

Řídicí struktury v Javě

Přehled

- Větvení `if-else`
- Cyklus `while` - kde stačí vstupní po
- Cyklus `for` - vstupní a pokračovací podmínka, akce po každém provedení - obdoba téhož v C/C++
- Cyklus `for` ve variantě *iterace* po prvcích pole, seznamu, množiny... - tedy něco jako `foreach`
- Cyklus `do-while` - pokračovací podmínka se testuje na konci těla cyklu
- Vícecestné větvení `switch - case - default` - podobné jako v C/C++

Větvení výpočtu — podmíněný příkaz

Podmíněný příkaz

neboli *neúplné větvení* pomocí `if`

```
if (logický výraz) příkaz
```

- Platí-li *logický výraz* (má hodnotu `true`), provede se *příkaz*.
- Neplatí-li, neprovede se nic.

Příklad podmíněného příkazu

Tedy javové `if`:

```
if (name.equals("Debora"))  
    System.out.println("Hi, Debora");
```

je ekvivalentní `if` v Pythonu:

```
if name == 'Debora':  
    print('Hi, Debora')
```

Rozdíly:

- v Javě stejně jako v C/C++ musejí být závorky kolem podmínky
- v Pythonu je za podmínkou dvojtečka `:`
- a povinné odsazení příkazu/ů, které se podmíněně provedou

Příklad podmíněného bloku příkazů

V Javě nutno uzavřít do bloku {}:

```
if (name.equals("Debora")) {  
    System.out.println("Hi, Debora");  
    System.out.println("I am your friend!");  
}
```

V Pythonu stačí odsadit:

```
if name == 'Debora':  
    print('Hi, Debora')  
    print('I am your friend!')
```

Úplné větvení

- `if` (logický výraz) `příkaz1` `else` `příkaz2`
- Platí-li *logický výraz*, provede se *příkaz1*.
- Neplatí-li, provede se *příkaz2*.

Úplné větvení

V Javě:

```
if (name.equals("Debora"))  
    System.out.println("Hi Debora");  
else  
    System.out.println("Who are you?");
```

V Pythonu:

```
if name == 'Debora':  
    print('Hi Debora!')  
else:  
    print('Who are you?')
```

Lépe do bloků

Je-li ve větvích po jednom příkazu, lze nechat tak. Lepší je ovšem vždy závorkovat, uzavřít bloky:

```
if (name.equals("Debora")) {
    System.out.println("Hi Debora");
} else {
    System.out.println("Who are you?");
}
```

Postupné větvení

V Javě není speciální konstrukce "elseif", která v negativním případě zkouší další větev. Jednoduše se použije `else` a za ním `if`.

```
if (name.equals("Debora"))
    System.out.println("Hi Debora");
else if(name.equals("Joshua"))
    System.out.println("Hi Joshua");
else
    System.out.println("Who are you?");
```

Zatímco v Pythonu ano:

```
if name == 'Debora':
    print('Hi Debora!')
elif name == 'Joshua':
    print('Hi Joshua!')
else:
    print('Who are you?')
```

Rozepsané postupné větvení

Výše uvedená javová konstrukce přepsaná s blokovými závorkami:

```
if (name.equals("Debora")) {
    System.out.println("Hi Debora");
} else {
    if(name.equals("Joshua")) {
        System.out.println("Hi Joshua");
    } else {
        System.out.println("Who are you?");
    }
}
```

Cyklus `while`, tj. s podmínkou na začátku

`while`

Tělo cyklu se provádí tak dlouho, **dokud** platí podmínka, obdoba v Pascalu, C a dalších

- V těle cyklu je jeden jednoduchý příkaz:

```
while (podmínka) příkaz;
```

Cyklus `while` se složeným příkazem

- Nebo příkaz složený z více a uzavřený ve složených závorkách:

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

- Tělo cyklu se nemusí provést ani jednou — to v případě, že hned při prvním testu 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++;
```

- Provede v cyklu jen ten výpis, inkrementaci již ne a program se tudíž zacyklí!!!

Doporučení k psaní cyklů/větvení

- Pišme proto vždy takto:

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

```
}
```

- U větvení obdobně:

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

Příklad použití **while** cyklu

- Dokud nejsou přečteny všechny vstupní argumenty — vč. toho případu, kdy není ani jeden:

```
int i = 0;  
while (i < args.length) {  
    System.out.println(args[i]);  
    i++;  
}
```

Příklad **while** — celočíselné dělení

- Dalším příkladem (pouze ilustračním, protože na danou operaci existuje v Javě vestavěný operátor) je použití **while** pro realizaci celočíselného dělení se zbytkem.

```
public class DivisionBySubtraction {  
    public static void main(String[] args) {  
        int dividend = 10; // dělenec  
        int divisor = 3; // dělitel  
        int quotient = 0; // podíl  
        int remainder = dividend;  
        while (remainder >= divisor) {  
            remainder -= divisor;  
            quotient++;  
        }  
        System.out.println("Podíl 10/3 je " + quotient);  
        System.out.println("Zbytek 10/3 je " + remainder);  
    }  
}
```

Cyklus **do-while**, tj. 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 — hodí se tam, kde něco musí aspoň jednou proběhnout

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

Příklad použití `do-while` cyklu

- Tam, kde pro úspěch algoritmu "musím aspoň jednou zkusit", např. čtu tak dlouho, dokud není z klávesnice načtena požadovaná hodnota.

