6. Příkazy a řídící struktury v Javě

- Příkazy v Javě
- Řídicí příkazy (větvení, cykly)

Příkazy v Javě

Přiřazovací příkaz `=`

Řízení toku programu (větvení, cykly)

Volání metody

Návrat z metody - příkaz `return`

Příkaz je ukončen středníkem `;`

v Pascalu středník příkazy ***odděluje***, v Javě (C/C++) ***ukončuje***

Přiřazení v Javě

Operátor přiřazení `=` (assignment)

na levé straně musí být proměnná

na pravé straně výraz ***přiřaditelný*** (assignable) do této proměnné

Rozlišujeme přiřazení primitivních hodnot a odkazů na objekty

Přiřazení primitivní hodnoty

Na pravé straně výraz vracející hodnotu primitivního typu

číslo, logická hodnotu, znak (ale ne např. řetězec)

Na levé straně proměnná téhož typu jako přiřazovaná hodnota nebo typu širšího

např. `int` lze přiřadit do `long`

Při zužujícím přiřazení se také provede konverze, ale může dojít ke ztrátě informace

např. `int` -> `short`

Přiřazením primitivní hodnoty se hodnota zduplikuje ("opíše") do proměnné na levé straně.

Přiřazení odkazu na objekt

Konstrukci `=` lze použít i pro přiřazení do objektové proměnné:

```
Zivocich z1 = new Zivocich();
```

Co to udělalo?

1. vytvořilo nový objekt typu `Zivocich` (***`new Zivocich()`***)
2. přiřadilo jej do proměnné `z1` typu `Zivocich`

Nyní můžeme ***odkaz*** na tentýž vytvořený objekt znovu přiřadit - do `z2`:

```
Zivocich z2 = z1;
```

Proměnné `z1` a `z2` ukazují nyní na stejný objekt typu živočich!!!

Proměnné objektového typu obsahují ***odkazy*** (reference) na objekty, ne objekty samotné!!!

Volání metody

Metoda objektu je vlastně procedura/funkce, která realizuje svou činnost primárně s proměnnými objektu.

Volání metody určitého objektu realizujeme:

identifikaceobjektu.názevmetody(skutečnéparametry)

- **identifikace objektu**, jehož metodu voláme
- **.** (tečka)
- **název metody**, jíž nad daným objektem voláme
- v závorách uvedeme ***skutečné parametry*** volání (záv. může být prázdná, nejsou-li parametry)

Návrat z metody

Buďto automaticky posledním příkazem v těle metody

nebo explicitně příkazem `return` ***návratováhodnota***

způsobí ukončení provádění těla metody a návrat, přičemž může být specifikována návratová hodnota

typ skutečné návratové hodnoty musí korespondovat s deklarovaným typem návratové hodnoty

Řízení toku programu v těle metody

Příkaz (neúplného) větvení `if`

`if (logický výraz) příkaz`

platí-li ***logický výraz*** (má hodnoty `true`), provede se ***příkaz***

Příkaz úplného větvení `if` - `else`

```
if (logický výraz)
    příkaz1
else
    příkaz2
```

platí-li ***logický výraz*** (má hodnoty `true`), provede se ***příkaz1***

neplatí-li, provede se ***příkaz2***

Větev `else` se **nemusí uvádět**

Cyklus s podmínkou na začátku

Tělo cyklu se provádí tak dlouho, **dokud** platí podmínka

obdoba `while` v Pascalu

v těle cyklu je jeden jednoduchý příkaz ...

```
while (podmínka)
    příkaz;
```

... nebo příkaz složený

```
while (podmínka) {
    příkaz1;
    příkaz2;
    příkaz3;
    ...
}
```

Tělo cyklu se nemusí provést ani jednou - pokud už hned na začátku podmínka neplatí

Doporučení k psaní cyklů/větvení

Větvení, cykly: doporučuji vždy psát se **složeným příkazem v těle** (tj. se složenými závorkami)!!!

jinak hrozí, že se v těle větvení/cyklu z neopatrnosti při editaci objeví něco jiného, než chceme, např.:

```
while (i < a.length)
    System.out.println(a[i]); i++;
```

`i`

Pišme proto vždy takto:

```
while (i < a.length) {
    System.out.println(a[i]); i++;
}
```

Příklad použití "while" cyklu

Dokud nejsou přečteny všechny vstupní argumenty:

```
int i = 0;
while (i < args.length) {
    "přečti argument args[i]"
    i++;
}
```

Dalším příkladem je použití `while` pro realizaci celočíselného dělení se zbytkem:

Celočíselné dělení se zbytkem

Cyklus s podmínkou na konci

Tělo se provádí **dokud** platí podmínka (vždy aspoň jednou)

obdoba `repeat` v Pascalu (podmínka je ovšem ***interpretována opačně***)

Relativně málo používaný - je méně přehledný než `while`

Syntaxe:

```
do {
    příkaz1;
    příkaz2;
    příkaz3;
    ...
} while (podmínka);
```

Příklad použití "do-while" cyklu

Dokud není z klávesnice načtena požadovaná hodnota:

```
String vstup = "";
float cislo;
boolean nacteno; // vytvoř reader ze standardního vstupu
BufferedReader in = new BufferReader(new InputStream(System.in)); // dokud není zadáno
číslo, čti
do {
    vstup = in.readLine();
    try {
        cislo = Float.parseFloat(vstup);
        nacteno = true;
    } catch (NumberFormatException nfe) {
        nacteno = false;
    }
} while (!nacteno);
System.out.println("Nacteno cislo "+cislo);
```

```

