Datové typy: primitivní, objektové, pole. Výrazy.

Obsah

Úvod k datovým typům v Javě

Cíl	Naučit se pracovat s primitivními a objektovými datovými typy v Javě, vymezit to vůči obecně známým principům (např. z Pascalu)
Předpoklady	Znát základní datové typy (číselné, logické, znakové) - např. z Pascalu

- Primitivní vs. objektové typy
- Kategorie primitivních typů: integrální, boolean, čísla s pohyblivou řádovou čárkou
- Pole: deklarace, vytvoření, naplnění, přístup k prvkům, rozsah indexů

Primitivní vs. objektové datové typy - opakování

Java striktně rozlišuje mezi hodnotami

- **primitivních datových typů** (čísla, logické hodnoty, znaky) a
- **objektových typů** (řetězce a všechny uživatelem definované [tj. vlastní] typy-třídy)

Základní rozdíl je v práci s proměnnými:

- proměnné primitivních typů *přímo obsahují danou hodnotu*, zatímco
- proměnné objektových typů obsahují pouze *odkaz na příslušný objekt*

Důsledek -> **dvě objektové proměnné** mohou nést odkaz na **tentýž objekt**

Přiřazení proměnné primitivního typu - opakování

- Příklad:

```
double a = 1.23456;
double b = a;
a += 2;
```

Přiřazení objektové proměnné - opakování

- Příklad, deklarujeme třídu `Counter`

[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Counter]takto:

```
public class Counter {
   private double value;
   public Counter(double v) {
      value = v;
   }
   public void add(double v) {
      value += v;
   }
   public void show() {
      System.out.println(value);
   }
}
```

- nyní ji použijeme:

```
Counter c1 = new Counter(1.23456);
Counter c2 = c1;
c1.add(2);
c1.show();
c2.show();
```

dostaneme:

```
3.23456
3.23456
```

Primitivní datové typy

Primitivní datové typy

Proměnné těchto typů nesou **elementární**, z hlediska Javy **atomické**, **dále nestrukturované** hodnoty.

Deklarace takové proměnné (kdekoli) způsobí:

1. rezervování příslušného paměťového prostoru (např. pro hodnotu `int`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=int] čtyři bajty)
2. zpřístupnění (pojmenování) tohoto prostoru identifikátorem proměnné

V Javě existují tyto skupiny primitivních typů:

1. **integrální typy** (obdoba ordinálních typů v Pascalu) - zahrnují typy *celočíselné* (`byte` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=byte], `short` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=short], `int` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=int] a `long` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=long]) a typ `char` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=char];
2. typy **čísel s pohyblivou řádovou čárkou** (`float` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=float] a `double` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=double])
3. typ **logických hodnot** (`boolean` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=boolean]).

Integrální typy - celočíselné

V Javě jsou celá čísla vždy interpretována jako znaménková

"Základním" celočíselným typem je 32bitový `int` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=int] s rozsahem `-2 147 483 648` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=-2 147 483 648] až `2147483647` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=2147483647]

větší rozsah (64 bitů) má `long` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=long], cca `+/- 9*10^18` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=+/- 9*10^18]

menší rozsah mají

- `short` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=short] (16 bitů), tj. -32768..32767
- `byte` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=byte] (8 bitů), tj. -128..127

Pro celočíselné typy existují (stejně jako pro floating-point typy) konstanty - *minimální a maximální hodnoty* příslušného typu. Tyto konstanty mají název vždy `Typ.MIN_VALUE` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= Typ.MIN_VALUE], analogicky MAX... Viz např. Minimální a maximální hodnoty [http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/hodnoty/MinMaxHodnoty.java]

Integrální typy - "char"

`char` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=char] představuje jeden 16bitový znak v kódování UNICODE

Konstanty typu `char` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=char] zapisujeme

- v apostrofech - 'a', 'Ř'
- pomocí escape-sekvencí - \n (konec řádku) \t (tabulátor)
- hexadecimálně - \u0040 (totéž, co 'a')
- oktalově - \127

Typ `char` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=char] - kódování

Java vnitřně kóduje znaky a řetězce v UNICODE, pro vstup a výstup je třeba použít některou za serializací (převodu) UNICODE na sekvence bajtů:

- např. vícebajtová kódování UNICODE: **UTF-8** a UTF-16
- osmibitová kódování ISO-8859-x, Windows-125x a pod.