```
float number;  
boolean isOK;  
// create a reader from standard input  
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));  
// until a valid number is given, try to read it  
do {  
    String input = in.readLine();  
    try {  
        number = Float.parseFloat(input);  
        isOK = true;  
    } catch (NumberFormatException nfe) {  
        isOK = false;  
    }  
} while(!isOK);  
System.out.println("We've got the number " + number);
```

Totéž s možností ukončení

- Použití příkazu `break`
- Realizuje "násilné" ukončení průchodu *cyklem* (nebo *větvením* `switch`).
- Doplnění `break` do *cyklu*:

```
float number;  
boolean isOK;  
// create a reader from standard input  
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));  
// until a valid number is given, try to read it  
do {  
    String input = in.readLine();  
    // if the input is empty then finish  
    if(input.length() == 0) break;  
}
```

```

try {
    number = Float.parseFloat(input);
    isOK = true;
} catch (NumberFormatException nfe) {
    isOK = false;
}
} while(!isOK);
System.out.println("We've got the number " + number);

```

break v cyklu a podmínice

- **break** ukončí cyklus **for** podobně jako předtím **do-while**:

```

int i = 0;
for (; 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");
}

```

Příklad: Načítej, dokud není zadáno číslo

```

import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.IOException;
public class UntilEnteredEnd {
    public static void main(String[] args) throws IOException {
        BufferedReader input = new BufferedReader(
            new InputStreamReader(System.in));
        String line = "";
        do {
            line = input.readLine();
        } while (!line.equals("end"));
        System.out.println("Uživatel zadal " + line);
    }
}

```

Cyklus `for`

- Obdobně jako `for` cyklus v C/C++ jde de-facto o rozšíření cyklu `while`.
- Zapisujeme takto:

```
for(počáteční op.; vstupní podm.; příkaz po každém průch.)  
    příkaz;
```

- Anebo obvykleji a bezpečněji mezi `{` a `}` proto, že když přidáme další příkaz, už nezapomeneme dát jej do složených závorek:

```
for (počáteční op.; vstupní podm.; příkaz po každém průch.) {  
    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.

Doporučení — asymetrické intervaly a pevný počet

- `for` se většinou užívá jako cyklus s pevným počtem opakování, známým při vstupu do cyklu. Tento počet nemusí být vyjádřený konstantou (přímo zadaným číslem), ale neměl by se v průběhu cyklu měnit.
- Používejte *asymetrické* intervaly (ostrá a neostrá nerovnost):
 - počáteční přiřazení `i = 0` a
 - inkrementaci `i++` je *neostrou nerovností*: `i` se na začátku rovná 0), zatímco
 - opakovací podmínka `i < 10` je *ostrou nerovností*: `i` už hodnoty 10 *nedosáhne!*
- Vytvarujte se složitých příkazů v hlavičce (kulatých závkách) `for` cyklu.
- Je lepší to napsat podle situace před cyklus nebo až do jeho těla!

Doporučení — řídicí proměnná

- V cyklu `for` se téměř vždy vyskytuje tzv. *řídicí proměnná*,
- tedy ta, která je v něm inicializována, (obvykle) inkrementována a testována.
- 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 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, v novějších verzích Javy i podle hodnot jiných typů, vč. objektových.
- Chová se spíše jako `switch-case` v C, — zejména se chová jako C při "break-through"

Struktura `switch - case - default`

```
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`.

`switch` další info

- Řídicí výraz může nabývat hodnot
 - primitivních typů `byte`, `short`, `char` a `int`, dále

- výčtových typů (`enum`),
- typu `String` a některých dalších.
- Tutoriál Oracle Java: [Switch statement](#)

switch příklad s čísly

```
public class MultiBranching {
    public static void main(String[] args) {
        if (args.length == 1) {
            int i = Integer.parseInt(args[0]);
            switch (i) {
                case 1: System.out.println("jednicka"); break;
                case 2: System.out.println("dvojka"); break;
                case 3: System.out.println("trojka"); break;
                default: System.out.println("neco jineho"); break;
            }
        } else {
            System.out.println("Pouziti: java MultiBranching <cislo>");
        }
    }
}
```

switch příklad se String

Převzato z tutoriálu Oracle

```
switch (month.toLowerCase()) {
    case "january":
        monthNumber = 1;
        break;
    case "february":
        monthNumber = 2;
        break;
    case "march":
        monthNumber = 3;
        break;
    ...
}
```

switch příklad se společnými větvemi case

Převzato z tutoriálu Oracle

```
int month = 2;
int year = 2000;
```

```
int numDays = 0;
```

```
switch (month) { case 1: case 3: case 5: case 7: case 8: case 10: case 12: numDays = 31; break; case 4:  
case 6: case 9: case 11: numDays = 30; break; ...
```

Použití nové syntaxe větví →

- V nových verzích Java 14+ lze použít namísto otravného ukončování větví pomocí **break** (což je kdysi poděděné z C) nové syntaxe s šipkou →.

