Seminář Java

Příkazy a pole v Javě

Radek Kočí

Fakulta informačních technologií VUT

Únor 2008

Téma přednášky

- Příkazy
- Operátory
- Dokumentace
- Pole

Příkazy v Javě

- volání metody
- příkaz je ukončen středníkem (`;`)
- přiřazovací příkaz (`=`)
- řízení toku programu
- návrat z metody (`return`)

Přiřazení

Přiřazení **=**

- Na levé straně musí být proměnná.
- Na pravé straně musí být *přiřaditelný* výraz.

Primitivní typy

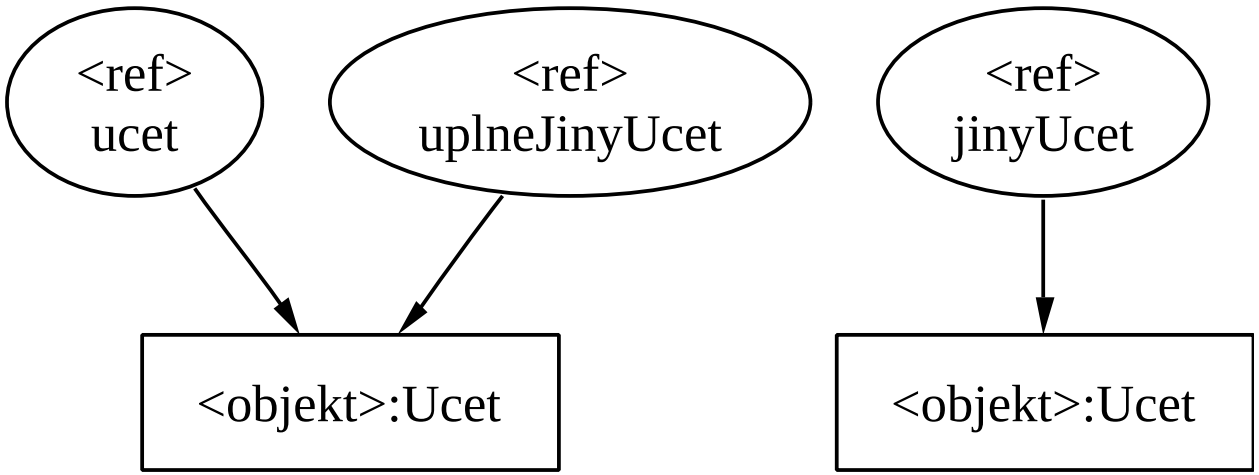
- přiřazením se hodnota zduplikuje
- konverze typů (`short` → `int`, `int` → `short`)

Přiřazení odkazu na objekt

- Proměnné objektového typu obsahují odkazy (reference) na objekty, ne objekty samotné!!!
- přiřazením se duplikuje pouze reference

Přiřazení proměnné objektového typu

```
public class Banka {
    public static void main(String[] args) {
        Ucet ucet = new Ucet();
        Ucet jinyUcet = new Ucet();
        Ucet uplneJinyUcet = ucet;
    }
}
```



Řídicí příkazy

- if
- while
- do – while
- for
- switch
- break, continue

Příkazy

Příkazy mohou být jednoduché

- `pole[i] = 20;`

nebo složené

- `{ pole[i] = 20; i++; }`

Řízení toku programu v těle metody

Příkaz (neúplného) větvení if

`if (logický výraz) příkaz`

- platí-li logický výraz (má hodnotu true), provede se příkaz

---

Příkaz úplného větvení if - else

```
if (logický výraz)
   příkaz1
else
   příkaz2
```

- platí-li logický výraz (má hodnoty `true`), provede se `příkaz1`
- neplatí-li, provede se `příkaz2`
- větev else se nemusí uvádět
- větvení `if -- else` můžeme vnořovat do sebe

Cyklus s podmínkou na začátku

- Tělo cyklu se provádí tak dlouho, dokud platí podmínka

v těle cyklu je jeden jednoduchý příkaz ...

```
while (podmínka)
    příkaz;
```

... nebo příkaz složený

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

- Tělo cyklu se nemusí provést ani jednou – pokud už hned na začátku podmínka neplatí

Cyklus s podmínkou na konci

- Tělo se provádí dokud platí podmínka (vždy aspoň jednou).
- Relativně málo používaný – je méně přehledný než `while`

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

Cyklus ”for”

- de-facto jde o rozšíření while, lze jím snadno nahradit

```
for (počáteční operace; vstupní podmínka;
          příkaz po každém průchodu)
{
   příkaz1;
   příkaz2;
   ...
}
```

