Algoritmizace a programování

Výrazy

- Operátory
- Výrazy

Verze pro akademický rok 2012/2013

Operace, operátory

- **Unární** – jeden operand, operátor se zapisuje ve většině případů před operand, v některých případech následuje operátor až za operandem
  `<operator><operand>` nebo `<operand><operator>`
- **Binární** – dva operandy, jazyk Java používá **infixový** zápis, operátor je mezi oběma operandy, jiné možné způsoby jsou prefixový (operátor je zapsán před operandy, prefixový způsob zápisu používá např. jazyk Lisp) a sufixový respektive postfixový (operátor je zapsán za operandy)
  `<operand1><operator><operand2>`
- **Ternární** – má tři operandy, v jazyce Java je ternární logický operátor
  `<operand1><operator1c><operand2><operator2c><operand3>`
- Výsledkem operace je hodnota – typ výsledné hodnoty závisí obecně na příslušné operaci a na operandech (na typu operandů)
- Konstrukce výrazů – výraz po vyhodnocení má hodnotu
- Výraz ukončený středníkem je příkazem

Operátor přiřazení

- Operátor přiřazení je v jazyce Java reprezentován znakem `=` , operaci přiřazení zapisujeme

  `<promenna> = <vyraz>`

  - `<promenna>` první operand musí být identifikátor proměnné
  - `<vyraz>` může být reprezentován konstantní hodnotou, proměnnou, konstantou nebo obecně výrazem
  - `<vyraz>` hodnota výrazu na pravé straně přiřazení musí být typově kompatibilní vzhledem k přiřazení s typem proměnné, která je na levé straně operátoru přiřazení
- Operace přiřazení má výslednou hodnotu rovnou přiřazené hodnotě
- **Příkaz přiřazení** – ukončení operace středníkem

- Příklady příkazů přiřazení (obsahují operaci přiřazení ukončenou středníkem)

```
int a,b,c;
a = 4;
b = 2 * a;
c = b = a;
```

```
int a, b, c;
a = 12;
b = 20;
c = a; a = b; b = c;
```

Aritmetické operátory

Binární aritmetické operátory

- Použitelné na operandy číselných typů
  - **+** součet
  - **-** rozdíl
  - ***** násobení
  - **/** dělení reálné, dělení celočíselné
  - **%** zbytek po celočíselném dělení, dělení modulo

- Operandy jsou v obecném případě výrazy s hodnotou číselného typu, výsledná hodnota je dána odpovídající aritmetickou operací

- Chybová hlášení
  - Při dělení nulou
  - Aplikací operátorů na operandy nepřípustných typů

- Typ výsledné hodnoty operací `+, -, *, /` **závisí na typu operandů**, pokud je jeden z operandů reálný, potom je i výsledek reálný, v opačném případě je výsledek celočíselný

Aritmetické operátory

Použití binárních aritmetických operátorů

```
float b = 3.2f;
float a = 25;
float c, d;
c = a – b;
d = a / b;

int m = 25 / 6;
int n = 25 % 6;

d = 31 / 7;

d = a / 6;
```

- Nutno určit typ konstantní hodnoty
- Není nutno specifikovat typ float
  - viz konverze dále v prezentaci
- Výsledek 21.8
- Výsledek 7.8125

- Výsledek 4 – celočíselné dělení
- Výsledek 1 – zbytek po celočíselném dělení
- Výsledek 4 – celočíselné dělení, výsledek je přiřazen do proměnné reálného typu
- Výsledek je 4.1666665 – reálné dělení, jeden operand je reálný

Aritmetické operátory

Unární aritmetické operátory

- Použitelné na operandy číselných typů
  - **+** unární plus
  - **-** unární mínus
  - **++** inkrementace hodnoty operandu, prefixový i postfixový zápis
  - **--** dekrementace hodnoty operandu, prefixový i postfixový zápis
    - Operandem v inkrementaci, dekrementaci může být pouze **proměnná celočíselného typu**
    - Prefixový zápis – hodnota proměnné je inkrementována před použitím
    - Postfixový zápis – hodnota proměnné je inkrementována po použití
    - Výsledkem operace hodnota, vedlejším efektem přiřazení
    - `++i;  i = i + 1;` `--i;  i = i – 1;`

