7. Statické prvky třídy

V úvodních kapitolách jsme popsali deklaraci a používání datových atributů a metod instance. Jsou to nejčastěji používané součásti třídy, nicméně třída v Javě může obsahovat ještě další prvky. Nyní si popíšeme význam a použití statických datových atributů a statických metod.

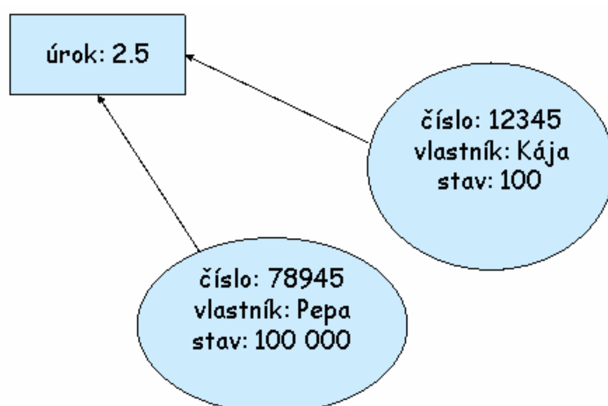
7.1. Statické datové atributy (statické proměnné, proměnné třídy)

Statický datový atribut (též **statická proměnná** či **proměnná třídy**) je společný pro všechny instance dané třídy. V deklaraci je za modifikátorem přístupu uveden modifikátor **static**.
Při vysvětlování statických proměnných budeme pokračovat v našem jednoduchém příkladě s účty. Budeme mít úrok z uložených peněz, který bude pro všechny účty stejný. Můžeme do každé instance přidat datový atribut *urok*, který bude mít vždy stejnou hodnotu. Při změně úroku však budeme muset zavolat metodu pro změnu úroku u každé instance.
Druhým a lepším řešením je použití statické proměnné, která bude v paměti pouze jednou a o které budou „vědět“ všechny instance. Tato proměnná vznikne při natažení kódu třídy *Ucet* do paměti, tj. ještě před vytvořením první instance. Deklaraci všech datových atributů včetně statického je v následujícím kódu.

```
public class Ucet {
     private static double urok = 2.5;
     private int cisloUctu;
     private String jmenoVlastnika;
     private double castka = 0;

  //zde následuje další kód třídy
}
```

Z jiných tříd se na statickou proměnnou odkazujeme se jménem třídy např. *Ucet.urok* (v našem případě tuto konstrukci nelze použít, protože proměnná třídy byla deklarována *private*). Uvnitř kódu třídy *Ucet* se na ni odkazujeme pouze pomocí identifikátoru.



Obrázek 7.1 Znázornění vztahu mezi instancemi třídy a statickým datovým atributem

Jak již bylo řečeno, statická proměnná existuje v paměti jen jednou a „vědí“ o ní všechny instance. Zjednodušeně je vztah instancí ke statické proměnné znázorněn na obrázku 7.1.
Statické proměnné se též používají pro vytváření **pojmenovaných konstant**, které jsou uvozeny modifikátory *public static final* (je to jedna z mála situací, kdy datové atributy nejsou *private*). Hodnotu pojmenovaných konstant nelze po přiřazení změnit (modifikátor *final*), proto mohou být veřejné. Použití modifikátoru *static* znamená, že tyto konstanty jsou v paměti uloženy pouze jednou. Ukázka deklarace takové konstanty vypadá následovně:

```
public static final double PI = 3.14;
```

Dalším příkladem pojmenovaných konstant mohou být konstanty ve třídě `Math` – hodnota pí (π) či Eulerova konstanta.

7.2. Statická metoda (metoda třídy)

Statická metoda (též se používá pojem **metoda třídy**) je metoda společná pro všechny instance, která má při deklaraci uveden modifikátor **static**. Statické metody jsou nezávislé na jakékoliv instanci třídy, tj.:

- pro použití není potřeba vytvořit žádnou instanci,
- statická metoda se nemůže přímo odkazovat na datové atributy instance či metody instance, ve statické metodě lze ale vytvořit instanci třídy a poslat ji zprávu,
- z instance lze přímo volat statické metody,
- statická metoda může přímo použít statické proměnné.

V našem příkladu vytvoříme statickou metodu na změnu úroku. Pro všechny účty najednou změníme výši úročení vkladů. Metoda se jmenuje *`setUrok()`* a má parametr typu *`double`*. Pojmenování *`setUrok()`* opět souvisí s konvencemi v Javě, kdy metoda, která slouží k nastavení nebo změně hodnoty datového atributu se pojmenovává *`setJmenoAtributu`*. Kromě metody pro změnu úroku bychom mohli napsat i statickou metodu *`getUrok()`*, která by vracela aktuální hodnotu úroku. I tato metoda by měla v hlavičce modifikátor *`static`*. Kód metody *`setUrok()`* vypadá takto:

```
public static void setUrok (double novyUrok) {
     urok = novyUrok;
}
```

Pro volání metody *`setUrok()`* není nutné mít vytvořenou instanci. Mimo kód třídy *`Ucet`* se metoda volá pomocí jména třídy a tečky tedy:

```
Ucet.setUrok(3.0);
```

