Porovnávání v Javě

```
<link rel="stylesheet"  href="http://cdnjs.cloudflare.com/ajax/libs/font-
awesome/3.1.0/css/font-awesome.min.css">
```

Porovnávání a pořadí (uspořádání)

- Obecně rozlišujeme, zda chceme zjišťovat *shodnost* (*rovnost*, *ekvivalenci*):
  - mezi dvěma *primitivními hodnotami*
  - mezi dvěma *objekty*
- U *primitivních hodnot* jsou rovnost i pořadí určeny **napevno** a nelze je změnit.
- U *objektů* lze porovnání i uspořádání programově určovat.

Rovnost primitivních hodnot

- Rovnost primitivních hodnot zjišťujeme pomocí operátorů:
  - `==` (rovná se)
  - `!=` (nerovná se)
- U integrálních typů funguje bez potíží
- U čísel floating-point (`double`, `float`) je třeba porovnávat s určitou tolerancí

```
1 == 1 // true
1 == 2 // false
1 != 2 // true

1.000001 == 1.000001 // true
1.000001 == 1.000002 // false
Math.abs(1.000001 - 1.000002) < 0.001 // true
```

Uspořádání primitivních hodnot

- Uspořádání primitivních hodnot funguje pomocí operátorů `<`, `⇐`, `>=`, `>`
- U *primitivních hodnot* nelze koncept uspořádání ani rovnosti programově měnit.

Uspořádání není definováno na typu `boolean`, tj. neplatí `false < true`!

```
1.000001 <= 1.000002  // true
```

Jak chápat rovnost objektů

Identita objektů, `==`

vrací `true` při rovnosti **odkazů**, tj. když oba odkazy **ukazují na tentýž objekt**

Rovnost obsahu, metoda `equals`

vrací `true` pri logické ekvivalenci objektů (musí být explicitně nadefinované)

```
Person pepa1 = new Person("Pepa");
Person pepa2 = new Person("Pepa");
Person pepa3 = pepa1;

pepa1 == pepa2; // false
pepa1 == pepa3; // true
```

Porovnávání objektů pomocí ==

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

Objekty jsou identické = odkazy obsahují stejnou adresu objektu.

Porovnávání objektů dle obsahu

Rovnost obsahu, metoda `equals`:

- Dva objekty jsou *rovné* (*rovnocenné*), mají-li stejný obsah.
- Na zjištění rovnosti se použije metoda `equals`, kterou je potřeba překrýt.
- Pro nadefinování rovnosti bude hlavička metody **vždy** vypadat následovně:

```
@Override
public boolean equals(Object o)
```

- Parametrem je objekt typu `Object`.
- Jestli parametr není objekt typu `<class-name>`, obvykle je potřeba vrátit `false`.
- Pak se porovnají jednotlivé vlastnosti objektů a jestli jsou stejné, metoda vráti `true`.

Porovnávání objektů — komplexní číslo

Dvě komplexní čísla jsou stejná, když mají stejnou reálnou i imaginární část.

```
public class ComplexNumber {
  private int real, imag;

  public ComplexNumber(int r, int i) {
    real = r; imag = i;
  }
  @Override
  public boolean equals(Object o) {
    if (this.getClass() != o.getClass()) return false;

    ComplexNumber that = (ComplexNumber) o;
    return this.real == that.real
        && this.imag == that.imag;
  }
}
```

Porovnávání objektů — osoba I

Popis kódu na následujícím slajdu:

- Dvě osoby budou stejné, když mají stejné jméno a rok narození.
- Rovnost nemusí obsahovat porovnání všech atributů (porovnání `age` je zbytečné, když máme `yearBorn`).
- `String` je objekt, proto pro porovnání musíme použít metodu `equals`.
- Klíčové slovo `instanceof` říká "mohu pretypovat na daný typ".

Metoda `equals` musí být reflexivní, symetrická i tranzitivní (javadoc).

