Poznámky, doplnění k předchozím tématům

„POUŽÍVEJTE PROSTŘEDKY S POROZUMĚNÍM“

Logické příkazy

Logický příkaz – úplná podmínka
Podmíněný výraz

- **Úplná podmínka**
  - Jedná se o **příkaz** – v jednotlivých větvích jsou příkazy, které mohou být zastoupeny blokem příkazů
- **Podmíněný výraz**
  - Jedná se **výraz** – po vyhodnocení má hodnotu konkrétního typu, tuto hodnotu je možné použít ve složitějším výrazu, přiřadit proměnné, vypsat

Kombinování úplných a neúplných podmínek

```
if (<podmínka1>)
    if (<podmínka2>) <příkaz1>;
else <příkaz2>;
```

- Příkaz `<příkaz2>` se provede v případě, že podmínka `<podmínka1>` má hodnotu `true` a podmínka `<podmínka2>` hodnotu `false`, větev `else` se přiřazuje nejbližšímu `if`.
- Pokud si chceme vynutit jinou interpretaci – provedení příkazu `<příkaz2>` pro `<podmínka1>==false`, potom je nutné použít blok příkazů:

```
if (<podmínka1>){
    if (<podmínka2>) <příkaz1>;
} else <příkaz2>;
```

- Obecné doporučení – používat blok příkazů v obou větvích (pro jednoznačné vymezení obdobně i v dalších příkazech `while` a `for`)

Cykly

- Jazyk Java
  - Tři různé příkazy cyklu – `while`, `do-while`, `for`
  - Všechny tři cykly – řízené podmínkou
  - Pří splnění podmínky – opakované provádění těla cyklu
  - Nesplnění podmínky – ukončení cyklu, pokračování dalším příkazem za cyklem
  - `do-while` – podmínka na konci cyklu, tělo cyklu se provede minimálně jednou
  - `for` – záhlaví cyklu obsahuje kromě podmínky i inicializační a iterační část – zápis může být kompaktnější a přehlednější
  - Všechny tři příkazy cyklu jsou navzájem zaměnitelné – všechny tři příkazy mají prakticky stejné vyjadřovací schopnosti (jedná se o stejně silné nástroje) – libovolný příkaz cyklu můžeme zapsat pomocí libovolného jiného příkazu cyklu

- Při používání cyklů – **nutno zabránit vzniku nekonečného cyklu**
  - Příkazy cyklu Java – řízené podmínkou
  - Pokud vykonávání kódu vstoupí do těla cyklu (podmínka cyklu nabyla hodnoty `true`) – příkazy v těle cyklu vi konečném počtu opakování musí měnit/ovlivňovat hodnotu podmínky cyklu, tak aby tato podmínka v konečném počtu kroků nabyla hodnoty `false`

Standardní výstup – formátovaní výstupu

- Metody `print()`, `println()` mají omezené možnosti formátování výpisu
- Komfortnější formátování hodnot do textových řetězců umožňuje metoda `format()`, opět se jedná o metodu výstupního proudu `out` třídy `System`

```
System.out.format(<formatovaci retezec>, <hodnoty>)
```

- `<formatovaci retezec>` je konstantní textový řetězec, který může obsahovat parametry, parametry začínají znakem %, pro každý parametr (až na výjimky) musí být za formátovacím textovým řetězcem uvedena vypisovaná hodnota, která se při výpisu naformátuje dle specifikovaného formátu do vypisovaného textového řetězce. Hodnoty v seznamu `<hodnoty>` jsou navzájem odděleny čárkou, pořadí a typ musí odpovídat pořadí a typu parametrů ve formátovacím řetězci. Každá hodnota je v obecném případě dána výrazem (proměnná, konstanta, výraz), jehož hodnota musí být typově kompatibilní s typem určeným příslušným parametrem.

```
int i = 1483;
System.out.format("Cislo: %d",i);
```
// celé číslo v desítkové soustavě
```
System.out.format("Cislo: %o",i);
```
// celé číslo v osmičkové soustavě
```
System.out.format("Cislo: %x",i);
```
// celé číslo v šestnáctkové soustavě
```
System.out.format("Cislo: %X",i);
```
// celé číslo v šestnáctkové soustavě

```
System.out.format("Cislo: %6d",i);
```
// celé číslo s určením počtu min. pozic
```
System.out.format("Cislo: %-6d",i);
```
// určení počtu pozic, zarovnání vlevo
```
System.out.format("Cislo: %+d",i);
```
// s výpisem kladného znaménka
```
System.out.format("Cislo: %06d",i);
```
// s výpisem nevýznamných nul
```
System.out.format("Cislo: %,6d",i);
```
// s oddělovačem řádů

