KAPITOLA 3

Řídicí příkazy

Klíčové znalosti a pojmy
- Vstup znaků z klávesnice
- Úplný tvar příkazu `if`
- Použití příkazu `switch`
- Úplný tvar cyklu `for`
- Použití cyklu `while`
- Použití cyklu `do-while`
- Ukončení cyklu příkazem `break`
- Použití příkazu `break` místo příkazu goto
- Použití příkazu `continue`
- Vnořené cykly

V této kapitole se seznámíte s příkazy, které řídí postup provádění programu. Příkazy řízení programu se řadí do třech kategorií: příkazy *výběru*, kam patří příkazy `if` a `switch`, příkazy *iterace*, které zahrnují cykly `for`, `while` a `do-while`, a příkazy *skoku*, k nimž náleží příkazy `break`, `continue` a `return`. S výjimkou příkazu `return`, kterým se budeme zabývat v další části knihy, rozebereme v této kapitole všechny zbývající řídicí příkazy včetně příkazů `if` a `for`, které jsme již stručně představili. Úvodem kapitoly ukážeme, jak lze zajistit jednoduchý vstup z klávesnice.

Vstup znaků z klávesnice

Než se zaměříme na řídicí příkazy jazyka Java, uděláme krátkou odbočku, díky které dokážeme začít psát interaktivní programy. Až dosud všechny ukázkové programy v knize poskytovaly informace *pro* uživatele, ale nepřijímaly žádná data *od* uživatele. Pracovali jsme tedy s konzolovým výstupem, ale nikoli konzolovým vstupem (vstupem z klávesnice). Hlavní důvod spočívá v tom, že možnosti zadávání dat v jazyce Java jsou založeny na těch jeho vlastnostech, které budeme vysvětlovat teprve v pozdějších částech knihy. Většina reálných programů a apletů jazyka Java navíc namísto textové konzole funguje v prostředí grafického okna. Z těchto důvodů se v této knize s konzolovým vstupem setkáte jen zřídka. Jeden typ konzolového vstupu se však používá poměrně snadno: jedná se o čtení znaků z klávesnice. Vzhledem k tomu, že jsou na uvedené funkci založeny příklady v této kapitole, krátce ji nyní popíšeme.

K načtení znaku z klávesnice slouží metoda `System.in.read()`. Objekt `System.in` je doplněk objektu `System.out`. Jedná se o vstupní objekt připojený ke klávesnici. Metoda `read()` vyčká, dokud uživatel nestiskne nějakou klávesu, a poté vrátí výsledek. Znak je vrácen jako celé číslo, takže je potřeba hodnotu přetypovat na typ `char`, aby ji bylo možné přiřadit proměnné typu `char`. Konzolový vstup se standardně ukládá do *řádkové vyrovnávací paměti* (line buffer). Pojem *vyrovnávací paměť* zde označuje malou část paměti, která uchovává znaky před tím, než

je načte program. V tomto případě vyrovnávací paměť obsahuje celý řádek textu. Chcete-li proto svému programu odeslat libovolné zadané znaky, musíte po nich stisknout klávesu Enter. Následující program načítá znak z klávesnice:

```
// Program načítá znak z klávesnice.
class VstupKlav {
  public static void main(String args[])
    throws java.io.IOException {

    char ch;

    System.out.print("Stiskněte klávesu a poté Enter: ");

    ch = (char) System.in.read(); // načte znak ◄---------- Načtení znaku z klávesnice.

    System.out.println("Byla zadána klávesa: " + ch);
  }
}
```

Ukázka spuštění programu vypadá takto:

```
Stiskněte klávesu a poté Enter: t
Byla zadána klávesa: t
```

V programu si všimněte, že metoda `main()` začíná takto:

```
public static void main(String args[])
  throws java.io.IOException {
```

Vzhledem k tomu, že se používá metoda `System.in.read()`, program musí zahrnovat klauzuli `throws java.io.IOException`. Tento řádek je nezbytný ke zpracování chyb vstupu. Jedná se o součást mechanizmu obsluhy výjimek jazyka Java, k němuž se dostaneme v kapitole 9. Prozatím jeho přesný význam nemusíte znát.

