4. Základní konstrukce metod

V těle metody se mohou vyskytnout následující typy příkazů:

- volání metody,
- přiřazení,
- příkaz return,
- sekvence (posloupnost, blok příkazů),
- selekce (rozhodování, větvení),
- iterace (cyklus, opakování), včetně příkazů skoku z cyklu,
- prázdný příkaz,
- vyvolání a obsluha výjimek,
- příkaz assert pro testování.

Tři první uvedené typy příkazů jsme objasnili v kapitole 2. V této kapitole popíšeme další čtyři příkazy, výjimkami se budeme zabývat v kapitole 12, příkazu `assert` se v těchto skriptech nebudeme věnovat.

4.1. Sekvence

Posloupnost příkazů v Javě zapíšeme tak, že jednotlivé příkazy a deklarace oddělíme středníkem.

```
    Ucet mujUcet = new Ucet (1,"Jarmila",10000);
    mujUcet.vloz(100) ;
    double stavMehoUctu = mujUcet.getStav() ;
```

Příkazy lze napsat i takto:

```
Ucet mujUcet = new Ucet (1, "Jarmila", 10000); mujUcet.vloz(100) ;double
stavMehoUctu = mujUcet.getStav();
```

ale vzhledem k nepřehlednosti doporučujeme zapisovat na každý řádek jeden příkaz či deklaraci. Řadu po sobě jdoucích příkazů a deklarací můžeme (a v mnoha případech musíme) spojit do bloku pomocí složených závorek.

```
    {
        Ucet mujUcet = new Ucet (1, "Jarmila", 10000);
        mujUcet.vloz(100) ;
        double stavMehoUctu = mujUcet.getStav() ;
    }
```

Po složené závorce už neuvádíme středník.

4.1.1. Rozsah platnosti lokálních proměnných

Každá dvojice složených závorek v metodě představuje blok, který může obsahovat deklaraci lokálních proměnných a příkazy (sekvenci příkazů). S tím souvisí i rozsah platnosti lokální proměnné, tj. kde se lze odkazovat na lokální proměnnou. Pro rozsah platnosti platí dvě základní pravidla:

- lokální proměnnou lze používat od místa, kde se deklaruje, až po uzavírací závorku bloku, ve kterém je deklarována,
- pokud je proměnná či formální parametr deklarovaný v kulatých závorkách těsně před začátkem bloku (před složenými závorkami), tak je lze používat v rámci celého bloku.

Pravidla si ukážeme na následujícím kódu (příkazy `if` a `for` jsou vysvětleny dál v této kapitole):

```
 1  public void metoda(int cislo) {
 2    int promenna1 = 0;
 3    for (int i = 0; i < 10; i++) {
 4        // nějaké příkazy
 5    }
 6    int promenna2 = 0;
 7    if (promenna1 > 0) {
 8        int promenna3 = promenna2;
 9        // další příkazy
10    }
11    // další příkazy
12  }
```

Formální parametr `cislo` se může používat v celé metodě. Lokální proměnná `promenna1` se může používat v celé metodě, neboť je deklarována na prvním řádku a blok končí na posledním řádku metody. Proměnná `i` (řádek 3) se může používat až po konec bloku příkazu `for` na řádku 5. Proměnná `promenna2` se může používat od své deklarace až po konec metody. Proměnná `promenna3` se může používat od místa své deklarace po konec bloku `if` na řádku 10.

> Datové atributy a metody mají rozsah platnosti po celé třídě. Na rozdíl od lokálních proměnných je lze nejdříve používat a poté deklarovat.

4.2. Selekce

Pro větvení má Java dva příkazy: `if` a `switch`.

