Datové typy: primitivní, objektové, pole. Výrazy.

Obsah

Úvod k datovým typ#m v Jav#

Cíl	Nau#it se pracovat s primitivními a objektovými datovými typy v Jav#, vymezit to v##i obecn# známým princip#m (nap#. z Pascalu)
P#edpoklady	Znát základní datové typy (#íselné, logické, znakové) - nap#. z Pascalu

- Primitivní vs. objektové typy
- Kategorie primitivních typ#: integrální, boolean, #ísla s pohyblivou #ádovou #árkou
- Pole: deklarace, vytvo#ení, napln#ní, p#ístup k prvk#m, rozsah index#

Primitivní vs. objektové datové typy - opakování

Java striktn# rozlišuje mezi hodnotami

- **primitivních datových typ#** (#ísla, logické hodnoty, znaky) a
- **objektových typ#** (#et#zce a všechny uživatelem definované [tj. vlastní] typy-t#ídy)

Základní rozdíl je v práci s prom#nnými:

- prom#nné primitivních typ# *p#ímo obsahují danou hodnotu*, zatímco
- prom#nné objektových typ# obsahují pouze *odkaz na p#íslušný objekt*

D#sledek -> **dv# objektové prom#nné** mohou nést odkaz na **tentýž objekt**

P#i#azení prom#nné primitivního typu - opakování

- P#íklad:

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

P#i#azení objektové prom#nné - opakování

- P#íklad, deklarujeme t#ídu `Counter [http://www.google.com/search?q=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

Prom#nné t#chto typ# nesou **elementární**, z hlediska Javy **atomické**, **dále nestrukturované** hodnoty.

Deklarace takové prom#nné (kdekoli) zp#sobí:

1. rezervování p#íslušného pam##ového prostoru (nap#. pro hodnotu `int` [http://www.google.com/search?q=int] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=int] #ty#i bajty)
2. zp#ístupn#ní (pojmenování) tohoto prostoru identifikátorem prom#nné
3. Místo, kde je pam##ový prostor pro prom#nnou rezervován, závisí na tom, zda se jedná o porm#nnou lokální (tzn. bu# parametr metody nebo prom#nná v metod# deklarovaná), pak se vyhradí na zásobníku, nebo zda jde o prom#nnou objektu #i t#ídy -- pak má místo v rámci pam##ového prostoru objektu.

V Jav# existují tyto skupiny primitivních typ#:

1. **integrální typy** (obdoba ordinálních typ# v Pascalu) - zahrnují typy *celo#íselné* (`byte` [http://www.google.com/search?q=byte] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=byte], `short` [http://www.google.com/search?q=short] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=short], `int` [http://www.google.com/search?q=int] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=int] a `long` [http://www.google.com/search?q=long] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=long]) a typ `char` [http://www.google.com/search?q=char] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=char];
2. typy **#ísel s pohyblivou #ádovou #árkou** (`float` [http://www.google.com/search?q=float] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=float] a `double` [http://www.google.com/search?q=double] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=double])
3. typ **logických hodnot** (`boolean` [http://www.google.com/search?q=boolean]

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

Integrální typy - celo#íselné

V Jav# jsou celá #ísla vždy interpretována jako znaménková

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

v#tší rozsah (64 bit#) má `long` [http://www.google.com/search?q=long] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=long], cca `+/- 9*10^18` [http://www.google.com/search?q=+/- 9*10^18] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=+/- 9*10^18]

menší rozsah mají

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

Pro celo#íselné typy existují (stejn# jako pro floating-point typy) konstanty - *minimální a maximální hodnoty* p#íslušného typu. Tyto konstanty mají název vždy `Typ.MIN_VALUE` [http://www.google.com/search?q=Typ.MIN_VALUE] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Typ.MIN_VALUE], analogicky MAX... Viz nap#. Minimální a maximální hodnoty [http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/hodnoty/MinMaxHodnoty.java]

Integrální typy - "char"

`char` [http://www.google.com/search?q=char] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=char] p#edstavuje jeden 16bitový znak v kódování UNICODE

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

- v apostrofech - 'a', '#'
- pomocí escape-sekvencí - \n (konec #ádku) \t (tabulátor)
- hexadecimáln# - \u0040 (totéž, co 'a')
- oktalov# - \127

Typ `char` [http://www.google.com/search?q=char] http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:S[earch?search=char] - kódování

Java vnit#n# kóduje znaky a #et#zce v UNICODE, pro vstup a výstup je t#eba použít n#kterou za serializací (p#evodu) UNICODE na sekvence bajt#:

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

Problém m#že nastat p#i interpretaci kódování znak# národních abeced uvedených p#ímo ve zdrojovém textu programu.