- Příklady

```
int a = +5;
int min = -164;
```

```
int k = 7;
++k;
int l = k++;
l = ++k;
```

Relační operátory

- Relační operátory jsou binární operátory, slouží k porovnání hodnot operandů
- Konstrukce logických výrazů
- Operandy mohou být libovolného typu, oba operandy musí být typově kompatibilní
- Výsledkem operace je hodnota logického typu (`true` nebo `false`)
- Operátory
  - **==** porovnání rovnosti
  - **!=** nerovnost
  - **<** menší
  - **<=** menší nebo rovno
  - **>** větší
  - **>=** větší nebo rovno

- Pozor na porovnávání ostré rovnosti, nerovnosti reálných čísel, porovnáváme „přibližnou“ rovnost

```
4 == 5  // false          4 < 5    // true
4 != 5  // true           4 >= 5   // false
```

Logické operátory

- Z logických operátorů je jediný operátor unární ostatní jsou binární
- Operandy jsou logického typu, výsledek logického typu (`true`, `false`)
- Konstrukce logických výrazů
  - **`!`** negace, NOT, unární operátor, zapisuje se před operand
  - **`&&`** logický součin AND se zkráceným vyhodnocováním
  - **`||`** logický součet OR se zkráceným vyhodnocováním
  - **`&`** logický součin AND s úplným vyhodnocováním
  - **`|`** logický součet OR s úplným vyhodnocováním
  - **`^`** logický výlučný součet XOR

- Operátory `&, | , ^` se používají i jako bitové operátory
- Pravdivostní tabulka jednotlivých funkcí
- Příklad

```
(a > 3) & (a < 10)
```

Logické operátory

- Pravdivostní tabulka logických funkcí

x	y	!x	x | y	x & y	x ^ y
false	false	true	false	false	false
false	true	true	true	false	true
true	false	false	true	false	true
true	true	false	true	true	false

- Úplné a neúplné vyhodnocování logických výrazů

  `(x > 4) & (x < 8) & (y > 3) & (y < 6)` – v případě, že není splněna první část této podmínky (pro `x <= 4`) má celý výraz hodnotu `false`, není nutno vyhodnocovat další tři základní podmínky výrazu

  `(x < 6) | (y > 12)` – v případě, že první část podmínky (`x < 6`) má hodnotu `true`, celý výraz má hodnotu `true` bez ohledu na hodnotu druhé části výrazu

  - Jazyk Java má k dispozici dva druhy logických operátorů pro operace logického součtu a logického součinu, které zajistí buď úplné nebo neúplné vyhodnocování obdobných logických výrazů

Bitové operace

- Při bitových operacích se provádějí operace s jednotlivými bity operandů
- Operandy nesmějí být typu `float` nebo `double`
  - **&** bitový součin AND
  - **|** bitový součet OR
  - **^** bitový exkluzivní součet XOR
  - **<<** bitový posun vlevo
  - **>>** bitový posun vpravo
  - **>>>** neznaménkový posun vpravo
  - **~** jedničkový doplněk, unární operátor

Operátor zřetězení

- Operátor zřetězení je reprezentován znakem **+**
- Dříve ukazován u konstantních textových řetězců
- Binární operátor
- Alespoň jeden z operandů je textový řetězec, pokud druhý z operandů není textový řetězec, je na textový řetězec konvertován

```
int a = 5;
int b = 12;
System.out.println("Hodnota prvniho cisla " + a);
System.out.println("Obe cisla za sebou " + a + b);
System.out.println(a + b + " obe cisla jinak");
```