4.2.1. Příkaz if

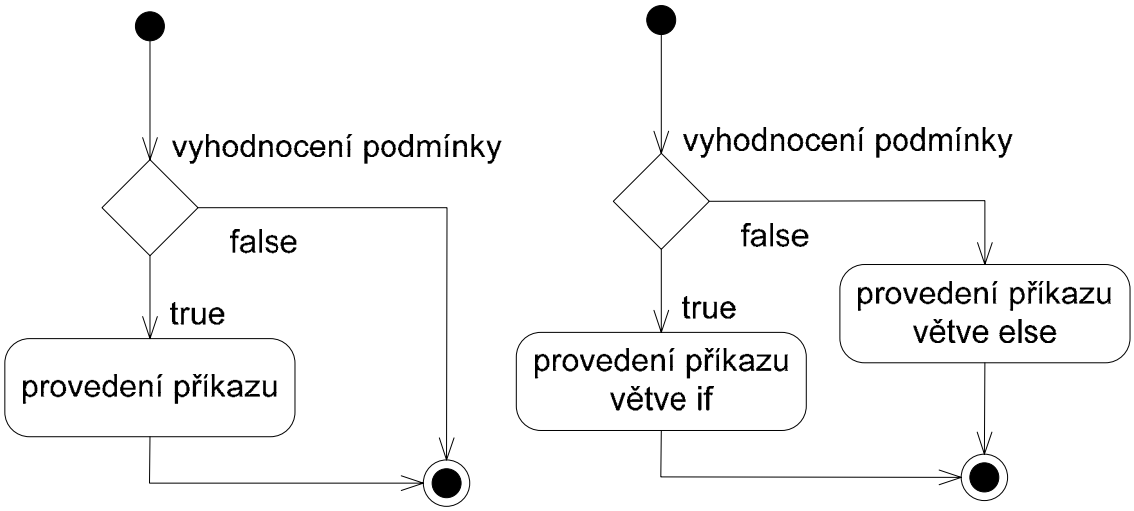
Příkaz **if** má podmínku a jednu nebo dvě větve. Výsledkem podmínky musí být hodnota typu `boolean`. Podmínka je vždy uvedena v kulatých závorkách. Syntaxe příkazu `if` s jednou větví je následující:

```
if (podmínka) {
    příkaz;
}
```

Příkaz se provede pouze v případě, že podmínka platí, tj. byla vyhodnocena jako `true`. Pokud podmínka byla vyhodnocena jako `false`, neprovede se nic, příkaz `if` skončí a kód pokračuje dalším příkazem.
Syntaxe příkazu `if` s větví **else**:

```
if (podmínka) {
    příkaz1;
}
else {
    příkaz2;
}
```

Pokud je výsledek vyhodnocení podmínky hodnota `true`, provede se `příkaz1`, pokud je výsledkem `false`, provede se `příkaz2`. Následující diagramy zobrazují obě varianty příkazu `if` – bez větve `else` a s větví `else`.



Obrázek 4.1 Diagram průběhu příkazu if s jednou a dvěma větvemi

Každá větev příkazu *if* může obsahovat libovolný počet příkazů (jsou seskupeny ve složených závorkách). Jako příklad použití příkazu *if* si můžeme znovu uvést kód metody pro výběr z účtu z kapitoly 0.

```
public boolean vyber (double castka){
     if ((stav – castka) >= 0) {
          stav = stav – castka;
          return true;
     }
     else {
          return false;
     }
}
```

Při zápisu příkazu *if* do kódu je vhodné dodržovat následující pravidla:

- větev *if* i *else* vždy uvádět se složenými závorkami, a to i v případě, že větev obsahuje jen jeden příkaz,
- odsazovat kód a párovat závorky kvůli přehlednosti.