Ve zdroj. textu správn# napsaného javového vícejazy#ného programu by žádné národní znaky V#BEC nem#ly vyskytovat.

Je vhodné umístit je do speciálních soubor# tzv. *zdroj#* (v Jav# objekty t#ídy `java.util.ResourceBundle` [http://www.google.com/search?q=java.util.ResourceBundle] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=java.util.ResourceBundle]).

#ísla s pohyblivou #ádovou #árkou

Kódována podle ANSI/IEEE 754-1985

- `float` [http://www.google.com/search?q=float] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=float] - 32 bit#
- `double` [http://www.google.com/search?q=double] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=double] - 64 bit#

Možné zápisy literál# typu `float` [http://www.google.com/search?q=float] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=float] (klasický i semilogaritmický tvar) - povšimn#te si "f" nebo "F" za #íslem - je u float nutné!:

`float f = -.777f, g = 0.123f, h = -4e6F, 1.2E-15f;` [http://www.google.com/search?q=float f = -.777f, g = 0.123f, h = -4e6F, 1.2E-15f;] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=float f = -.777f, g = 0.123f, h = -4e6F, 1.2E-15f;]

`double` [http://www.google.com/search?q=double] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=double]: tentýž zápis, ovšem bez "f" za konstantou a s v#tší povolenou p#esností a rozsahem

Vestav#né konstanty s pohyblivou #ádovou #árkou

- Kladné "nekone#no": `Float.POSITIVE_INFINITY` [http://www.google.com/search?q=Float.POSITIVE_INFINITY] http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Float.POSITIVE_INFINIT

Y], podobn# záporné: `Float.NEGATIVE` [http://www.google.com/search?q=Float.NEGATIVE] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Float.NEGATIVE]...

- totéž pro `Double` [http://www.google.com/search?q=Double] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Double]
- Obdobn# existují pro oba typy konstanty uvád#jící rozlišení (nejmenší uložitelnou absolutní hodnotu r#znou od 0) daného typu - `MIN_VALUE` [http://www.google.com/search?q=MIN_VALUE] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=MIN_VALUE], podobn# pro `MAX_VALUE` [http://www.google.com/search?q=MAX_VALUE] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=MAX_VALUE]...
- Konstanta `NaN` [http://www.google.com/search?q=NaN] [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*

P#ípustné hodnoty jsou `false` [http://www.google.com/search?q=false] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=false] a `true` [http://www.google.com/search?q=true] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=true].

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

Typ `void` [http://www.google.com/search?q=void] http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:S [earch?search=void]

Význam podobný jako v C/C++.

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

Ozna#uje "prázdný" typ pro sd#lení, že ur#itá metoda *nevrací žádný výsledek*.

Všechno, co jste cht#li v#d#t o primitivních datových typech...

...najdete na Sun Microsystems, Inc.: The Java Tutorial: Primitive Data Types [http://java.sun.com/docs/books/tutorial/java/nutsandbolts/datatypes.html].

Pole

Pole v Jav#

Pole v Jav# je speciálním **objektem**

M#žeme mít pole jak primitivních, tak objektových hodnot

- pole primitivních hodnot tyto **hodnoty obsahuje**
- pole objekt# obsahuje **odkazy na objekty**

Krom# pole v Jav# existují i jiné objekty na ukládání více hodnot - tzn. kontejnery, viz dále

Syntaxe deklarace

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

Poznámka

na rozdíl od C/C++ nikdy neuvádíme p#i deklaraci po#et prvk# pole - ten je podstatný až p#i **vytvo#ení objektu pole**

