10. Operátory a výrazy, porovnávání objektů

- Operátory v Javě: aritmetické, logické, relační, bitové
- Porovnávání primitivních hodnot a objektů, metody equals a hashCode
- Ternární operátor podmíněného výrazu
- Typové konverze
- Operátor zřetězení

Aritmetické

`+`, `-`, `*`, `/` a `%` (zbytek po celočíselném dělení)

Logické

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):

- **`!`**

Relační (porovnávací)

Tyto lze použít na porovnávání primitivních hodnot:

- **`<`, `<=`, `>=`, `>`**

Test na rovnost/nerovnost lze použít na porovnávání primitivních hodnot i objektů:

- **`==`, `!=`**

Upozornění:

- pozor na porovnávání objektů: == vrací true jen při rovnosti odkazů, tj. jsou-li objekty identické. Rovnost obsahu (tedy "rovnocennost") objektů se zjišťuje voláním metody o1.equals(Object o2)
- 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

Porovnávání objektů

Použití ==

- Porovnáme-li dva objekty (tzn. odkazy na objekty) prostřednictvím

operátoru == dostaneme rovnost jen v případě, jedná-li se o dva odkazy na tentýž objekt - tj. dva ***totožné*** objekty.
- Jedná-li se o dva obsahově stejné objekty existující samostatně, pak == vrátí false.

Chceme-li (intuitivně) chápat rovnost objektů podle obsahu, tj.

- dva objekty jsou ***rovné*** (***rovnocenné***, nikoli ***totožné***), mají-li stejný obsah, pak
- musíme pro danou třídu překrýt metodu `equals`, která musí vrátit true, právě když se ***obsah*** výchozího a srovnávaného objektu rovná.
- Fungování `equals` lze srovnat s porovnáváním dvou databázových záznamů podle primárního klíče.
- Nepřekryjeme-li `equals`, funguje původní `equals` přísným způsobem, tj. ***rovné si budou jen totožné objekty***.

Porovnávání objektů - příklad

Příklad: objekt třídy Clovek nese informace o člověku. Dva objekty položíme stejné (rovnocenné), nesou-li stejná příjmení:

Obrázek 9.1. Dva lidi jsou stejní, mají-li stejná příjmení

```
public class Clovek implements Comparable {
    String jmeno, prijmeni;
    public Clovek (String j, String p) {
        jmeno = j;
        prijmeni = p;
    }
    public boolean equals(Object o) {
        if (o instanceof Clovek) {
            Clovek c = (Clovek)o;
            return prijmeni.equals(c.prijmeni);
        } else
            throw new IllegalArgumentException(
            "Nelze porovnat objekt typu Clovek s objektem jineho
typu");
    }
}
```

Méně agresivní verze by nemusela při porovnávání s jiným objektem než `Clovek` vyhodit výjimku, pouze vrátit false.

Metoda hashCode

Jakmile u třídy překryjeme metodu equals, měli bychom současně překrýt objektů i metodu `hashCode`:

- `hashCode` vrací celé číslo (int) „co nejlépe" charakterizující obsah objektu, tj.
- pro dva stejné (`equals`) objekty ***musí vždy vrátit stejnou hodnotu***.
- Pro dva obsahově různé objekty by `hashCode` naopak měl vracet různé hodnoty (ale není to stoprocentně nezbytné a ani nemůže být vždy splněno). Metoda `hashCode` totiž nemůže vždy být prostá.

Metoda hashCode - příklad

V těle `hashCode` s oblibou „přehráváme“ (delegujeme) řešení na volání `hashCode` jednotlivých složek objektu - a to těch, které figurují v equals:

Obrázek 9.2. Třída Clovek s metodami equals a hashCode

```
public class Clovek implements Comparable {
    String jmeno, prijmeni;
    public Clovek (String j, String p) {
        jmeno = j;
        prijmeni = p;
    }
    public boolean equals(Object o) {
        if (o instanceof Clovek) {
            Clovek c = (Clovek)o;
            return prijmeni.equals(c.prijmeni);
        } else
            throw new IllegalArgumentException(
            "Nelze porovnat objekt typu Clovek s objektem jineho
typu");
    }
    public int hashCode() {
         return prijmeni.hashCode();
    }
}
```

Bitové

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

Dále viz např. Bitové operátory

Operátor podmíněného výrazu ? :

Jediný ***ternární operátor***

Platí-li první operand (má hodnotu `true`) ->

- 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.

Operátory typové konverze (přetypování)

- Podobně jako v C/C++
- Píše se ***(typ)hodnota***, např. (`Clovek`)o, kde o byla proměnná deklarovaná jako `Object`.

- Pro objektové typy se ve skutečnosti nejedná o žádnou konverzi spojenou se změnou obsahu objektu, nýbrž pouze o potvrzení, že běhový typ objektu je požadovaného typu - např. (viz výše) že `o` je typu `Clovek`.
- Naproti tomu u primitivních typů se jedná o úpravu hodnoty - např. int přetypujeme na short a „ořeže" se tím rozsah.

Operátor zřetězení +

Výsledkem je vždy řetězec, ale argumenty mohou být i jiných typů, např.

sekvence `int i = 1; System.out.println("promenna i="+i);` je v pořádku

s řetězcovou konstantou se spojí řetězcová podoba dalších argumentů (např. čísla).

Pokud je argumentem zřetězení odkaz na objekt `o` ->

- je-li `o == null` -> použije se řetězec `null`
- je-li `o != null` -> použije se hodnota vrácená metodou `o.toString()` (tu lze překrýt a dosáhnout tak očekávaného řetězcového výstupu)

Priority operátorů a vytváření výrazů

nejvyšší prioritu má násobení, dělení, nejnižší přiřazení.