Operátory přiřazení

- Jazyk Java má další operátory přiřazení
  - **+=** `<promenna> += <vyraz>` `<promenna> = <promenna> + <vyraz>`
  - **-=**
  - ***=** obdobně ostatní zde vyjmenované operátory přiřazení
  - **/=**
  - **%=**
  - **&=**
  - **^=**
  - **|=**
  - **>>=**
  - **<<=**
  - **>>>=**

Operátor přetypování

- Přetypování využijeme máme-li hodnotu nějakého typu a potřebujeme z ní „vytvořit“ hodnotu jiného typu (například z `float` na `int`)

```
(<typ>) <vyraz>
```

- Výsledkem je hodnota výrazu `<vyraz>` reprezentovaná jako hodnota typu `<typ>`
- Přetypování má nejvyšší prioritu. Pokud chceme přetypovat výslednou hodnotu výrazu, musíme výraz vložit do závorek.

```
double d = 15.76E-3;
float f = (double) d;
float x = (double) (2 * f);
```

Rozšiřující a zužující konverze

- **Rozšiřující konverze** jazyk Java provádí implicitně, není nutné expl. přetypování

  `byte` → `short` → `int` → `long` → `float` → `double`

  `char` → `int`

- Při těchto konverzích nedochází ke ztrátě informace (poznámka: ke ztrátě informace může dojít při konverzi `long` na `double` nebo `int`, `long` na `float`)

- **Zužující konverze** se neprovádějí implicitně, pokud takovou konverzi potřebujeme je nutné využít operátor přetypování

  `double` → `float` → `long` → `int` → `short` → `byte`

  `int` → `char`

- Při zužujících konverzích může dojít ke ztrátě informace nebo změně hodnoty

```
float f= 13.74f;
System.out.println((int)f);    //13
int i = 129;
System.out.println((byte) i);  //-127
```

Postup vyhodnocování operací

- Operace přiřazení jsou vyhodnocovány zprava doleva

  `a = b = c = d`
- Ostatní operace jsou vyhodnocovány zleva doprava

  `a + b + c + d`

- Operace jsou vyhodnocovány s ohledem na prioritu – operace s vyšší prioritou je vyhodnocena dříve

  `a + b * c + d`

- Chceme-li změnit implicitní pořadí vyhodnocování použijeme okrouhlé závorky
- Okrouhlé závorky musí být párové a mohou být libovolně vnořené

Priority operátorů

Operátory jsou uvedeny **od nejvyšší priority k nejnižší**

- `(typ)  []  .` přetypování, indexování, reference
- `!  ++  --  -  +  ~` unární operátory
- `*  /  %` multiplikativní – násobení a dělení
- `+  -` aditivní – sčítání a odčítání
- `<<  >>  >>>` bitový posun
- `<<  >>  >>>` bitový posun
- `<  >  <=  >=` relační operátory
- `==  !=` relační operátory rovno, nerovno
- `&` logický i bitový součin AND
- `^` logický i bitový XOR
- `|` logický i bitový součet OR
- `&&` logický zkrácený součin
- `||` logický zkrácený součet
- `?:` logický ternární operátor
- `=  +=  -=  *=` operátory přiřazení
- `,` operátor čárka v příkazu `for`

Konstrukce výrazů

- Výraz – série operací
- Obsahuje konstantní hodnoty, proměnné, operátory, volání metod

- Pro výpočet matematických funkcí ve výrazech voláme (používáme) metody třídy `Math`,
  Třída `Math` je umístěna v balíčku `java.lang`.
  Dále jsou uvedeny některé metody třídy `Math`, volání metod vrací (poskytuje) hodnotu:
  - Pro výpočet druhé odmocniny z hodnoty dané výrazem `<vyraz>` použijeme
    `Math.sqrt(<vyraz>)`
  - Výpočet obecné mocniny
    `Math.pow(<vyraz1>, <vyraz2>)`
  - Výpočet absolutní hodnoty čísla
    `Math.abs(<vyraz>)`
  - Pro výpočet goniometrických funkcí použijeme
    `Math.sin(<vyraz>)`
    `Math.cos(<vyraz>)`
  - Výpočet exponenciální funkce
    `Math.exp(<vyraz>)`
  - Výpočet logaritmu
    `Math.log(<vyraz>)` přirozený logaritmus
    `Math.log10(<vyraz>)` dekadický logaritmus
  - Hodnota Ludolfova čísla
    `Math.PI`