Syntaxe p#ístupu k prvk#m*jménopole[indexprvku]*Používáme

- jak pro **p#i#azení** prvku do pole: `jménopole[indexprvku] = hodnota;` `[http://www.google.com/search?q= jménopole[indexprvku] = hodnota;]` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= jménopole[indexprvku] = hodnota;]
- tak pro **#tení** hodnoty z pole `prom#nná = jménopole[indexprvku];` `[http://www.google.com/search?q= prom#nná = jménopole[indexprvku];]` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= prom#nná = jménopole[indexprvku];]

Syntaxe vytvo#ení *objektu* pole: jako u jiného objektu - voláním konstruktoru:

jménopole = new typhodnoty[po#etprvk#]; nebo vzniklé pole rovnou naplníme hodnotami/odkazy

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

Pole (2)

Pole je objekt, je t#eba ho p#ed použitím nejen **deklarovat**, ale i **vytvo#it**:

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

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

Nyní m#žeme 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` [http://www.google.com/search?q=lidi] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=lidi] napln#ny první dva prvky odkazy na objekty.
- Zbylé prvky z#staly napln#ny prázdnými odkazy `null` [http://www.google.com/search?q=null] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=null].

Pole - co když deklarujeme, ale nevytvo#íme?

Co kdybychom pole pouze deklarovali a nevytvo#ili:

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

Toto by skon#ilo s b#hovou chybou "NullPointerException", pole neexistuje, nelze do n#j tudíž vkládat prvky!

Pokud tuto chybu ud#láme v rámci metody:

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

p#eklada# nás varuje:

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

Pokud ovšem

`pole` [http://www.google.com/search?q=pole] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=pole]

bude prom#nnou objektu/t#ídy:

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

P#eklada# chybu neodhalí a po spušt#ní se objeví:

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

Pole - co když deklarujeme, vytvo#íme, ale nenaplníme?

Co kdybychom pole deklarovali, vytvo#ili, ale nenaplnili p#íslušnými prvky:

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

Toto by skon#ilo také s b#hovou chybou *NullPointerException*:

- pole existuje, má p#t prvk#, ale první z nich je prázdný, nelze tudíž volat jeho metody (resp. v#bec používat jeho vlastnosti)!

Kopírování polí

V Jav# obecn# p#i#azení prom#nné objektového typu vede pouze k **duplikaci odkazu**, **nikoli** celého odkazovaného **objektu**.

Nejinak je tomu u polí, tj.:

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

V prom#nné*lidi2*je nyní odkaz na stejné pole jako je v*lidi1*.

Zatímco, provedeme-li vytvo#ení nového pole + *arraycopy*, pak *lidi2* obsahuje duplikát/klon/kopii p#vodního pole.

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

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

Poznámka

Samoz#ejm# bychom mohli kopírovat prvky ru#n#, nap#. pomocí **for [http://www.google.com/search?q=for]** [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=for] cyklu, ale volání *System.arraycopy* je zaru#en# nejrychlejší a p#itom stále platformov# nezávislou metodou, jak kopírovat pole.

Také *arraycopy* však do cílového pole zduplikuje jen **odkazy na objekty**, nevytvo#í kopie objekt#!

Operátory a výrazy

Cíl	Zvládnout použití operátor# v Jav# a nau#it se sestavovat výrazy r#zných typ#
P#edpoklady	Znát obecné principy syntaxe a vyhodnocování výraz# v pg. jazycích (nap#. Pascalu)

- Operátory v Jav#: aritmetické, logické, rela#ní, bitové

- Ternární operátor podmín#ného výrazu
- Typové konverze
- Operátor z#et#zení

Aritmetické

`+` [http://www.google.com/search?q=+] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=+], `-` [http://www.google.com/search?q=-] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=-], `*` [http://www.google.com/search?q=*] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=*], `/` [http://www.google.com/search?q=/] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=/] a `%` [http://www.google.com/search?q=%] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=%] (zbytek po celo#íselném d#lení)

Pozn: operátor d#lení / je polymorfní, funguje pro celo#íselné argumenty jako *celo#íselný*, pro floating-point (float, double) jako "oby#ejný".