To, že objekt `System.in` používá ukládání do řádkové vyrovnávací paměti, je občas zdrojem potíží. Když stisknete klávesu Enter, je do vstupního datového proudu vložena posloupnost znaků pro návrat na začátek řádku a nový řádek. Tyto znaky navíc zůstávají ve vstupní vyrovnávací paměti, dokud je nenačtete. U některých aplikací je proto nutné uvedené znaky odebrat (jejich přečtením), aby mohla následovat další operace vstupu. Příklad představíme v další části této kapitoly.

Příkaz if

Příkaz `if` jsme předvedli v kapitole 1. V této kapitole jej prozkoumáme podrobněji. Příkaz `if` má tento úplný tvar:

```
if(podmínka) příkaz;
else příkaz;
```

kde cílem příkazů `if` a `else` jsou jednotlivé příkazy. Klauzule `else` není povinná. Cílem příkazů `if` a `else` mohou být také bloky příkazů. Obecný formát příkazu `if` s bloky příkazů vypadá takto:

```
if(podmínka)
{
  posloupnost příkazů
}
else
{
  posloupnost příkazů
}
```

Jestliže je podmínka pravdivá, provede se cíl příkazu `if`. V opačném případě se provede cíl příkazu `else`, je-li uveden. Program v žádném případě nevykoná cíle obou příkazů. Podmínka řídící příkaz `if` musí poskytnout výsledek typu `boolean`.

Na ukázku příkazu `if` (a několika dalších řídicích příkazů) vytvoříme a rozvineme jednoduchou počítačovou hru s hádáním výsledků, která by byla vhodná pro mladší děti. V první verzi hry program požádá hráče o písmeno od A do Z. Pokud hráč stiskne správnou klávesu, program zareaguje vypsáním zprávy `** Správně **`. Kód programu vypadá takto:

```
// Hra s hádáním písmen.
class Hadej {
  public static void main(String args[])
    throws java.io.IOException {

    char ch, odpoved = 'K';

    System.out.println("Myslím si písmeno od A do Z.");
    System.out.print("Zkus je uhodnout: ");

    ch = (char) System.in.read(); // načte znak z klávesnice

    if(ch == odpoved) System.out.println("** Správně **");
  }
}
```

Program vyzve hráče k zadání písmene a poté přečte znak z klávesnice. Pomocí příkazu `if` pak zkontroluje, zda se znak shoduje se správnou odpovědí, což je v tomto případě písmeno `K`. Jestliže hráč zadal písmeno `K`, zobrazí se zpráva. Při zkoušení programu pamatujte na to, že je potřeba zadat velké písmeno `K`.

Když budeme v rozvíjení hry s hádáním pokračovat, můžeme v další verzi pomocí příkazu `else` vypsat zprávu o tom, že uživatel zvolil chybné písmeno.

```
// Druhá verze hry s hádáním písmen.
class Hadej2 {
  public static void main(String args[])
    throws java.io.IOException {
```

```
    char ch, odpoved = 'K';

    System.out.println("Myslím si písmeno od A do Z.");
    System.out.print("Zkus je uhodnout: ");

    ch = (char) System.in.read(); // načte znak

    if(ch == odpoved) System.out.println("** Správně **");
    else System.out.println("Je mi líto, netrefil ses.");
  }
}
```

Vnořené příkazy if

Vnořený příkaz if je příkaz `if`, který je cílem jiného příkazu `if` nebo `else`. Vnořené příkazy `if` jsou v programování velmi časté. Ohledně vnořených příkazů `if` v jazyce Java je potřeba si pamatovat zejména to, že příkaz `else` vždy odkazuje na nejbližší příkaz `if`, který se nachází ve stejném bloku jako příkaz `else` a zatím není přidružen k příkazu `else`. Následuje příklad:

```
if(i == 10) {
  if(j < 20) a = b;
  if(k > 100) c = d;
  else a = c; // tento příkaz else patří k příkazu if(k > 100)
}
else a = d; // tento příkaz else patří k příkazu if(i == 10)
```

Jak naznačují komentáře, poslední příkaz `else` nepatří k příkazu `if(j < 20)`, protože není umístěn ve stejném bloku (i když se jedná o nejbližší příkaz `if` bez příkazu `else`). Závěrečný příkaz `else` místo toho přísluší k příkazu `if(i == 10)`. Vnitřní příkaz `else` odkazuje na příkaz `if(k > 100)`, protože to je nejbližší příkaz `if` v rámci stejného bloku.