4.2.2. Příkaz switch

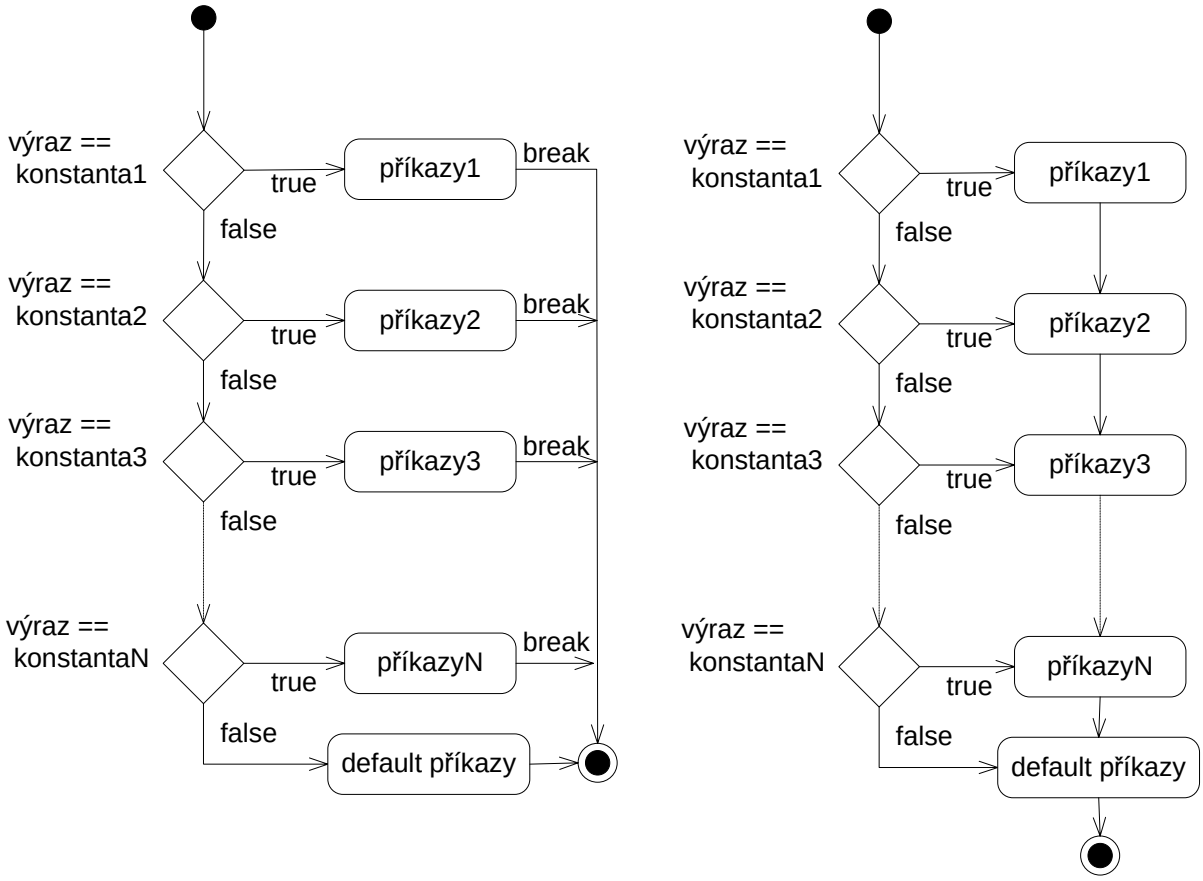
Méně používaný příkaz **switch** na základě hodnoty výrazu provádí příslušnou větev příkazu (větví obvykle bývá několik). Výraz musí být typu *char*, *byte*, *short* nebo *int* nebo výčtový typ. Syntaxe je následující:

```
switch (výraz) {
     case konstanta1: příkazy1; [break;]
     case konstanta2: příkazy2; [break;]
     case konstanta3: příkazy3; [break;]
     case konstanta4: příkazy4; [break;]
     // větví může být libovolný počet
     default: příkazy;
}
```

V bloku může být umístěno několik větví začínajících klíčovým slovem **case**, konstantou a dvojtečkou (v Javě nelze použít např. podmínku i > 5). Za dvojtečkou může být uvedeno více příkazů, obvykle se uvádí dva – vlastní příkaz (např. volání nějaké metody) a příkaz **break**. Jako poslední se obvykle uvádí větev uvozena slovem **default** následovaným dvojtečkou. Při provádění příkazu *switch* se

nejprve vyhodnotí výraz a poté se provedou příkazy ve větvi `case` s odpovídající hodnotou. Pokud se odpovídající hodnota nenajde, provádí se větev `default`. Pokud není větev `case` ukončena příkazem `break`, pokračuje se v provádění příkazů na následující větvi i když je u nich uvedena jiná hodnota než jakou má výraz (tj. záleží na pořadí uvedení větví!). Diagram na obrázku 4.2 zobrazuje průběh zpracování příkazu `switch` ve dvou variantách – s `break` na konci příkazu a bez `break`.

