Objektové obálky nad primitivními hodnotami

Table of Contents

Primitivní typy a jejich obálky

- Java má primitivní typy — `int`, `char`, `double`, …
- Ke každému primitivnímu typu existuje varianta objektového typu — `Integer`, `Character`, `Double`, …
- Tyto objektové typy se nazývají *wrappers*.
- Jejich objekty jsou neměnné.

Není nutné používat `new`

- využije se tzv. *autoboxing*, např. `Integer five = 5;`
- Obdobně autounboxing, `int alsoFive = five;`

```
Integer objectInt = new Integer(42);
Integer objectInt2 = 42;
```

Konstanty objektových obálek

- Objektové obálky (např. `Double`) mají různé konstanty:
  - `MIN_VALUE` je minimální hodnota jakou může `double` obsahovat
  - `POSITIVE_INFINITY` reprezentuje konstantu kladné nekonečno
  - `NaN` je zkratkou Not-a-Number — dostaneme ji např. dělením nuly

- Protože konstanty jsou statické, jejich hodnoty získáme přes název třídy:

```
double d = Double.MIN_VALUE;
d = Double.NEGATIVE_INFINITY;
```

Metody objektových obálek

- např. pro `Double` existuje `static double parseDouble(String s)` — udělá ze `String` číslo, z `"1.0"`, vytvoří číslo `1.0`
- obdobně pro `Integer` a další číselné typy
- pro převody na číselné typy dále `int intValue()` převod `double` do typu `int`
- `boolean isNaN()` — test, jestli není hodnotou číslo

Více konstant a metod popisuje javadoc.

Víc hodnot

Test:

- Objektové typy (`Integer`) mají od primitivních (`int`) jednu hodnotu navíc — uhádněte jakou!
- je to `null`
- `Integer` je objektový typ, proměnná je odkaz na objekt

Rozdíl od primitivních typů

Proč tedy vůbec používat primitivní typy, když máme typy objektové?

`int i = 1`

- zabere v paměti právě jen 32 bitů
- používáme přímo danou paměť, jednička je uložena přímo v `i`

`Integer i = 1`

- je třeba alokace paměti pro objekt, zkonstruování objektu s obsahem `1`
- v proměnné `i` je pouze odkaz, je to (nepatrně) pomalejší

Výkon může být u velkého počtu objektů problém, např. vytvoření miliónu proměnných typu `Integer` namísto `int` *může* mít dopad na výkon a zcela jistě zabere dost paměti, asi zbytečně.

Doporučení

- Používejte *hlavně primitivní typy*
- Využívejte metody objektových typů, hlavně statické, kde není třeba mít objekt
- Řada objektových jazyků vůbec primitivní typy jako v Javě nemá, vše jsou objekty

Konverze mezi primitivními a objektovými typy

- Java podporuje automatické zabalení (boxing) a vybalení (unboxing) mezi primitivními typy a wrappery.
- Proto je následující kód je naprosto v pořádku:
- Nicméně použití *primitivního typu* je obvykle lepší nápad.

Příklad konverze

```
int primitiveInt = 42;
// jakoby new Integer(primitiveInt),
// tedy zabalení hodnoty do objektu
Integer objectInt = primitiveInt;
// vybalení hodnoty z objektu
primitiveInt = new Integer(43);
```

Zvláštnosti

- Zajímavost, anebo spíš podraz Javy:

```
Integer i1 = 100; // between -127 and 128
Integer i2 = 100;
boolean referencesAreEqual = (i1 == i2); // true

i1 = 300;
i2 = 300;
boolean referencesNotEqual = (i1 == i2); // false
```

- Poučení: objekty pomocí `==` obvykle neporovnáváme (budeme se učit o `equals`).

Proč tak podivné chování?

- Optimalizace využití paměti: *"valueOf() returns an Integer instance representing the specified int value. This method will always cache values in the range -128 to 127, inclusive, and may cache*

other values outside of this range."

- Důsledek použití návrhové vzoru *Flyweight (muší váha)*, kdy konstruktor (respektive autoboxing kód v Javě) někdy vrací již dříve vytvořenou instanci (de facto *Singleton*), jindy zas zcela novou instanci.