Pomocí vnořeného příkazu `if` lze dále vylepšit hru s hádáním písmen. Nová část kódu poskytuje hráči informace o chybné volbě.

```
// Třetí verze hry s hádáním písmen.
class Hadej3 {
  public static void main(String args[])
    throws java.io.IOException {

    char ch, odpoved = 'K';

    System.out.println("Myslím si písmeno od A do Z.");
    System.out.print("Zkus je uhodnout: ");

    ch = (char) System.in.read(); // načte znak

    if(ch == odpoved) System.out.println("** Správně **");
    else {
      System.out.print("Je mi líto, správné písmeno je ");
```

```
      // vnořený příkaz if
      if(ch < odpoved) System.out.println("blíže konci abecedy.");
      else System.out.println("blíže začátku abecedy.");
    }
  }
}
```

Toto je vnořený příkaz if.

Následuje ukázka spuštění programu:

```
Myslím si písmeno od A do Z.
Zkus je uhodnout: Z
Je mi líto, správné písmeno je blíže začátku abecedy.
```

Posloupnost if-else-if

Na vnořeném příkazu `if` je založena běžná programátorská konstrukce, která se označuje jako *posloupnost* `if-else-if`. Vypadá takto:

```
if(podmínka)
  příkaz;
else if(podmínka)
  příkaz;
else if(podmínka)
  příkaz;
.
.
.
else
  příkaz;
```

Podmínky se vyhodnocují shora dolů. Program ihned po nalezení pravdivé podmínky provede související příkaz a zbytek posloupnosti vynechá. Jestliže není pravdivá žádná podmínka, vykoná poslední příkaz `else`. Závěrečný příkaz `else` často funguje jako výchozí podmínka. To znamená, že když selžou všechny jiné podmíněné testy, provede se poslední příkaz `else`. Pokud žádný poslední příkaz `else` není uveden a všechny ostatní podmínky jsou nepravdivé, nenastane žádná akce.

Jako ukázka posloupnosti `if-else-if` slouží následující program:

```
// Toto je ukázka posloupnosti if-else-if.
class Posloupnost {
  public static void main(String args[]) {
    int x;

    for(x=0; x<6; x++) {
      if(x==1)
        System.out.println("Proměnná x se rovná jedné.");
      else if(x==2)
        System.out.println("Proměnná x se rovná dvěma.");
      else if(x==3)
        System.out.println("Proměnná x se rovná třem.");
```

```
      else if(x==4)
        System.out.println("Proměnná x se rovná čtyřem.");
      else
        System.out.println("Proměnná x nepatří do intervalu 1 až 4."); <-----
    }                                                                    Toto je výchozí příkaz.
  }
}
```

Program poskytne tento výstup:

```
Proměnná x nepatří do intervalu 1 až 4.
Proměnná x se rovná jedné.
Proměnná x se rovná dvěma.
Proměnná x se rovná třem.
Proměnná x se rovná čtyřem.
Proměnná x nepatří do intervalu 1 až 4.
```

Je zřejmé, že výchozí příkaz `else` je proveden pouze tehdy, pokud neuspěje žádný z předchozích příkazů `if`.

Příkaz switch

Dalším příkazem výběru v jazyce Java je příkaz `switch`. Příkaz `switch` poskytuje vícenásobné větvení. Umožňuje tedy, aby program volil mezi několika alternativami. Vícenásobné testy lze sice sestavit i z řady vnořených příkazů `if`, ale postup založený na příkazu `switch` je v mnoha situacích efektivnější. Funguje tímto způsobem: Hodnota výrazu se postupně porovnává se seznamem konstant. Při nalezení shody je provedena sekvence příkazů, která patří příslušné shodě. Příkaz `switch` má tento obecný tvar:

```
switch(výraz) {
  case konstanta1:
    posloupnost příkazů
    break;
  case konstanta2:
    posloupnost příkazů
    break;
  case konstanta3:
    posloupnost příkazů
    break;
.
.
.
  default:
    posloupnost příkazů
}
```