- Formátování znakové hodnoty – parametr %c
- Formátování textového řetězce –parametr %s
- Obdobně jako u čísel lze zadat počet pozic a zarovnání formátované hodnoty vlevo

```
double d = 1960.00467;
System.out.format("Cislo: %f",d);
```
// reálné číslo, běžný výpis, oddělovač desetinných míst závisí na lokalizaci
```
System.out.format("Cislo: %g",d);
```
// výpis buď ve vědecké notaci nebo jako běžné reálné číslo, oddělovač desetinných míst je vždy tečka
```
System.out.format("Cislo: %e",d);
```
// výpis reálného čísla ve vědecké notaci

Při výpisu reálných čísel lze určit celkový počet pozic a počet desetinných míst
```
System.out.format("Cislo: %15.4f",d);
```
// počet pozic, desetinných míst

Obdobně – zarovnání vlevo, vynucení znaménka, nevýznamných nul, oddělovač řádů
```
System.out.format(" %-15.4f", d);
System.out.format(" %+15.4f", d);
System.out.format(" %015.4f", d);
System.out.format(" %,15.4f", d);
```

- Ukončení aktuální řádky výstupního proudu – formátovací parametr %n
- Pořadí hodnoty <poradi>$ – %2$X
- Reálné číslo představující datum a čas a jeho formátování

Standardní vstup – zpracování vstupu

- Metody instance/objektu typu `Scanner`
- Čtení celého čísla typu `int` – metoda `nextInt()`
- Čtení reálného čísla typu `double` – `nextDouble()`
- Další metody – `nextByte(), nextShort(), nextLong(), nextFloat(), nextBoolean(),`

- Čtení celé řádky do textového řetězce
  `sc.nextLine()`

- Čtení znaku – není k dispozici metoda pro načtení jediného znaku – lze využít/separovat první znak zadaného textového řetězce
  `next().charAt(0)`
  `nextLine().charAt(0)`

- Metody vrací načtenou hodnotu svým jménem
- Způsob zpracování vstupu – metoda `nextLine()` a ostatní metody
- Kombinace načítání čísel a znaků, čísel a textových řetězců

- Scanner – podporuje lokalizaci – lokalizace
  `useLocale(Locale.US) //` použití předdefinované instance třídy `Locale`
  `useLocale(new Locale("cs", "CZ")) //` vytvoření nové instance třídy `Locale` pro české národní prostředí
  (poznámka, `System.out` nemá přímo prostředky pro lokalizaci)

Použití hodnot proměnných

- Hodnotu proměnné lze použít pouze v případě, že jí předtím byla přiřazena hodnota
  - Použití hodnoty proměnné např. použití ve výraze
  - Jazyk Java – použití proměnné s doposud nedefinovanou hodnotou – syntaktická chyba
  - Implicitní inicializace proměnných
    - Lokální proměnná (proměnné deklarované v metodách) nejsou implicitně inicializovány, po přidělení paměti hodnoty proměnných nejsou definovány
    - Atributy tříd, položky polí – implicitní inicializace paměti přidělené po použití `new` – vynulování obsahu přidělené paměti

- Přiřazení hodnoty proměnné
  - Operace přiřazení

- Použití předchozí hodnoty proměnné je skryto i v následujících výrazech
  - Inkrementace `i++` tj. v podstatě `i = i + 1`
  - Přiřazení `s *= x` `s = s * x`

Aritmetika celých a reálných čísel

Celá čísla

- Celočíselný typ – interval celočíselných hodnot zobrazitelných daným typem
- Každá hodnota příslušného typu – v paměti počítače zobrazena přesně
- Porovnání hodnot
- Aritmetické operace – může docházet k přetečení
  - Aritmetické operace – *sčítání, odčítání, násobení, dělení, modulo* – pokud je jeden operand typu `long`, výsledek typ `long`, jinak je výsledek typu `int`

Reálná čísla

- Reálný typ – interval hodnot zobrazitelných daným typem
- Hodnota reálného typu – v paměti není zobrazena přesně – garantovaný počet platných číslic
- *Mantisa, exponent* – omezení mantisy – zaokrouhlování
- **Problém práce s reálnými čísly je nepřesnost**
- Nelze se spoléhat na to, že výsledkem dvou reálných výrazů představujících stejnou hodnotu budou dvě totožná čísla
- Porovnání hodnot
  - Např. testování rovnosti – testujeme, zda daná čísla jsou „skoro stejná“ – s určitou přesností
  - Test `if (x == y) {…}`
  - Můžeme nahradit testem

```
if (Math.abs(x - y) < skoroNula) {…}
if (jsouSkoroStejne(x, y)) {…}
```

Aritmetika reálných čísel

- Jaký očekáváme výsledek (hodnoty vypsané na výstupu) při použití následující konstrukce

```
float f = 100.0F;
System.out.println(f);
for (int i = 0; i < 10; i++) {
  f += 0.1;
  System.out.println(f);
}
```

- ?? Rádo by

```
100.0
100.1
100.2
100.3
100.4
100.5
100.6
100.7
100.8
100.9
101.0
```