Příklad použití ”for” cyklu

```
for (int i = 0; i < 10; i++) {
    System.out.println(i);
}
```

Ekvivalent s while:

```
int i=0;
while (i < 10) {
    System.out.println(i);
    i++;
}
```

Vícecestné větvení ”switch - case - default”

- Větvení do více možností na základě ordinální hodnoty

```
switch(výraz) {
   case hodnota1: prikaz1a;
                  prikaz1b;
                  break;
   case hodnota2: prikaz2a;
                  ...
                  break;
   default:       prikazDa;
                  ...
}
```

- 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.

Příkaz ”break”

- Realizuje ”násilné” ukončení průchodu cyklem nebo větvením switch
- Syntaxe použití break v cyklu:

```
for (int i = 0; i < a.length; i++) {
   if(a[i] == 0) {
      break; // skoci se za konec cyklu
   }
}
if (a[i] == 0) {
   System.out.println("0 je na pozici "+i);
} else {
   System.out.println("0 v poli neni");
}
```

Příkaz ”continue”

- Používá se v těle cyklu.
- Způsobí přeskočení zbylé části průchodu tělem cyklu.
- Běh pokračuje další iterací.

```
for (int i = 0; i < a.length; i++) {
   if (a[i] == 5)
      continue;
   System.out.println(i);
}
```

Operátory a výrazy

- Aritmetické
- Logické
- Relační
- Bitové
- Operátor podmíněného výrazu ? :
- Relační operátory

Aritmetické operátory

- `+`, `-`, `*`, `/` a `%` (zbytek po celočíselném dělení)
- platí podobná pravidla jako v C/C++
  - `int / int` $\Rightarrow$ `int`
  - `double / int` $\Rightarrow$ `double`
  - `short / int` $\Rightarrow$ `int`

Logické operátory

- logické součiny (AND):
  - & (nepodmíněný - vždy se vyhodnotí oba operandy),
  - && (podmíněný - líné vyhodnocování - druhý operand se vyhodnotí, jen nelze-li o výsledku rozhodnout z hodnoty prvního)
- logické součty (OR):
  - | (nepodmíněný - vždy se vyhodnotí oba operandy),
  - || (podmíněný - líné vyhodnocování - druhý operand se vyhodnotí, jen nelze-li o výsledku rozhodnout z hodnoty prvního)
- negace (NOT):
  - !

Bitové operátory

Bitové:

- součin &
- součet |
- exkluzivní součet (XOR) ^ (znak ”stříška”)
- negace (bitwise-NOT) ~ (znak ”tilda”)

Posuny:

- vlevo << o stanovený počet bitů
- vpravo >> o stanovený počet bitů s respektováním znaménka
- vpravo >>> o stanovený počet bitů bez respektování znaménka

Operátor podmíněného výrazu ? :

Podmíněný výraz

- Jediný ternární operátor
- Platí-li první operand (má hodnotu true) $\Rightarrow$
  - výsledkem je hodnota druhého operandu
  - jinak je výsledkem hodnota třetího operandu
- Typ prvního operandu musí být boolean, typy druhého a třetího musí být přiřaditelné do výsledku.

```
if (a > b)
   c = a - b;
else
   c = b - a;
```

```
c = (a > b ?  a - b :  b - a);
```

Relační (porovnávací) operátory

- Tyto lze použít na porovnávání primitivních hodnot:
  - `<`, `<=`, `>=`, `>`
- Test na rovnost/nerovnost lze použít na porovnávání primitivních hodnot i objektů:
  - `==`, `!=`
  - pozor na srovnávání floating-points čísel na rovnost: je třeba počítat s chybami zaokrouhlení; místo porovnání na přesnou rovnost raději používejme jistou toleranci: `abs(expected-actual) < delta`
  - pozor na porovnávání objektů: `==` vrací `true` jen při rovnosti odkazů, tj. jsou-li objekty identické!

Komentáře

Základní typy komentářů (podobně jako např. v C/C++)

- *řádkové* od značky `//` do konce řádku
- *blokové* (na libovolném počtu řádků) začínají `/*` pak je text komentáře, končí `*/`
- *dokumentační* (na libovolném počti řádků) od značky `/**` po značku `*/` Každý další řádek může začínat mezerami či *, hvězdička se v komentáři neprojeví.