V případě verzí jazyka Java před uvedením sady JDK 7 musel být *výraz* řídící příkaz `switch` typu `byte`, `short`, `int` nebo `char`, případně výčet. (Výčty popíšeme v kapitole 12.) Počínaje sadou JDK 7 může být *výraz* také typu `String`. To znamená, že moderní verze jazyka Java dovolují

řídit příkaz `switch` pomocí řetězce. (Tento postup předvedeme v kapitole 5 v rámci výkladu typu `String`.) Příkaz `switch` je namísto většího výrazu často kontrolován pouhou proměnnou.

Každá hodnota uvedená v příkazech `case` musí být jedinečný konstantní výraz (například hodnota literálu). Duplicitní hodnoty `case` nejsou přípustné. Typ každé hodnoty musí být kompatibilní s typem *výrazu*.

Posloupnost příkazů `default` se aktivuje v případě, že výrazu neodpovídá žádná konstanta `case`. Příkaz `default` je volitelný. Pokud není uveden a žádná podmínka není pravdivá, program neprovede žádnou akci. Při nalezení shody začne program provádět příkazy přidružené k příslušnému příkazu `case` až do výskytu nejbližšího příkazu `break` nebo – v případě příkazu `default` či posledního příkazu `case` – dokud program nedosáhne konce příkazu `switch`.

Jako ukázku příkazu `switch` lze uvést následující program:

```
// Toto je ukázka příkazu switch.
class UkazSwitch {
  public static void main(String args[]) {
    int i;

    for(i=0; i<10; i++)
      switch(i) {
        case 0:
          System.out.println("Proměnná i se rovná nule.");
          break;
        case 1:
          System.out.println("Proměnná i se rovná jedné.");
          break;
        case 2:
          System.out.println("Proměnná i se rovná dvěma.");
          break;
        case 3:
          System.out.println("Proměnná i se rovná třem.");
          break;
        case 4:
          System.out.println("Proměnná i se rovná čtyřem.");
          break;
        default:
          System.out.println("Proměnná i je větší nebo rovna pěti.");
      }
  }
}
```

Tento program poskytne uvedený výstup:

```
Proměnná i se rovná nule.
Proměnná i se rovná jedné.
Proměnná i se rovná dvěma.
Proměnná i se rovná třem.
Proměnná i se rovná čtyřem.
Proměnná i je větší nebo rovna pěti.
```

```
Proměnná i je větší nebo rovna pěti.
Proměnná i je větší nebo rovna pěti.
Proměnná i je větší nebo rovna pěti.
Proměnná i je větší nebo rovna pěti.
```

Je zřejmé, že při každém průchodu cyklem se provedou příkazy, které patří ke konstantě `case` odpovídající proměnné `i`. Všechny ostatní příkazy program vynechá. Když se hodnota proměnná `i` rovná pěti nebo je větší, nevyhovuje žádný příkaz `case`, takže se spustí příkaz `default`.

Příkaz `break` technicky vzato není povinný, ale většina konstrukcí `switch` jej obsahuje. Když se příkaz `break` nachází v rámci sekvence příkazů `case`, způsobí, že program opustí celý příkaz `switch` a pokračuje provádění dalším příkazem za konstrukcí `switch`. Jestliže však posloupnost příkazů patřící příkazu `case` nekončí příkazem `break`, budou provedeny všechny příkazy *uvnitř a za* příslušným příkazem `case`, dokud program nenarazí na příkaz `break` (nebo konec příkazu `switch`).

Prohlédněte si pozorně následující program. Přijdete na to, co program vypíše na obrazovce, aniž byste se podívali na jeho publikovaný výstup?

```
// Toto je ukázka příkazu switch bez příkazů break.
class BezPrikazuBreak {
  public static void main(String args[]) {
    int i;

    for(i=0; i<=5; i++) {
      switch(i) {
        case 0:
          System.out.println("Proměnná i je menší než jedna.");
        case 1:
          System.out.println("Proměnná i je menší než dvě.");
        case 2:
          System.out.println("Proměnná i je menší než tři.");
        case 3:
          System.out.println("Proměnná i je menší než čtyři.");
        case 4:
          System.out.println("Proměnná i je menší než pět.");
      }
      System.out.println();
    }
  }
}
```

Program prochází všechny tyto příkazy case.