> Podobný význam jako `break` mají ve větvi `case` příkazy `return` (ukončí se celá metoda) a `throw` (vznikne výjimka a přejde se na zpracování výjimky obvykle v jiné metodě).



Obrázek 4.2 Diagram variant průběhu příkazu switch s break na konci větve a bez break

Následující metoda převádí řetězec na malá písmena a „maže“ diakritická znaménka u českých samohlásek (je to součást řešení domácího úkolu z projektu `HadaniSlov`).

```
    private String upravitSlovo(String slovo) {
         char [] znaky = slovo.toLowerCase().toCharArray();
         for (int i = 0; i < znaky.length; i++) {
              switch (znaky[i]) {
                   case 'á': znaky[i]='a'; break;
                   case 'ä': znaky[i]='a'; break;
                   case 'é': znaky[i]='e'; break;
                   case 'ě': znaky[i]='e'; break;
                   case 'ë': znaky[i]='e'; break;
                   case 'í': znaky[i]='i'; break;
                   case 'ó': znaky[i]='o'; break;
                   case 'ö': znaky[i]='o'; break;
                   case 'ú': znaky[i]='u'; break;
                   case 'ů': znaky[i]='u'; break;
```

```
                    case 'ü': znaky[i]='u'; break;
                    case 'ý': znaky[i]='y'; break;
                    default:
                }
            }
            return new String(znaky);
        }
```

V verzi 5.0 byl příkaz `switch` rozšířen o podporu výčtového typu (enum) – popis bude v kapitole 8 věnované tomuto typu.

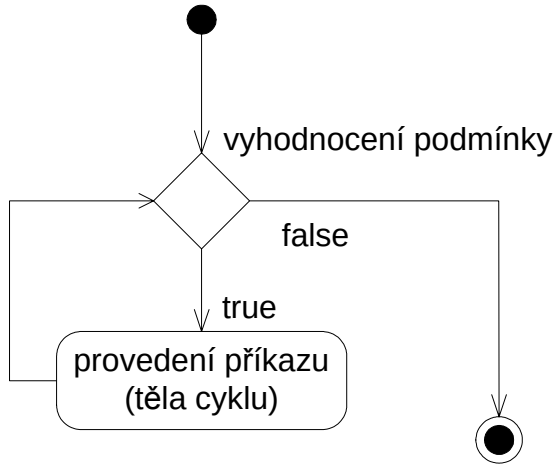
4.3. Iterace (cykly)

V Javě jsou definovány tři druhy cyklů – příkazy `while`, `do-while` a `for`. S nimi souvisí i dva příkazy pro skok z cyklu – `break` a `continue`.

4.3.1. Příkaz while

Nejčastěji se používá cyklus **while** s následující syntaxí:

```
    while ( podmínka ){
        příkaz;
    }
```



Obrázek 4.3 Diagram průběhu příkazu while

Provádění příkazu začíná vždy testem podmínky. Pokud je podmínka splněna (výsledkem je hodnota `true`), provede se příkaz a znovu se přejde na test podmínky. Při nesplnění podmínky provádění cyklu končí. Pokud podmínka není na začátku splněna, neprovede se příkaz v cyklu `while` ani jednou.
Cyklus `while` se nejčastěji používá v situaci, kdy předem neznáme počet opakování. V následující ukázce se generují náhodná celá čísla v intervalu `<0;10)` do té doby, než se vygeneruje číslo 0. Pro generování náhodných čísel se používá třída `Random` z balíčku `java.util`.

```
java.util.Random generator = new java.util.Random();
int nahoda = generator.nextInt(10);
while ( nahoda != 0 ) {
    System.out.println(nahoda);
    nahoda = generator.nextInt(10);
}
```