Komentáře

```
// řádkový komentář

/*
   blokový
   (víceřádkový) komentář
*/

/**
   dokumentační
   (víceřádkový) komentář
*/
```

Generování dokumentace

Dokumentace

- je generována nástrojem `javadoc`
  - z dokumentačních komentářů
  - a ze samotného zdrojového textu
- *je tedy možné dokumentovat (základním způsobem) i program bez vložených komentářů!*
- má standardně podobu HTML stránek (s rámy i bez)
- chování `javadoc` můžeme změnit volbami (options) při spuštění

Dokumentační komentáře uvádíme:

- před hlavičkou třídy (komentuje třídu jako celek)
- před hlavičkou metody nebo proměnné (komentuje příslušnou metodu nebo proměnnou)

Značky pro dokumentační komentáře

Nástroj `javadoc` můžeme podrobněji instruovat pomocí značek vkládaných do dokumentačních komentářů, např.:

@author	specifikuje autora API/programu
@version	označuje verzi API, např. ”1.4.2”
@deprecated	informuje, že prvek je zavrhovaný
@exception	popisuje informace o výjimce, kterou metoda propouští (”vyhazuje”)
@param	popisuje jeden parametr metody
@since	uvedeme, od kdy (od které verze pg.) je věc podporována/přítomna
@see	uvedeme odkaz, kam je také doporučeno nahlédnout (související věci)

Ukázka použití dokumentačních komentářů

```
package ija1.ucty;

/**
 * Trida ucet
 * @author R. Koci
**/
public class Ucet {
    /** Majitel uctu */
    protected String majitel;
    protected double zustatek;

    public void pridej(double castka) {
        zustatek += castka;
    }
    ...
}
```

Pole

Pole

- Pole v Javě je speciálním objektem.
- Můžeme mít pole jak primitivních, tak objektových hodnot
  - pole primitivních hodnot tyto hodnoty obsahuje
  - pole objektů obsahuje odkazy na objekty
- Kromě pole v Javě existují i jiné objekty na ukládání více hodnot – *kontejnery* (bude později ...)

Pole – I

Před použitím je nutné pole

- deklarovat
- vytvořit
- inicializovat (naplnit)

Syntaxe deklarace

- **`typ [] identifikator`**
- na rozdíl od C/C++ nikdy neuvádíme při deklaraci počet prvků pole – ten je podstatný až při vytvoření objektu pole

Pole – II

Vytvoření pole

- jako u jiného objektu – voláním konstruktoru:
  - **`nazevPole = new typ[velikost];`**
  - **`int[] pole = new int[10];`**
- nebo inicializací při deklaraci:
  - **`int[] nazevPole = { 1, 2, 3 };`**

Syntaxe přístupu k prvkům

- **`pole[i]`**
- **`pole[i] = 20;`**
- **`int j = pole[2];`**

Pole – III

```
Ucet [] ucty;                    // deklarace pole
ucty = new Ucet[5];              // vytvoření pole

// vytvoření objektu
// a inicializace 1.  prvku pole
ucty[0] = new Ucet("Franta");

ucty[0].vypisInfo();  // přístup k prvku pole
```

- V poli `ucty` je naplněn 1. prvek odkazem na objekt
- Ostatní prvky zůstaly naplněny prázdnými odkazy `null`.

Pole – IV

Co když vynecháme vytvoření pole?

```
Ucet [] ucty;
ucty[0] = new Ucet("Franta");
   // chyba, pole neexistuje
```

Co když vynecháme inicializaci pole?

```
Ucet [] ucty;
ucty = new Ucet[5];
ucty[0].vypisInfo();
   // chyba, prvek neexistuje
```

Kopírování pole

Přiřazení proměnné objektového typu (a tedy i polí) vede pouze k duplikaci odkazu, nikoli celého odkazovaného objektu.

```
Ucet [] ucty = new Ucet[5];
Ucet [] ucty2;
ucty2 = ucty;
```

Proměnná `ucty2` obsahuje odkaz na stejné pole jako `ucty`.

Kopírování pole

```
Ucet [] ucty2 = new Ucet[5];
System.arraycopy(ucty, 0, ucty2, 0,
                 ucty.length);
```

- Proměnná `ucty2` obsahuje kopii původního pole.
- Také `arraycopy` však do cílového pole zduplikuje jen odkazy na objekty, nevytvoří kopie objektů!

Výpis argumentů programu

```
public class Pole {
    public static void main(String[] args) {
        for(int i=0; i<args.length; i++)
            System.out.println(args[i]);
    }
}
```