Ve třídě *`Ucet`* (tj. v jiné metodě stejné třídy) se lze na tuto metodu odkazovat také celým jménem včetně názvu třídy, či pouze jménem metody – překladač jméno třídy doplní.
Statické metody se používají ve více situacích:

- pro změnu statického datového atributu (viz předchozí příklad), pro získání hodnoty statického datového atributu (pokud je *`private`*).
- pro provedení operace, u které není potřeba instance – příkladem mohou být matematické operace, např. sin, cos, druhá odmocnina. Třída *`Math`* a další příklady tohoto použití jsou v další části této kapitoly. Dalším příkladem těchto statických metod může být např. metoda *`parseInt()`* ve třídě *`Integer`*, která převede řetězec na proměnnou typu *`int`*.
- pro získání instance třídy v situaci, kdy pomocí přetížení konstruktoru nelze odlišit jednotlivé varianty vstupních parametrů. Příkladem může být konstruktor pro vytvoření trojúhelníka – nejsme schopni pomocí parametrů odlišit variantu, kdy jsou zadány tři strany od varianty, kdy zadáme dvě strany a jeden úhel (v obou případech budou zadána 3 reálná čísla). Proto použijeme statické metody, které se liší názvem. Bližší popis je v projektu *`Trojuhelniky`*.
- pro získání instance třídy v situaci, kdy na základě parametrů vrací statická metoda různé instance – toto souvisí s dědičností a polymorfismem. Příkladem může být získání ovladače k SQL databázi pomocí metody *`forName()`* ze třídy *`Class`*. Podrobnější popis této problematiky je mimo zaměření skript.

7.3. Statické datové atributy a metody ve třídách Math a System

7.3.1. Třída Math

Třída *`Math`* ze standardu Javy obsahuje pouze statické konstanty a statické metody. Třída nemá veřejný konstruktor, nelze tedy vytvořit instanci a ani to není smysluplné. Takto navržená třída se označuje jako utilita.

Dvě konstanty ze třídy `Math` jsou uvedeny v tabulce 7.1, vybrané statické metody ze třídy `Math` jsou v tabulce 7.2.

konstanty třídy	**význam**
Math.**PI**	hodnota pí (π)
Math.**E**	Eulerova konstanta přirozeného logaritmu

Tabulka 7.1 Konstanty třídy Math

statické metody ve třídě	**popis metody**
double Math.**abs**(double a)	absolutní hodnota
double Math.**ceil**(double a)	zaokrouhlení na nejbližší vyšší celé číslo
double Math.**floor**(double a)	zaokrouhlení na nejbližší nižší celé číslo
double Math.**rint**(double a)	zaokrouhlení na nejbližší celé číslo
double Math.**sin**(double a)	sin úhlu v radiánech
double Math.**cos**(double a)	cos úhlu v radiánech
double Math.**tan**(double a)	tangens úhlu v radiánech
double Math.**toDegrees**(double a)	převod úhlu v radiánech do stupňů
double Math.**toRadians**(double a)	převod úhlu ve stupních do radiánů
double Math.**pow**(double a, double b)	mocnina a^b
double Math.**sqrt**(double a)	druhá odmocnina
double Math.**log**(double a)	přirozený logaritmus
double Math.**log10**(double a)	desítkový logaritmus
double Math.**exp**(double a)	přirozená mocnina e^a

Tabulka 7.2 Přehled vybraných metod třídy Math

7.3.2. Třída System

Třída `System` z balíčku `java.lang` poskytuje tři statické proměnné sloužící pro vstup a výstup z/na konzoly.

statická proměnná	**popis**
InputStream System.**in**	vstup z konzoly
PrintStream System.**out**	výstup na konzolu
PrintStream System.**err**	chybový výstup

Tabulka 7.3 Přehled statických proměnných třídy System

Podrobný popis typů `InputStream` a `PrintStream` je uveden v kapitole věnované vstupu a výstupu. Zde si jenom uvedeme, že u třídy `PrintStream` se nejčastěji používají následující metody:

```
void println (String text)
void print (String text)
void printf(String format, Object ... args)
```

První metoda vypíše text a odřádkuje, metoda `print` nedoplňuje na konec výpisu odřádkování. Metoda `printf` umožňuje formátovat výstup – bližší popis je v kapitole věnované třídě `String`.