```

Načítej, dokud není zadáno číslo

Cyklus "for"

obecnější než `for` v Pascalu, podobně jako v C/C++

De-facto jde o rozšíření `while`, lze jím snadno nahradit

Syntaxe:

```
for (počáteční operace; vstupní podmínka; příkaz po každém průchodu)
    příkaz;
```

anebo (obvyklejší, bezpečnější)

```
for (počáteční operace; vstupní podmínka; příkaz po každém průchodu) {
    příkaz1;
    příkaz2;
    příkaz3;
    ...
}
```

Příklad použití "for" cyklu

Provedení určité sekvence určitý počet krát

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

Vypíše na obrazovku deset řádků s čísly postupně 0 až 9 Pět pozdravů nebo Výpis prvků pole objektů "for" cyklem

Doporučení k psaní `for` cyklů

Používejte asymetrické intervaly (ostrá a neostrá nerovnost):

podmínka daná počátečním přiřazením `i = 0` a inkrementací `i++` je ***neostrou nerovností***, zatímco

opakovací podmínka `i < 10` je ***ostrou nerovností*** -> `i` už nesmí hodnoty 10 dosáhnout!

Vytvarujte se složitých příkazů v hlavičce (kulatých závorkách) `for` cyklu -

je lepší to napsat podle situace před cyklus nebo až do jeho těla

Někteří autoři nedoporučují psát deklaraci řídicí proměnné přímo do závorek cyklu

`for (int i = 0; ...`

ale rozepsat takto:

```
int i;
for (i = 0; ...
```

potom je proměnná `i` přístupná ("viditelná") i mimo cyklus - za cyklem, což se však ne vždy hodí.

Vícecestné větvení "switch - case - default"

Obdoba pascalského `select - case - else`

Větvení do více možností na základě ordinální hodnoty

Syntaxe:

```
switch(výraz) {
    case hodnota1: prikaz1a;
                   prikaz1b;
```

```
                    prikaz1c;
                    ...
                    break;
    case hodnota2: prikaz2a;
                    prikaz2b;
                    ...
                    break;
    default:        prikazDa;
                    prikazDb;
                    ...
}
```

Je-li ***výraz*** roven některé z hodnot, provede se sekvence uvedená za příslušným `case`. Sekvenci obvykle ukončujeme příkazem `break`, který předá řízení ("skočí") na první příkaz za ukončovací závorkou příkazu `switch`.Vícecestné větvení

Vnořené větvení

Větvení `if - else` můžeme samozřejmě vnořovat do sebe:

```
if (podmínka_vnější) {
    if (podmínka
vnitřní
1) {
        ...
    } else {
        ...
    }
} else {
    if (podmínka
vnitřní
2) {
        ...
    } else {
        ...
    }
}
```

Vnořené větvení (2)

Je možné "šetřit" a neuvádět složené závorky, v takovém případě se `else` vztahuje vždy k nejbližšímu neuzavřenému `if`, např. znovu předchozí příklad:

```
if (podmínka_vnější)
    if (podmínka
vnitřní
1)
        ...
    else // vztahuje se k nejbližšímu if
         // s if (podmínka
vnitřní
1)
        ...
else // vztahuje se k prvnímu if,
     // protože je v tuto chvíli
     // nejbližší neuzavřené
    if (podmínka
vnitřní
2)
        ...
    else // vztahuje se k if (podmínka
vnitřní
2)
        ...
```

Tak jako u cyklů - tento způsob zápisu nelze v žádném případě doporučit!!!

Vnořené větvení

Řetězené "if - else if - else"

Někdy rozvíjíme pouze druhou (negativní) větev:

```
if (podmínka1) {
    ...
} else if (podmínka2) {
    ...
} else if (podmínka3) {
    ...
} else {
    ...
}
```

Neplatí-li podmínka1, testuje se podmínka2, neplatí-li, pak podmínka3...

neplatí-li žádná, provede se příkaz za posledním - samostatným - `else`.

Opět je dobré všude psát složené závorky!!!

Řetězené if

Příkazy "break"

Realizuje "násilné" ukončení průchodu cyklem nebo větvením `switch`

Syntaxe použití `break` v **cyklu**:

```
for (int i = 0; i < a.length; i++) {
    if(a[i] == 0) {
        break; // skoci se za konec cyklu
    }
}
if (a[i] == 0) {
    System.out.println("Nasli jsme 0 na pozici "+i);
} else {
    System.out.println("0 v poli neni");
}
```

použití u `switch` jsme již viděliVícecestné větvení "switch - case - default"

Příkaz "continue"

Používá se v těle cyklu.

Způsobí přeskočení zbylé části průchodu tělem cyklu

```
for (int i = 0; i < a.length; i++) {
    if (a[i] == 5)
        continue;
    System.out.println(i);
}
```

Výše uvedený příklad vypíše čísla `1`, `2`, `3`, `4`, `6`, `7`, `8`, `9`, nevypíše hodnotu `5`. Řízení průchodu cyklem pomocí "break" a "continue"

"break" a "continue" s návěštím

Umožní ještě jemnější řízení průchodu vnořenými cykly

pomocí návěští můžeme naznačit, který cyklus má být příkazem `break` přerušen nebo

tělo kterého cyklu má být přeskočeno příkazem `continue`. Návěští

Doporučení k příkazům break a continue

Raději NEPOUŽÍVAT, ale jsou menším zlem než by bylo `goto` (kdyby v Javě existovalo...), protože

nepředávají řízení dále než za konec struktury (cyklu, větvení).

Toto však již neplatí pro break a continue na návěští!

Poměrně často se používá `break` při sekvenčním vyhledávání prvku