Problém může nastat při interpretaci kódování znaků národních abeced uvedených přímo ve zdrojovém textu programu.

Ve zdroj. textu správně napsaného javového vícejazyčného programu by žádné národní znaky VŮBEC neměly vyskytovat.

Je vhodné umístit je do speciálních souborů tzv. *zdrojů* (v Javě objekty třídy `java.util.ResourceBundle` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=java.util.ResourceBundle]).

Čísla s pohyblivou řádovou čárkou

Kódována podle ANSI/IEEE 754-1985

- `float`WIKIPEDIA
  [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=float] - 32 bitů
- `double`WIKIPEDIA
  [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=double] - 64 bitů

Možné zápisy literálů typu `float`WIKIPEDIA
[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=float] (klasický i semilogaritmický tvar) - povšimněte si "f" za číslem - je u float nutné!:

```
float f = -.777f, g = 0.123f, h = -4e6f, 1.2E-15f;
```WIKIPEDIA
[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=float f = -.777f, g = 0.123f, h = -4e6f, 1.2E-15f;]

`double`WIKIPEDIA
[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=double]: tentýž zápis, ovšem bez "f" za konstantou!, s větší povolenou přesností a rozsahem

# Vestavěné konstanty s pohyblivou řádovou čárkou

Kladné a záporné nekonečno:

- `Float.POSITIVE_INFINITY` WIKIPEDIA
  [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Float.POSITIVE_INFINITY ], totéž s `NEGATIVE`WIKIPEDIA
  [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=NEGATIVE]...
- totéž pro `Double`WIKIPEDIA
  [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Double]
- obdobně existují pro oba typy konstanty uvádějící rozlišení daného typu - `MIN_VALUE`
  WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=MIN_VALUE ], podobně s `MAX`WIKIPEDIA
  [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=MAX]...

Konstanta `NaN` WIKIPEDIA
[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=NaN ] - *Not A Number*

Viz také Minimální a maximální hodnoty [http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/hodnoty/MinMaxHodnoty.java]

# Typ logických hodnot - *boolean*

Přípustné hodnoty jsou `false` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=false] a `true` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=true].

Na rozdíl od Pascalu na nich *není* definováno uspořádání, nelze je porovnávat pomocí <, >, <=, >=.

# Typ `void` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=void]

Význam podobný jako v C/C++.

Není v pravém slova smyslu datovým typem, nemá žádné hodnoty.

Označuje "prázdný" typ pro sdělení, že určitá metoda *nevrací žádný výsledek*.

# Pole

## Pole v Javě

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 - tzn. kontejnery, viz dále

Syntaxe deklarace

`typhodnoty [] jménopole` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=typhodnoty [] jménopole]

### Poznámka

na rozdíl od C/C++ nikdy neuvádíme při deklaraci počet prvků pole - ten je podstatný až při **vytvoření objektu pole**

Syntaxe přístupu k prvkům*jménopole[indexprvku]*Používáme

- jak pro **přiřazení** prvku do pole: `jménopole[indexprvku] = hodnota;`WIKIPEDIA

  [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= jménopole[indexprvku] = hodnota;]

- tak pro **čtení** hodnoty z pole `proměnná = jménopole[indexprvku];`WIKIPEDIA

  [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= proměnná = jménopole[indexprvku];]

Syntaxe vytvoření *objektu* pole: jako u jiného objektu - voláním konstruktoru:

*jménopole = new typhodnoty[ početprvků ];* nebo vzniklé pole rovnou naplníme hodnotami/odkazy

*jménopole* = new *typhodnoty*[] **{prvek1, prvek2, ...}**;

# Pole (2)

Pole je objekt, je třeba ho před použitím nejen **deklarovat**, ale i **vytvořit**:

```
Person[] lidi;
lidi = new Person[5];
```

`Clovek`WIKIPEDIA

[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Clovek]

Nyní můžeme pole naplnit:

```
lidi[0] = new Person("Václav Klaus");
lidi[1] = new Person("Libuše Benešová");


lidi[0].writeInfo();
lidi[1].writeInfo();
```

- Nyní jsou v poli `lidi`WIKIPEDIA

  [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=lidi] naplněny první dva prvky odkazy na objekty.

