

- Budeme používat složené příkazy:

- složený příkaz nebo blok pro posloupnost
- příkaz **if** nebo **switch** pro větvení
- příkaz **while**, **do** nebo **for** pro cyklus

Podmíněné opakování bloku nebo složeného příkazu

Příklad větvení **if**

Příklad: Jestliže $x < y$ vyměňte hodnoty těchto proměnných

Nechť proměnné *x* a *y* jsou deklarovány a jsou typu *int*.

Varianta 1

```
if (x < y)
    tmp = x;
    x = y;
    y = tmp;
```

Varianta 2

```
if (x < y)
    int tmp = x;
    x = y;
    y = tmp;
```

Varianta 3

```
int tmp;
if (x < y)
    tmp = x;
    x = y;
    y = tmp;
```

Varianta 4

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

- Která varianta je správně a proč?

Příklad větvení **if-then-else**

Příklad: do proměnné *min* uložte menší z čísel *x* a *y* a do *max* uložte větší z čísel.

Varianta 1

```
if (x < y)
    min = x;
    max = y;
else
    min = y;
    max = x;
```

Varianta 2

```
if (x < y) {
    min = x;
    max = y;
} else {
    min = y;
    max = x;
}
```

- Která varianta odpovídá našemu zadání?

Cyklus **for**

- Základní příkaz cyklu **for** má tvar **for** (*inicializace*; *podmínka*; *změna*) příkaz
- Odpovídá cyklu **while** ve tvaru:


```
inicializace;
while (podmínka) {
    příkaz;
    změna;
}
```
- Změnu řídicí proměnné lze zkráceně zapsat operátorem inkrementace nebo dekrementace **++** a **--**
- Alternativně lze též použít zkrácený zápis přiřazení, např. **+=**

Příklad

```
for (int i = 0; i < 10; ++i) {
    System.out.println("i: " + i);
}
```

Cyklus **while**

- Základní příkaz cyklu **while** má tvar **while** (*podmínka*) příkaz

Příklad

```
int x = 10;
int y = 3;
int q = x;

while (q >= y) {
    q = q - y;
}
```

- Jaká je hodnota proměnné *q* po skončení cyklu?

Shrnutí přednášky

Diskutovaná témata

- Číselné typy, výrazy, přiřazení a operátory
- Výstup programu a matematické funkce
- Algoritmus a jeho popis
- Přehled řídicích struktur, složený příkaz
- Větvení **if**, cykly **while** a **for**

Kódovací konvence

- **Příště: Dokončení řídicích struktur, textové řetězce, vstup programu, reprezentace typů**