Reálné výrazy

- Převod stupňů na radiány

```
alfaRad = alfaGrad * Math.PI / 180;
alfaRad = (alfaGrad * Math.PI) / 180;
alfaRad = alfaGrad / 180 * Math.PI;
alfaRad = Math.PI / 180 * alfaGrad;
```

- Převod radiánů na stupně

```
alfaGrad = alfaRad * 360 / (2 * Math.PI);
alfaGrad = alfaRad * 360 / 2 / Math.PI;
alfaGrad = alfaRad / 2 / Math.PI * 360;
```

Celočíselné výrazy

- Pořadí prvků v dvourozměrné tabulce v případě, že máme daný číslo řádku a číslo sloupce prvku v tabulce

```
poradi = (r-1) * ps + s;
```

- Naopak máme dáno pořadí prvku v dvourozměrné tabulce, chceme zjistit do kterého řádku a do kterého sloupce prvek patří

```
r = (poradi – 1) / ps + 1;
s = (poradi – 1) % ps + 1;
```

- Řešení obou úloh závisí na celkovém počtu řádků a sloupců tabulky a na tom, zda tabulka má řádky a sloupce číslované od 0 nebo od 1 a také na tom, jak je chápáno pořadí prvku v tabulce – zda postupně po řádcích nebo po sloupcích. Uvedené výrazy jsou platné pro číslování od jedné a procházení tabulky po řádcích a velikost tabulky `pr`, `ps`

Celočíselné výrazy

- Inkrementace hodnoty proměnné o 1

```
i = i + 1;
i += 1;
i++;
++i;
```

- Inkrementace hodnoty proměnné o 2

```
i = i + 2;
```

- Cyklická inkrementace a dekrementace hodnoty proměnné v rozsahu od 1 do `N`

```
index = (index % N) + 1;
index = (index + N - 2) % N + 1;
```

Logické výrazy

- Změna hodnoty logické proměnné na opačnou
  `b = !b`
- Test dělitelnosti
  `cislo % d == 0`
- Test čísla v intervalu
  `(x > -3.4)  & (x <= 11.2)`
- Test znaku na číslici
  `c >= ‘0' & c <= '9'`
- Test znaku na písmeno anglické abecedy
  `c >= 'A' & c<= 'Z' | c >='a' & c<= 'z‘`
  `((c >= 'A‘) & (c<= 'Z‘)) | ((c >='a‘) & (c<= 'z‘))` totéž, pouze jinak zapsané
  Pro testování znaků na číslice a písmena budeme potom výhradně používat prostředky třídy `Character`

Úlohy k procvičování

- Co se vypíše na terminálový výstup a jaká bude hodnota proměnné n po provedení následujících příkazů

```
int n = 1;
System.out.println(n);
System.out.println(++n);
System.out.println(n++);
System.out.println(n);
System.out.println(++n + n);
System.out.println(n++ + n);
System.out.println(++n + ++n);
System.out.println(++n + n++);
System.out.println(n++ + ++n);
System.out.println(n);
System.out.println(-++n);
System.out.println(---n);
```

- Zapište všechny možné způsoby dekrementace hodnoty proměnné o 1
- Zapište všechny možné způsoby inkrementace hodnoty celočíselné proměnné o 5

- Zapište výrazy pro cyklickou inkrementaci a dekrementaci celočíselné proměnné v rozsahu od 0 do N-1