Proměnné, deklarace

Table of Contents

Proměnné . . . 1
Pojmenování proměnných . . . 1
Konvence v Javě na rozdíl od Pythonu . . . 1
Kategorie proměnných . . . 2
Realizace proměnných . . . 2
Lokální proměnné . . . 2
Jak to přesně je? . . . 2
Deklarace . . . 3
Odvození typu . . . 3
Odvození typového parametru . . . 3
Odvození typu lambda výrazu . . . 4

Proměnné

- Obecně slouží k pamatování si hodnot v paměti během chodu programu.
- Některé — a to lokální proměnné — si pamatují jen během volání podprogramu (metody) a pak zmizí.
- Jiné — objekty a prvky objektů — přetrvávají, jak dlouho potřebujeme, nejdéle do konce běhu programu.
- Trvalé ukládání dat už se děje jinde — na vnějších pamětech, discích, NVRam.

Pojmenování proměnných

- Proměnné se stejně jako pro metody a třídy pojmenovávají pomocí *identifikátorů*.
- Identifikátor podobně jako např. v Pythonu musí:
- začínat *písmenem* nebo znakem podtržítka `_`, které ovšem raději nepoužívejme, není to v Javě zvykem

Konvence v Javě na rozdíl od Pythonu

názvy tříd a dalších typů, třeba výčtů, a konstant

začínáme písmenem velkým — třída `Person`, konstanta `MAX_COUNT`

atributy, proměnné, metody…

malým písmenem — atribut `numHeads`, metoda `print`

víceslovné názvy

namísto podtržítek jako v Pythonu (`my_variable`) píšeme `myVariable`; kromě konstant, kde píšeme velkými písmeny a slova oddělujeme podtržítky (`MAX_COUNT`)

Kategorie proměnných

V Javě jsou proměnné realizovány jako místa v paměti nesoucí hodnotu nebo odkaz na objekt. Rozlišujeme tyto kategorie podle místa výskytu proměnné:

atributy

(nebo též *proměnné objektu*, *instanční proměnné*)

statické proměnné

(nebo též *statické atributy* nebo *proměnné třídy*)

lokální proměnné

(místní proměnné, *proměnné v metodě*)

Realizace proměnných

atributy

uloženy v rámci dat objektu, tedy na haldě

statické proměnné

uloženy v rámci dat třídy, tedy taky na haldě, ale v celém programu jen jednou

lokální proměnné

na zásobníku; alokují se při volání metody (tzv. *automatické proměnné*)

Lokální proměnné

- Ve příkladu s bankovními účty se v metodě `main` objevily proměnné (účty), které nebyly atributy objektů, ale pracovalo se s nimi pouze v metodě samotné.
- Takové proměnné se označují podobně jako v jiných jazycích jako *lokální*.
- Podobně jako atributy mají i tyto proměnné svůj datový typ - primitivní nebo objektový.
- V případě primitivního pak proměnná nese přímo hodnotu (například číslo nebo `boolean`).
- V případě objektového nese odkaz na objekt.
- V tomto ohledu se to tedy nijak neliší od atributů.

Jak to přesně je?

```
public static void createAccount() {
```

```
    Account petrsAccount = new Account();
    petrsAccount.add(100.0);
}
```

- `petrsAccount` je lokální (automatická) proměnná metody `createAccount`, tedy na zásobníku.
- Jedná se pouze o odkaz na objekt vytvořený při `new Account()`.
- Jakmile metoda `createAccount` skončí, zmizí hned i odkaz `petrsAccount`.
- Zdali zanikne i objekt vytvořeného Account, závisí na tom, zda objekt ještě odněkud jinud "vidím", mám na něj odkaz.
- V tomto případě již ne, takže v případě potřeby jej běhové prostředí uvolní (objekt zanikne).

Deklarace

Úplně klasicky vypadá například deklarace lokální proměnné takto:

- Primitivní typy (čísla, `boolean`...):

```
int i = 2;
boolean isOK = true
```

- Objektové typy:

```
Person jan = new Person("Honza");
// compiles iff Employee is a subclass of Person
Person petr = new Employee("Petr");
```

Odvození typu

- Doposud jsme viděli, že na levé straně byl uveden deklarovaný typ proměnné.
- V novějších verzích Javy (10+) lze využít tzn. *odvození typu* (type inference).
- Z typu výrazu na pravé straně odvodíme typ proměnné na levé straně s `var`:

```
var petr = new Employee("Petr Servus");
// so this does not compile - Employee expected:
petr = new Person("Petr Svatý");
```

Odvození typového parametru

- Kromě výše uvedených jednoduchých situací uvidíme později další.
- U tzv. parametrizovaných typů, např. seznamů, lze odvodit typ prvku seznamu z jedné strany na druhou - i zleva doprava.

```
List<Person> listPeople = new ArrayList<>();
// or the type on the left is inferred to ArrayList<Person>
var listPeople = new ArrayList<Person>();
```

Odvození typu lambda výrazu

- Lambda výrazy (konstrukty funkcionálního paradigmatu) jsou také typované.
- Funguje zde odvození typů - zde se odvodí, že v seznamu jsou osoby:

```
var listPeople = new ArrayList<Person>();
// type of the list item is Person
// inferred type of lambda is `(Person p) -> p.print()`:
listPeople.forEach(p -> p.print());
```