```
switch (day) {  
    case MONDAY, FRIDAY, SUNDAY -> System.out.println(6);  
    case TUESDAY                 -> System.out.println(7);  
    case THURSDAY, SATURDAY      -> System.out.println(8);  
    case WEDNESDAY               -> System.out.println(9);  
}
```

- Ve vybrané větvi se provede příkaz nebo blok, je-li uveden v { }.
- Jednu větev lze vybrat více výrazy současně (např. **MONDAY, FRIDAY, SUNDAY**).

Výraz **switch**

- switch nemusíme používat jen jako příkaz vícecestného větvení
- zejména ve výše uvedených příkladech, kdy se v každé větvi provedlo jen přiřazení do stejné proměnné, je lepší použít switch v nové formě jako jakýsi *rozšířený podmíněný výraz*
- funguje v Javě 14+

```
int numLetters = switch (day) {  
    case MONDAY, FRIDAY, SUNDAY -> 6;  
    case TUESDAY                 -> 7;  
    case THURSDAY, SATURDAY      -> 8;  
    case WEDNESDAY               -> 9;  
};
```

Vnořené větvení

- Větvení **if - else** můžeme samozřejmě vnořovat do sebe.
- Toto je vhodný způsob zápisu:

```
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 if(podmínka_vnitřní_1)
else // vztahuje se k if(podmínka_vnější)
    if (podmínka_vnitřní_2)
        ...
    else // vztahuje se k if (podmínka_vnitřní_2) ...

```

- Tak jako u cyklů ani zde tento způsob zápisu (bez závorek) nelze v žádném případě doporučit!!!

Příklad vnořené větvení

```

public class NestedBranching {
    public static void main(String args[]) {
        int i = Integer.parseInt(args[0]);
        System.out.print(i+" je cislo ");
        if (i % 2 == 0) {
            if (i > 0) {
                System.out.println("sude, kladne");
            } else {
                System.out.println("sude, zaporne nebo 0");
            }
        } else {
            if (i > 0) {
                System.out.println("liche, kladne");
            } else {
                System.out.println("liche, zaporne");
            }
        }
    }
}

```

```
}  
}
```

Řetězené **if - else if - else**

- Časteji rozvíjíme pouze druhou (*negativní*) větev:

```
if (podmínka1) {  
    ... // platí podmínka1  
} else if (podmínka2) {  
    ... // platí podmínka2  
} else if (podmínka3) {  
    ... // platí podmínka3  
} else {  
    ... // neplatila žádná  
}
```

- Opět je dobré všude psát složené závorky.

Příklad **if - else if - else**

```
public class MultiBranchingIf {  
    public static void main(String[] args) {  
        if (args.length == 1) {  
            int i = Integer.parseInt(args[0]);  
            if (i == 1)  
                System.out.println("jednicka");  
            else if (i == 2)  
                System.out.println("dvojka");  
            else if (i == 3)  
                System.out.println("trojka");  
            else  
                System.out.println("jine cislo");  
        } else {  
            System.out.println("Pouziti: java MultiBranchingIf <cislo>");  
        }  
    }  
}
```

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; // pětku vynecháme
    System.out.println(i);
}

```

- Výše uvedený příklad vypíše čísla 1, 2, 3, 4, 6, 7, 8, 9, nevypíše hodnotu 5.

Příklad na **break** i **continue**

```

public class BreakContinue {
    public static void main(String[] args) {
        if (args.length == 2) {
            int limit = Integer.parseInt(args[0]);
            int skip = Integer.parseInt(args[1]);
            for (int i = 1; i <= 20; i++) {
                if (i == skip)
                    continue;
                System.out.print(i+" ");
                if (i == limit)
                    break;
            }
            System.out.println("\nKonec cyklu");
        } else {
            System.out.println(
                "Pouziti: java BreakContinue <limit> <vynechej>");
        }
    }
}

```



Příklad je pouze ilustrativní—v reálu bychom **break** na ukončení cyklu v tomto případě nepoužili a místo toho bychom **limit** dali přímo jako horní mez **for** cyklu.

break a **continue** s návěstím

- Umožní ještě jemnější řízení průchodu vnořenými cykly:
 - pomocí návěstí můžeme naznačit, který cyklus má být příkazem **break** přerušeno nebo
 - tělo kterého cyklu má být přeskočeno příkazem **continue**.

```

public class Label {
    public static void main(String[] args) {
        outer_loop:
        for (int i = 1; i <= 10; i++) {
            for (int j = 1; j <= 10; j++) {
                System.out.print((i*j)+" ");
            }
        }
    }
}

```

```
        if (i*j == 25) break outer_loop;
    }
    System.out.println();
}
System.out.println("\nKonec cyklu");
}
```

Repl.it demo k řídicím strukturám

- <https://repl.it/@tpitner/PB162-Java-Lecture-03-control-structures>