Cyklus `while` lze použít v situaci, kdy známe počet opakování – většinou se však dává v této situaci přednost příkazu `for`. Následující cyklus proběhne 10x.

```
int i = 1;
while ( i <= 10 ) {
     // příkazy
     i++;
}
```

Pomocí `while` lze nekonečný cyklus zapsat následovně[10]:

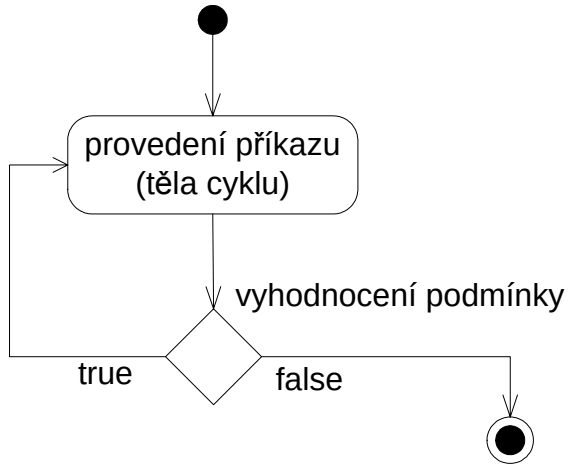
```
     while (true) {
          // příkazy
     }
```

Nekonečný cyklus lze ukončit pomocí příkazů `break` (ukončení cyklu), `return` (ukončení celé metody), `throw` (vyvolání výjimky) či voláním některých metod (ukončení aplikace, ukončení vlákna atd.). Nekonečné cykly používejte pouze v odůvodněných případech – pokud má cyklus skončit po splnění nějaké podmínky, je vždy přehlednější tuto podmínku uvést hned ve `while`.

4.3.2. Příkaz do-while

Nejméně používaným typem cyklu je cyklus **do-while** s následující syntaxí:

```
     do
          příkaz
     while ( podmínka );
```



Obrázek 4.4 Diagram průběhu cyklu do-while

Tento cyklus začíná provedením příkazu a až poté se testuje podmínka. Dle výsledku se provádí znovu příkaz (hodnota `true` podmínky) nebo se končí (hodnota `false`). V cyklu `do-while` se příkaz provede vždy alespoň jednou.

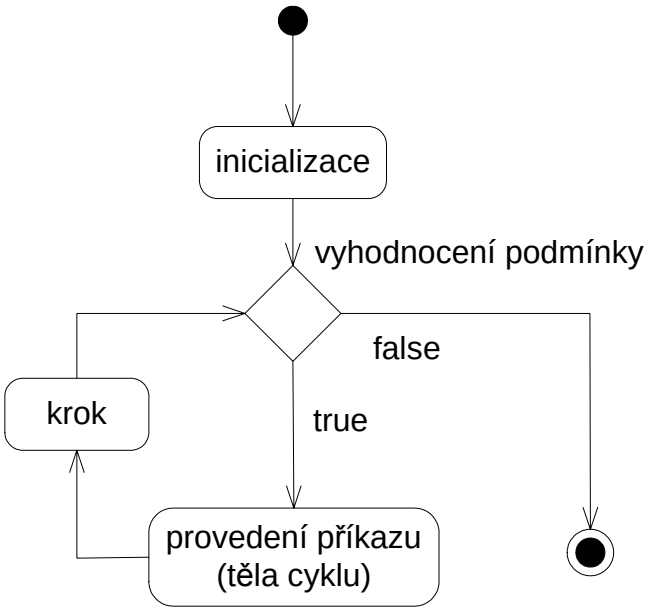
4.3.3. Příkaz for

Posledním typem cyklu je cyklus **for**, který má od verze 5.0 dvě varianty – klasickou variantu s inicializací, podmínkou a krokem a „for each“ variantu, která se používá pro procházení datových struktur. Zde si popíšeme první variantu, druhá varianta cyklu `for` bude popsána v kapitole věnované datovým strukturám (kapitola 11).
Klasický cyklus `for` používáme tam, kde známe počet iterací. Syntaxe je následující:

[10] Toto je jeden z mála případů, kdy se v praxi používají booleovské konstanty `true` či `false`.

```
for (inicializace; podmínka; krok) {
     příkazy;
}
```

Zpracování klasického cyklu `for` je zobrazeno na následujícím diagramu:



Obrázek 4.5 Diagram průběhu příkazu for

Inicializací určujeme **řídící proměnnou cyklu** a její vstupní hodnotu. Pokud je splněna podmínka, provede se příkaz následovaný operací s řídící proměnnou cyklu uvedenou v části krok. Poté se opět přejde na vyhodnocení podmínky. V následujícím příkladě se zopakuje obsah cyklu 10x (příklad odpovídá příkladu u cyklu `while`):

```
for (int i = 1; i <= 10; i++) {
     // příkazy
}
```

Následující cyklus se provede 50x s tím, že řídící proměnná cyklu bude nabývat pouze sudých hodnot:

```
     for (int i = 2; i <= 100; i += 2) {
          // příkazy
     }
```

Příkaz `for` lze přepsat do příkazu `while` následujícím způsobem:

```
     inicializace;
     while (podmínka) {
          příkaz;
          krok;
     }
```

> V hlavičce cyklu `for` lze vynechat jednotlivé části a používat cyklus `for` podobně jako cyklus `while`. Toto však vede k nepřehlednému kódu a doporučujeme se tomuto použití vyhýbat.

4.3.4. Příkazy break a continue

Jazyk Java zná příkazy **break** a **continue**, které ovlivňují průběh zpracování příkazů v rámci cyklu (příkaz `break` se používá i v příkazu `switch`).
Jestliže v těle cyklu použijeme `break`, výsledkem bude skok za konec tohoto cyklu.

```
    for (int i = 0; i <= 5; i++) {
        if (i == 3) break;
        System.out.println(i);
    }
    System.out.println("Konec programu");
```

Tato část programu bude mít tento výstup:

```
    0
    1
    2
    Konec programu
```

Příkaz `continue` způsobí, že je přeskočen zbytek těla cyklu a znovu se testuje podmínka cyklu (v případě cyklu `for` se provede ještě krok řídící proměnné cyklu). Pokud ve výše uvedeném příkladu použijeme místo `break` příkaz `continue`, bude výstup vypadat takto:

```
    0
    1
    2
    4
    5
    Konec programu
```

Za příkazy `Break` i `continue` lze uvést návěstí, na které tyto příkazy skočí – tj. lze opustit i několik do sebe vnořených cyklů. Návěstí ukončené dvojtečkou se uvádí v programu před začátkem vnějšího cyklu.

4.4. Prázdný příkaz

Prázdný příkaz, jak již název napovídá, nic nedělá. Prázdný příkaz vznikne po uvedení samostatného středníku. Výjimečně se použije úmyslně, často je použit neúmyslně. Občas se úmyslně používá v příkazu `if`. Je to situace, kdy by negativní podmínka byla nepřehledná či se obtížně vytváří:

```
if (složitá_podmínka) {
    ;
}
else {
    příkazy;
}
```

Neopatrné umístění prázdného příkazu někdy může mít nepříjemné důsledky. Následující příklad skončí chybou při překladu, neboť větev `else` je od příkazu `if` oddělena prázdným příkazem (přebývajícím středníkem za ukončující složenou závorkou):

```
if (i > 0) {
    // příkazy
};
else {
    // příkazy
}
```

V následujícím příkladu se programátor snaží vypsat čísla od 1 do 10:

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

Program však skončí nekonečným cyklem (tj. nikdy neskončí), neboť za podmínkou `while` je prázdný příkaz – přebývající středník mezi podmínkou a otevírací složenou závorkou.