Proměnné, deklarace

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.

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`

podtržítka

: většinou nepoužíváme

víceslovné názvy

: namísto podtržítek jako v Pythonu (`my_variable`) píšeme v "CamelCase" (`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*)

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.

```
public static void main(String[] args) {
   Account petrsAccount = new Account();
   petrsAccount.add(100.0);
}
```

Lokální v Javě = automatické

- Liší se však okamžikem vytvoření: vytvoří se při každém zavolání metody (v případě objektové proměnné se samozřejmě vytvoří jen ten odkaz, nikoli celý objekt).
- Proto jsou také označovány jako *automatické*.
- Technicky jsou vytvořeny alokací místa na zásobníku, tedy podobným mechanismem, jako se ukládají návratové adresy při volání metod.

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 (8+) lze využít tzn. *odvození typu* (type inference).
- Z typu výrazu na pravé straně odvodíme typ proměnné na levé straně:

```
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());
```