Porovnávání v Javě

Tomáš Pitner, upravil Marek Šabo

Porovnávání a pořadí (uspořádání)

- Obecně rozlišujeme, zda chceme zjišťovat *shodnost* (*rovnost*, *ekvivalenci*) mezi dvěma primitivními hodnotami nebo objekty, či nás zajímá,
- v jakém *pořadí* (*uspořádání*) hodnoty či objekty jsou, tzn. která/ý je dříve-později, evt. menší-větší.
- Zajímco u *primitivních hodnot* je jak rovnost tak pořadí určeno Javou *napevno* a nelze je programově změnit,
- u *objektů* lze u některých porovnání/uspořádání *chování programově určovat*.

Rovnost primitivních hodnot

- Rovnost/nerovnost primitivních hodnot zjišťujeme pomocí operátorů `==` (rovná se), `!=` (nerovná se)
- U integrálních typů funguje bez potíží
- U čísel floating-point (`double`, `float`) je většinou třeba porovnávat s určitou tolerancí, tzn. brát za rovná čísla odlišná o méně než jisté `delta`

Uspořádání primitivních hodnot

- Uspořádání primitivních hodnot, tzn. určení, která hodnota je menší/větší zjišťujeme pomocí operátorů `<`, `⇐`, `>=`, `>`
- Uspořádání není definováno na typu `boolean`, tzn. není určeno, zda `false < true`
- U *primitivních hodnot* nelze koncept uspořádání ani rovnosti programově měnit.

Jak chápat rovnost objektů

- Identita objektů: `==` vrací `true` právě při rovnosti *odkazů*, neboli když oba odkazy ukazují na přesně *tentýž objekt*, tj. jsou-li objekty identické. Odkazy obsahují tutéž adresu objektu.
- Rovnost obsahu: tedy jakási *rovnocennost*, *rovnost obsahu*, *ekvivalence* objektů se zjišťuje voláním metody `o1.equals(Object o2)`

Porovnávání objektů pomocí `==`

Použití `==` na objekty:

- 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 byť *obsahově stejné objekty*, ale existující samostatně, pak `==` vrátí `false`.

Porovnávání objektů dle obsahu

- 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* .

Příklad porovnávání objektů

- Objekt třídy `Person` nese informace o člověku.
- Zde dva objekty položíme stejné (rovnocenné), nesou-li stejná příjmení.

```
public class Person {
   private String firstname;
   private String surname;
   public Person (String j, String p) {
      firstname = j; surname = p;
   }
   public boolean equals(Object o) {
      if (o instanceof Person) {
         Person c = (Person)o;
         // dva lidé se rovnají, mají-li stejná příjmení
         return surname.equals(c.surname);
      } else {
         // porovnáváme-li osobu s ne-osobou...
         return false;
      }
   }
}
```

Metoda `hashCode`

S překrytím metody `equals` vyvstává jeden dosud nezmíněný problém:

- Jakmile u třídy překryjeme metodu `equals`, měli bychom současně překrýt i metodu `hashCode`. Tato metoda vrací celé číslo (`int`) co nejlépe charakterizující obsah objektu tak, aby:
- pro dva stejné (dle `equals`) objekty musí vždy vrátit *stejnou hodnotu*.
- Pro dva obsahově různé objekty by `hashCode` naopak měla vracet *různé hodnoty* (což ale není stoprocentně nezbytné a ani nemůže být vždy splněno).

Pozn. Metoda `hashCode` totiž nemůže obecně vždy být *prostá* — všechny složitější třídy mají potenciálně více možných hodnot (různých objektů) než je všech hodnot typu `int`, které `hashCode` vrací.

Příklad `hashCode` (1)

- V těle `hashCode` námi definované třídy s oblibou delegujeme řešení na volání `hashCode` jednotlivých složek našeho objektu
- a to těch, které figurují v `equals`. Rozšíříme minulý příklad.

Příklad `hashCode` (2)

```
public class Person {
   private String firstname;
   private String surname;
   public Person (String firstname, String surname) {
      this.firstname = firstname; this.surname = surname;
   }
   public boolean equals(Object o) {
      // zde zůstává porovnání příjmení
   }
   // nyní ale doplňujeme překrytí hashCode
   public int hashCode() {
      // které využívá volání hashCode na příjmení,
      // protože jedině příjmení využíváme v equals
      return surname.hashCode();
   }
}
```