Logické

Pracují nad logickými (booleovskými) hodnotami (samoz#ejm# v#. výsledk# porovnávání <, >, ==, atd.).

logické sou#iny (AND):

- `&` [http://www.google.com/search?q=&] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=&] (*nepodmín#ný* - vždy se vyhodnotí oba operandy),
- `&&` [http://www.google.com/search?q=&&] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=&&] (*podmín#ný* - líné vyhodnocování (lazy evaluation) - druhý operand se vyhodnotí, jen nelze-li o výsledku rozhodnout z hodnoty prvního)

logické sou#ty (OR):

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

negace (NOT):

- `!` [http://www.google.com/search?q=!]

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

Rela#ní (porovnávací)

Tyto lze použít na porovnávání primitivních hodnot:

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

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

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

Upozorn#ní:

- Pozor na porovnávání objekt#: == vrací true jen p#i rovnosti odkaz#, tj. jsou-li objekty *identické*. Rovnost *obsahu* (tedy "rovnocennost") objekt# se zji#uje voláním metody o1.equals(Object o2)
- Pozor na srovnávání floating-point #ísel na rovnost: je t#eba po#ítat s chybami zaokrouhlení; místo porovnání na p#esnou rovnost rad#ji používejme jistou toleranci: abs(expected-actual) < delta

Bitové

Bitové:

- sou#in & [http://www.google.com/search?q= &] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= &]
- sou#et | [http://www.google.com/search?q= |] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= |]
- exkluzivní sou#et (XOR) ^ [http://www.google.com/search?q= ^] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= ^] (znak "st#íška")
- negace (bitwise-NOT) ~ [http://www.google.com/search?q= ~] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= ~] (znak "tilda") - obrátí bity argumentu a výsledek vrátí

Posuny:

- vlevo `<<` [http://www.google.com/search?q=<<] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=<<] o stanovený po#et bit#
- vpravo `>>` [http://www.google.com/search?q=>>] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=>>] o stanovený po#et bit# s respektováním znaménka
- vpravo `>>>` [http://www.google.com/search?q=>>>] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=>>>] o stanovený po#et bit# bez respektování znaménka

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

Operátor podmín#ného výrazu `? :` [http://www.google.com/search?q=? :] http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:S [earch?search=? :]

Jediný *ternární operátor*, navíc polymorfní, pracuje nad r#znými typy 2. a 3. argumentu.

Platí-li první operand (má hodnotu `true` [http://www.google.com/search?q=true] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=true]) ->

- výsledkem je hodnota druhého operandu
- jinak je výsledkem hodnota t#etího operandu

Typ prvního operandu musí být `boolean` [http://www.google.com/search?q=boolean] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=boolean], typy druhého a t#etího musí být p#i#aditelné do výsledku.

Operátory typové konverze (p#etypování)

- Podobn# jako v C/C++
- Píše se *(typ)hodnota*, nap#. (Person)o, kde o byla prom#nná deklarovaná jako Object.
- Pro objektové typy se ve skute#nosti *nejedná o žádnou konverzi* spojenou se zm#nou obsahu objektu, nýbrž pouze o *potvrzení* (tj. typovou kontrolu), že b#hový typ objektu je požadovaného typu - nap#. (viz výše) že o je typu Person.
- Naproti tomu u primitivních typ# se jedná o úpravu hodnoty - nap#. int p#etypujeme na short a „o#eže“ se tím rozsah.

Operátor z#et#zení + [http://www.google.com/search?q=+] http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:S

earch?search=+]

Výsledkem je vždy #et#zec, ale argumenty mohou být i jiných typ#, nap#.

sekvence `int i = 1; System.out.println("variable i = " + i);` `[http://www.google.com/search?q=int i = 1; System.out.println("variable i = " + i);]` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=int i = 1; System.out.println("variable i = " + i);] je v po#ádku

s #et#zcovou konstantou se spojí #et#zcová podoba dalších argument# (nap#. #ísla).