Program zobrazí tento výstup:

```
Proměnná i je menší než jedna.
Proměnná i je menší než dva.
Proměnná i je menší než tři.
Proměnná i je menší než čtyři.
Proměnná i je menší než pět.

Proměnná i je menší než dva.
Proměnná i je menší než tři.
```

```
Proměnná i je menší než čtyři.
Proměnná i je menší než pět.

Proměnná i je menší než tři.
Proměnná i je menší než čtyři.
Proměnná i je menší než pět.

Proměnná i je menší než čtyři.
Proměnná i je menší než pět.

Proměnná i je menší než pět.
```

Program dokládá, že není-li přítomen žádný příkaz `break`, provádění pokračuje dalším příkazem `case`.

Příkazy `case` mohou být prázdné, jak je patrné z tohoto příkladu:

```
switch(i) {
  case 1:
  case 2:
  case 3: System.out.println("Proměnná i se rovná 1, 2 či 3.");
    break;
  case 4: System.out.println("Proměnná i se rovná 4.");
    break;
}
```

Má-li proměnná `i` v tomto úseku kódu hodnotu 1, 2 nebo 3, je proveden první příkaz `println()`. Jestliže se proměnná rovná 4, spustí se druhé volání `println()`. Příkazy `case` se běžně „vrství“ stejným způsobem jako v této ukázce, když několik příkazů `case` sdílí společný kód.

Vnořené příkazy switch

Příkaz `switch` může tvořit část sekvence příkazů nebo vnějšího příkazu `switch`. Označuje se to jako vnořený příkaz `switch`. Ke konfliktům nedochází ani v případech, že konstanty `case` vnitřního a vnějšího příkazu `switch` obsahují stejné hodnoty. Například následující úsek kódu je naprosto v pořádku:

```
switch(ch1) {
  case 'A': System.out.println("Tento případ A patří vnějšímu příkazu switch.");
    switch(ch2) {
      case 'A':
        System.out.println("Tento případ A patří vnitřnímu příkazu switch.");
        break;
      case 'B': // ...
    } // konec vnitřního příkazu switch
    break;
  case 'B': // ...
```

Cvičení 3.1: Začátek tvorby systému nápovědy jazyka Java

Napoveda.java

V tomto projektu vytvoříte jednoduchý systém nápovědy, který bude informovat o syntaxi řídicích příkazů Java. Program zobrazí nabídku s řídicími příkazy a poté vyčká, dokud uživatel jeden z nich nevybere. Po výběru možnosti program vypíše syntaxi příslušného příkazu. V této první verzi programu je k dispozici pouze nabídka k příkazům `if` a `switch`. Další řídicí příkazy doplníte v následujících projektech.

1. Vytvořte soubor s názvem `Napoveda.java`.
2. Program nejdříve zobrazí následující nabídku:

```
Nápověda k příkazům:
  1. if
  2. switch
Zvolte jednu možnost:
```

   Lze to zajistit pomocí následující posloupnosti příkazů:

```
System.out.println("Nápověda k příkazům:");
System.out.println("  1. if");
System.out.println("  2. switch");
System.out.print("Zvolte jednu možnost: ");
```

3. Poté program zjistí uživatelskou volbu voláním metody `System.in.read()`:

```
volba = (char) System.in.read();
```

4. Jakmile program získá vybranou hodnotu, zobrazí pomocí následujícího příkazu `switch` syntaxi vybraného příkazu.

```
switch(volba) {
  case '1':
    System.out.println("Příkaz if:\n");
    System.out.println("if(podmínka) příkaz;");
    System.out.println("else příkaz;");
    break;
  case '2':
    System.out.println("Příkaz switch:\n");
    System.out.println("switch(výraz) {");
    System.out.println("  case konstanta:");
    System.out.println("    posloupnost příkazů");
    System.out.println("    break;");
    System.out.println("  // ...");
    System.out.println("}");
    break;
  default:
    System.out.println("Vybraná možnost není k dispozici.");
}
```