Porovnávání objektů — osoba II

```
public class Person {
  private String name;
  private int yearBorn, age;

  public Person(String n, int yB) {
    name = n; yearBorn = yB; age = currentYear - yB;
  }
  @Override
  public boolean equals(Object o) {
    if (!(o instanceof Person)) return false;

    Person that = (Person) o;
    return this.name.equals(that.name)
        && this.yearBorn == that.yearBorn;
  }
}
```

Porovnávání objektů — použití

```
ComplexNumber cn1 = new ComplexNumber(1, 7);
ComplexNumber cn2 = new ComplexNumber(1, 7);
ComplexNumber cn3 = new ComplexNumber(1, 42);
cn1.equals(cn2); // true
cn1.equals(cn3); // false

Person karel1 = new Person("Karel", 1993);
Person karel2 = new Person("Karel", 1993);

karel1.equals(karel2); // true

karel1.equals(cn1); // false
cn2.equals(karel2); // false
```

Chybějící `equals`

Co když zavolám metodu `equals` aniž bych ji přepsal?

Použije se původní metoda `equals` ve třídě `Object`:

```
public boolean equals(Object obj) {
  return (this == obj);
}
```

Původní `equals` funguje přísným způsobem — rovné jsou jen identické objekty.

Jak porovnat typ třídy

Je `this.getClass() == o.getClass()` stejné jako `o instanceof Person`?

- Ne! Jestli třída `Manager` dědí od `Person`, pak:

```
manager.getClass() == person.getClass() // false
manager instanceof Person // true
```

- Co tedy používat?
  - `instanceof` porušuje symetrii `x.equals(y) == y.equals(x)`
  - `getClass` porušuje tzv. *Liskov substitution principle*
- Záleží tedy na konkrétní situaci.

Metoda `hashCode`

- Při překrytí metody `equals` nastává dosud nezmíněný problém.
- Jakmile překryjeme metodu `equals`, měli bychom současně překrýt i metodu `hashCode`.
- Metoda `hashCode` je také ve třídě `Object`, tudíž ji obsahuje každá třída.

```
@Override
public int hashCode()
```

- Metoda vrací celé číslo pro daný objekt tak, aby:
  - pro dva stejné (`equals`) objekty musí **vždy** vrátit *stejnou hodnotu*
  - jinak by metoda měla vracet _různé hodnoty (není to nezbytné a ani nemůže být vždy splněno)
  - složité třídy mají více různých objektů než je všech hodnot typu `int`

Příklad `hashCode` I

```
public class ComplexNumber {
  private int real;
  private int imag;

  ...

  @Override
  public boolean equals(Object o) { ... }

  @Override
  public int hashCode() {
    return 31*real + imag;
  }
}
```

Příklad hashCode II

```
public class Person {
  private String name;
  private int yearBorn, age;
  ...
  @Override
  public boolean equals(Object o) { ... }

  @Override
  public int hashCode() {
    int hash = name.hashCode();
    hash += 31*hash + yearBorn;
    return hash;
  }
}
```

Obecný hashCode

Nejlépe je vytvářet metodu následujícím způsobem (31 je prvočíslo):

```
  @Override
  public int hashCode() {
    int hash = attribute1;
    hash += 31 * hash + attribute2;
    hash += 31 * hash + attribute3;
    hash += 31 * hash + attribute4;
    return hash;
  }
```

A nebo ji generovat (pokud víte, co to dělá :-))

Proč `hashCode`

- Metoda se používá v hašovacích tabulkách, využívá ji například množina `HashSet`.
- Při zjištění, jestli se prvek X nachází v množině, metoda vypočítá její haš (*hash*).
- Pak vezme všechny prvky se stejným hašem a zavolá `equals` (haš mohl být stejný náhodou).
- Jestli má každý objekt **unikátní** haš, pak je tato operace **konstantní**.
- Jestli má každý objekt **stejný** haš, pak je operace `contains` **lineární**!

Jestli se `hashCode` napíše špatně (nevrací pro stejné objekty stejný haš) nebo zapomene — množina nad danou třídou přestane fungovat!

Uspořádání objektů

- Budeme probírat později
- V Javě neexistuje přetěžování operátorů `<, ⇐, >, >=`
- Třída musí implementovat rozhraní `Comparable` a její metodu `compareTo`

Repl.it demo k porovnávání primitivních hodnot a objektů

- https://repl.it/@tpitner/PB162-Java-Lecture-06-vs-equals