Pokud je argumentem z#et#zení odkaz na objekt `o [http://www.google.com/search?q=o ]` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=o]->

- je-li `o == null` `[http://www.google.com/search?q=o == null]` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=o == null] -> použije se #et#zec `"null"` `[http://www.google.com/search?q="null"]` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search="null"]
- je-li `o != null` `[http://www.google.com/search?q=o != null]` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=o != null] -> použije se hodnota vrácená metodou `o.toString()` `[http://www.google.com/search?q=o.toString()]` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=o.toString()] (tu lze p#ekrýt a dosáhnout tak o#ekávaného #et#zcového výstupu)

Priority operátor# a vytvá#ení výraz#

nejvyšší prioritu má násobení, d#lení, nejnižší p#i#azení

Porovnávání objekt#

- Porovnávání primitivních hodnot a objekt# je zásadn# odlišné.
- U objekt# lze krom# == a != použít metodu equals.

Rela#ní (porovnávací)

Na objekty nelze použít:

- `<` `[http://www.google.com/search?q=<]` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=<], `<=` `[http://www.google.com/search?q=<=]` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=<=], `>=` `[http://www.google.com/search?q=>=]` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=>=], `>` `[http://www.google.com/search?q=>]` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=>]

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

- `==` [http://www.google.com/search?q= `==`] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= `==`], `!=` [http://www.google.com/search?q= `!=`] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= !=]

Znovu podstatné upozorn#ní:

- pozor na porovnávání objekt#: == vrací true jen p#i rovnosti odkaz#, tj. jsou-li objekty identické. Rovnost obsahu (tedy "rovnocennost") objekt# se zjiš#uje voláním metody o1.equals(Object o2)

Porovnávání objekt#

Použití ==

- Porovnáme-li dva objekty (tzn. odkazy na objekty) prost#ednictvím operátoru == dostaneme rovnost jen v p#ípad#, jedná-li se o dva odkazy na tentýž objekt - tj. dva *totožné* objekty.
- Jedná-li se o dva obsahov# stejné objekty existující samostatn#, pak == vrátí false.

Chceme-li (intuitivn#) chápat rovnost objekt# podle obsahu, tj.

- dva objekty jsou *rovné* (*rovnocenné*, nikoli *totožné*), mají-li stejný obsah, pak
- musíme pro danou t#ídu p#ekrýt metodu equals, která musí vrátit true, práv# když se *obsah* výchozího a srovnávaného objektu rovná.
- Fungování equals lze srovnat s porovnáváním dvou databázových záznam# podle primárního klí#e.
- Nep#ekryjeme-li equals, funguje p#vodní equals p#ísným zp#sobem, tj. *rovné si budou jen totožné objekty*.

Porovnávání objekt# - p#íklad

P#íklad: objekt t#ídy Person nese informace o #lov#ku. Dva objekty položíme stejné (rovnocenné), nesou-li stejná p#íjmení:

Obrázek 1. Dva lidi jsou stejní, mají-li stejná p#íjmení

```
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 (v našem p#ípad#) rovnají, mají-li stejná p#íjmení
         return surname.equals(c.surname);
```

```
      } else {
         // porovnáváme-li osobu s objektem jiného typu, nikdy se nerovnají
         return false;
      }
   }
}
```

Metoda hashCode

Jakmile u t#ídy p#ekryjeme metodu equals, m#li bychom sou#asn# p#ekrýt objekt# i metodu hashCode:

- *hashCode* vrací celé #íslo (int) „co nejlépe“ charakterizující obsah objektu, tj.
- pro dva stejné (*equals*) objekty *musí vždy vrátit stejnou hodnotu*.
- Pro dva obsahov# r#zné objekty by *hashCode* naopak m#l vracet *r#zné* hodnoty (ale není to stoprocentn# nezbytné a ani nem#že být vždy spln#no).

  Metoda *hashCode* totiž nem#že vždy být prostá.

Metoda hashCode - p#íklad

V t#le *hashCode* s oblibou „p#ehráváme“ (delegujeme) #ešení na volání *hashCode* jednotlivých složek objektu - a to t#ch, které figurují v *equals*:

Obrázek 2. T#ída Person s metodami equals a hashCode

```
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 (v našem p#ípad#) rovnají, mají-li stejná p#íjmení
         return surname.equals(c.surname);
      } else {
         // porovnáváme-li osobu s objektem jiného typu, nikdy se nerovnají
         return false;
      }
   }
   public int hashCode() {
      return surname.hashCode();
   }
}
```