- ?? Může to být něco jako

```
100.0
100.1
100.2
100.299995
100.399994
100.49999
100.59999
100.69999
100.79999
100.89999
100.999985
```

Výsledek aritmetické operace, přetečení rozsahu

Celá čísla

- Jaký očekáváme výsledek na výstupu

```
int a = Integer.MAX_VALUE;
int b = 1;
long velkeCislo;
velkeCislo = a + b;
System.out.println(a + "" + b);
System.out.println(velkeCislo);
```

- ?

```
2147483647 + 1
2147483648
```

- Ve skutečnosti obdržíme

```
2147483647 + 1
-2147483648
```

- Důvodem je přetečení typu `int`
- Celočíselné operace a jejich výsledek – pokud je jeden z operandů `long` výsledek je `long`, jinak je výsledek `int`,
- Zkuste `s = a + b` , kde `s, a, b` jsou typu `byte`
- Dělení nulou v celých číslech - chyba

Reálná čísla

- Výsledek operace je typu `double`, v případě, že alespoň jeden z operandů je typu `double`
- Vzhledem k rozsahu hodnot není přetečení až tak obvyklým problémem – o to více může někdy potrápit
- Výsledkem přetečení jsou opět konkrétní hodnoty toho, kterého typu
  - Přetečení maximální kladné hodnoty typu – výsledek je `POSITIVE_INFINITY` příslušného typu
  - Přetečení maximální záporné hodnoty typu – výsledek je `NEGATIVE_INFINITY`
  - Přetečení hodnoty s maximálním záporným exponentem (jak v kladných, tak záporných číslech) – výsledek je `0.0` (respektive `-0.0` v záporných číslech)
- Na reálná čísla můžeme aplikovat všechny aritmetické operátory (`+ - * / % ++ --`)
- Dělení nulou v oboru reálných čísel – výsledek `NaN`, `POSITIVE_INFINITY` nebo `NEGATIVE_INFINITY` příslušného typu

Pole

- Uložení pole v paměti
- Vícerozměrná pole a jejich uložení
- Dvourozměrné pole
  - Pole vektorů
  - Vektory stejné délky
  - Vektory různé délky
- Přiřazení mezi proměnnými typu pole
- Vzájemné porovnání dvou polí
  - Aplikace relačních operátorů na proměnné typu pole
  - Jak porovnat obsah dvou polí
- Pole jako parametr metod
  - Parametry metod jsou předávané hodnotou
  - V případě pole je „předána hodnota reference“

- Metody s libovolným počtem parametrů

```
double soucet(double... a) {
  double vysledek = 0;
  for (double x: a) {
    vysledek += x;
  }
  return vysledek;
}

...

System.out.println(soucet(1,2,3,4,5));
double[] pole = {1,2,3,4,5};
System.out.println(soucet(pole));
```

- Význam formálního parametru a v metodě – pole
- Skutečné parametry
  - Výčet hodnot příslušného typu
  - Pole hodnot příslušného typu

Funkční programy, formální zápis

- Co má být výsledkem práce programátora
  - Postačí funkční program???
    - Specifikace zadání
    - **Řešení – program, který poskytuje správné výsledky pro všechny kombinace vstupů definované zadáním**
    - Ošetření chybových stavů
    - Ošetření chybného vstupu/zadání od uživatele
  - A co přehlednost, srozumitelnost, modifikovatelnost, efektivita kódu, členění kódu, paměťová náročnost???

- Přehlednost, srozumitelnost, modifikovatelnost
- Je formální zápis důležitý a proč?
  - Pojmenování proměnných, konstant, metod, tříd, balíků
  - Odsazování – zarovnání
  - Příkazový blok
  - Umístění `else`
  - Umístění `while` cyklu `do-while`

- Členění kódu – strukturované programování, objektové programování
- Efektivita kódu – *“We should forget about small efficiencies, say about 97% of the time: Premature optimization is the root of all evil.”—Donald Knuth*
- Paměťová náročnost

Chyby, testování, ladění

- Chyby syntaktické
  - Syntaktický analyzátor
  - V době kompilace
  - NetBeans – zvýraznění v editoru – již v průběhu psaní kódu je prováděna syntaktická analýza
- Chyby běhové
  - Dělení nulou v oboru celých čísel
  - Převod textových řetězců na čísla
  - Chyba – přerušení vykonávání kódu na daném místě – generování výjimky – výjimku lze v programu ošetřit
  - Pokud zůstane výjimka neošetřena, program vypíše standardní chybové hlášení a skončí
  - V případě některých byť chybných operací k chybě nedochází – přetečení rozsahu typu
- Chyby sémantické
  - Program lze přeložit, spustit,
  - Pro zadaná vstupní data (popřípadě pro některé kombinace zadaných vstupních dat) poskytuje chybné výsledky
- Ladění programu
  - Podpůrné prostředky v používaném vývojovém prostředí
  - Testování programu, testování jednotlivých fragmentů kód (metod), sada testů (testovacích úloh) by měla být úplná