- Zbylé prvky zůstaly naplněny prázdnými odkazy `null`WIKIPEDIA

  [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=null].

Pole - co když deklarujeme, ale nevytvoříme?

Co kdybychom pole pouze deklarovali a nevytvořili:

```
Person[] lidi;
lidi[0] = new Person("Václav Klaus");
```

Toto by skončilo s běhovou chybou "NullPointerException", pole neexistuje, nelze do něj tudíž vkládat prvky!

Pokud tuto chybu uděláme v rámci metody:

```
public class Pokus {
   public static void main(String args[]) {
      String[] pole;
      pole[0] = "Neco";
   }
}
```

překladač nás varuje:

```
Pokus.java:4: variable pole might not have been
    initialized pole[0] = "Neco"; ^ 1 error
```

Pokud ovšem

`pole` WIKIPEDIA The Free Encyclopedia [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=pole]

bude proměnnou objektu/třídy:

```
public class Pokus {
   static String[] pole;
   public static void main(String args[]) {
      pole[0] = "Neco";
   }
}
```

Překladač chybu neodhalí a po spuštění se objeví:

```
Exception in thread "main"
    java.lang.NullPointerException at Pokus.main(Pokus.java:4)
```

Pole - co když deklarujeme, vytvoříme, ale nenaplníme?

Co kdybychom pole deklarovali, vytvořili, ale nenaplnili příslušnými prvky:

```
Person[] lidi;
```

```
lidi = new Person[5];
lidi[0].writeInfo();
```

Toto by skončilo také s běhovou chybou *NullPointerException*:

- pole existuje, má pět prvků, ale první z nich je prázdný, nelze tudíž volat jeho metody (resp. vůbec používat jeho vlastnosti)!

Kopírování polí

V Javě obecně přiřazení proměnné objektového typu vede pouze k **duplikaci odkazu**, **nikoli** celého odkazovaného **objektu**.

Nejinak je tomu u polí, tj.:

```
Person[] lidi2;
lidi2 = lidi1;
```

V proměnné*lidi2*je nyní odkaz na stejné pole jako je v*lidi1*.

Zatímco, provedeme-li vytvoření nového pole + *arraycopy*, pak *lidi2* obsahuje duplikát/klon/kopii původního pole.

```
Person[] lidi2 = new Person[5];
System.arraycopy(lidi, 0, lidi2, 0, lidi.length);
```

viz též Dokumentace API třídy "System" [http://java.sun.com/j2se/1.4/docs/api/java/lang/System.html]

Poznámka

Samozřejmě bychom mohli kopírovat prvky ručně, např. pomocí **for**WIKIPEDIA The Free Encyclopedia [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=for] cyklu, ale volání *System.arraycopy* je zaručeně nejrychlejší a přitom stále platformově nezávislou metodou, jak kopírovat pole.

Také *arraycopy* však do cílového pole zduplikuje jen **odkazy na objekty**, nevytvoří kopie objektů!

Operátory a výrazy

Cíl	Zvládnout použití operátorů v Javě a naučit se sestavovat výrazy různých typů
Předpoklady	Znát obecné principy syntaxe a vyhodnocování výrazů v pg. jazycích (např. Pascalu)

- Operátory v Javě: aritmetické, logické, relační, bitové
- Ternární operátor podmíněného výrazu
- Typové konverze
- Operátor zřetězení

Aritmetické

+ [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=+], - [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=-], * [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=*], / [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=/] a % [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=%] (zbytek po celočíselném dělení)

Pozn: operátor dělení / je polymorfní, funguje pro celočíselné argumenty jako *celočíselný*, pro floating-point (float, double) jako "obyčejný".

Logické

Pracují nad logickými (booleovskými) hodnotami (samozřejmě vč. výsledků porovnávání <, >, ==, atd.).

logické součiny (AND):

- `&` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= &] (*ne-podmíněný* - vždy se vyhodnotí oba operandy),
- `&&` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= &&] (*podmíněný* - líné vyhodnocování - druhý operand se vyhodnotí, jen nelze-li o výsledku rozhodnout z hodnoty prvního)

logické součty (OR):

- `|` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= |] (*ne-podmíněný* - vždy se vyhodnotí oba operandy),
- `||` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= ||] (*podmíněný* - líné vyhodnocování - druhý operand se vyhodnotí, jen nelze-li o výsledku rozhodnout z hodnoty prvního)

negace (NOT):

- ! WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= !]

Relační (porovnávací)

Tyto lze použít na porovnávání primitivních hodnot:

- < WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= <], <= WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= <=], >= WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= >=], > WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= >]

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

- == WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= ==], != WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= !=]

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

Bitové

Bitové:

- součin `&` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= &]
- součet `|` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= |]
- exkluzivní součet (XOR) ^ WIKIPEDIA

[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= ^] (znak "stříška")

- negace (bitwise-NOT) ~ WIKIPEDIA

  [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= ~] (znak "tilda") - obrátí bity argumentu a výsledek vrátí

Posuny:

- vlevo <<WIKIPEDIA

  [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=<<] o stanovený počet bitů

- vpravo >>WIKIPEDIA

  [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=>>] o stanovený počet bitů s respektováním znaménka

- vpravo >>>WIKIPEDIA

  [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=>>>] o stanovený počet bitů bez respektování znaménka

Dále viz např. Bitové operátory [http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/operatory/Bitove.java]

Operátor podmíněného výrazu ? : WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=? :]

Jediný *ternární operátor*, navíc polymorfní, pracuje nad různými typy 2. a 3. argumentu.

Platí-li první operand (má hodnotu `true`WIKIPEDIA

[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=true]) ->

- výsledkem je hodnota druhého operandu
- jinak je výsledkem hodnota třetího operandu

Typ prvního operandu musí být `boolean`WIKIPEDIA

[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=boolean], typy druhého a třetího musí být přiřaditelné do výsledku.

Operátory typové konverze (přetypování)

- Podobně jako v C/C++
- Píše se *(typ)hodnota*, např. (Person)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* (tj. typovou kontrolu), že běhový typ objektu je požadovaného typu - např. (viz výše) že o je typu Person.
- Naproti tomu u primitivních typů se jedná o úpravu hodnoty - např. int přetypujeme na short a „ořeže“ se tím rozsah.

Operátor zřetězení +WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=+]

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

sekvence `int i = 1; System.out.println("promenna i="+i);`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=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 WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=o]->

- je-li `o == null`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=o == null] -> použije se řetězec `"null"`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search="null"]
- je-li `o != null`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=o != null] -> použije se hodnota vrácená metodou `o.toString()`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=o.toString()] (tu lze překrýt a dosáhnout tak očekávaného řetězcového výstupu)

Priority operátorů a vytváření výrazů

nejvyšší prioritu má násobení, dělení, nejnižší přiřazení

Porovnávání objektů

- Porovnávání primitivních hodnot a objektů je zásadně odlišné.
- U objektů lze kromě == a != použít metodu equals.

Relační (porovnávací)

Na objekty nelze použít:

- < WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= <], <= WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= <=], >= WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= >=], > WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= >]

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

- == WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= ==], != WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= !=]

Znovu podstatné 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)

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 1. Dva lidi jsou stejní, mají-li stejná příjmení

```
public class Person implements Comparable {
   private String firstname, 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 (v našem případě) rovnají, mají-li stejná příjmení
         return surname.equals(c.surname);
      } else {
         // porovnáváme-li osobu s objektem jiného typu, nikdy se nerovnají
         return false;
      }
   }
}
```

Méně agresivní verze by nemusela při porovnávání s jiným objektem než Person 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 2. Třída Clovek s metodami equals a hashCode

```
public class Person implements Comparable {
   private String firstname, 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 (v našem případě) rovnají, mají-li stejná příjmení
         return surname.equals(c.surname);
      } else {
         // porovnáváme-li osobu s objektem jiného typu, nikdy se nerovnají
         return false;
      }
   }
   public int hashCode() {
      return surname.hashCode();
   }
}
```