Pomocí statických metod `setIn()`, `setOut()` a `setErr()` mohou být tyto statické proměnné přesměrovány jinam. Např. v BlueJ jsou proměnné `System.in`, `System.out` a `System.err` přesměrovány do samostatného grafického okna pojmenovaného `Terminal`.
Další statické metody třídy `System` jsou uvedeny v tabulce 7.4.

hlavička metody	**popis**
void System.**exit**(int status)	ukončí aplikaci, jako parametr se zadává návratový kód (nula při úspěšném ukončení, kladná čísla pro označení chyby)
String System.**getEnv**(String name)	získání hodnoty environment proměnné (proměnné operačního systému), závisí na operačním systému
long System.**currentTimeMillis**()	získání aktuálního času v milisekundách, přesnost závisí na operačním systému
long System.**nanoTime**()	získání aktuálního času v nanosekundách, přesnost závisí na operačním systému
void System.**setSecurityManager** (SecurityManager s)	nastavení správce zabezpečení, toto téma je mimo rozsah těchto skript
Properties System.**getProperties**()	získání všech proměnných/vlastností JVM, mezi tyto vlastnosti patří verze JVM, identifikace operačního systému, domovský adresář uživatele
String System.**getProperty**(String name)	získání hodnoty konkrétní proměnné/vlastnosti JVM
String System.**setProperty**(String name, String value)	nastavení hodnoty konkrétní proměnné/vlastnosti JVM

Tabulka 7.4 Přehled vybraných metod třídy System

7.4. Standardní spouštění aplikace – metoda main

Aby bylo možno spustit Java aplikaci, musí existovat aspoň jeden vstupní bod do aplikace. V případě apletů se vytvoří instance hlavní třídy apletu a poté se volají metody `init()`, `start()`, `paint()`, `stop()` a `destroy()` v závislosti na stavu stránky s apletem. V případě servletů se vytvoří instance třídy a volá se metoda `doGet()`, `doPost()`, `doPut()`, `doDelete()` v závislosti na typu požadavku od klienta. Při spouštění Java aplikace v operačním systému se nevytváří automaticky instance, bylo zvoleno řešení s využitím statické metody `main()`.
Při spouštění JVM (příkaz `java` v operačním systému) říkáme, jaká třída má být natažena do paměti první:

```
    java MojeTrida
```

JVM hledá v kódu této třídy statickou metodu `main()` a tu spustí (pokud metoda `main()` neexistuje, tak JVM vypíše chybové hlášení). Metoda `main()` musí splňovat následující pravidla:

- metoda musí být veřejná (`public`), jinak by ji nebylo možné spustit,
- metoda musí být statická, není vytvořena žádná instance, pro kterou by bylo možné spustit metodu instance,
- metoda nevrací žádnou hodnotu (není možné nastavit žádnou proměnnou, do které by se uložil výsledek),
- metoda má jako vstupní parametr pole prvků typu `String` (jméno `args` je jediné, co lze v hlavičce metody změnit např. na `String parametry []`, ale obvykle se název pole parametrů nemění), do tohoto pole se ukládají parametry příkazové řádky.

Hlavička metody `main()` je následující:

```
          public static void main (String [] args)
```

Aplikace provede kód v metodě `main()` a skončí (pokud není aplikace ukončena dříve výskytem výjimky nebo metodou `System.exit()`). Pokud při zpracování metody `main()` vznikly vlákna, čeká se na jejich dokončení (vlákna vznikají např. při spuštění grafické aplikace). Následuje příklad metody `main()` v projektu `Trojuhelniky` (blíže viz kapitola 17):

```
     public static void main (String [] args) {
          Trojuhelniky troj = new Trojuhelniky();
          troj.zakladniCyklus();
     }
```

Při spouštění aplikace lze též zadávat parametry příkazové řádky. Jednotlivé parametry jsou odděleny mezerou, v následující příkladu jsou při spuštění aplikace tří parametry na příkazové řádce:

```
     java Pocitej 3 + 5
```

Tyto parametry jsou dostupné v poli, které předá JVM metodě `main()` při spuštění. V našem případě bude parametrem pole o třech prvcích. Parametry jsou uloženy jako řetězce (`String`).

Od verze 5.0 lze zapsat metodu `main()` s použitím proměnlivého počtu parametrů (viz kapitola 10.5):

```
   public static void main (String... args)
```

7.5. Počítání vytvořených instancí

Hezkým příkladem na používání statických datových atributů a statických metod je počítání vytvořených instancí nějaké třídy.

```
public class Pokus {
     private static long pocetInstanci = 0;
     public Pokus () {
          pocetInstanci++;
          // další příkazy konstruktoru
     }
     public static long getPocetInstanci() {
          return pocetInstanci;
     }
     // další části třídy
}
```

Je nutno zdůraznit, že se počítají vytvořené instance, ne instance existující[14]. Počítání existujících instancí je problematické, neboť není přesně určen okamžik, kdy instance přestane existovat. Pro účely počítání instancí můžeme za okamžik zrušení považovat chvíli, kdy Garbage Collector zavolá metodu `finalize()` – popis metody `finalize()` je v kapitole 6.6. Pro počítání existujících instancí je tudíž ještě potřeba překrýt metodu `finalize()` – celý kód bude vypadat následovně (klíčové slovo super je vysvětleno v kapitole 11.1 věnované dědičnosti):

```
public class Pokus {
     private static long pocetInstanci = 0;
     public Pokus () {
          pocetInstanci++;
          // další příkazy konstruktoru
     }
```

[14] Ani počet vytvořených instancí nemusí být přesný – pokud by třída `Pokus` měla potomky, tak i vytvoření instance potomka se započte mezi vytvořené instance třídy `Pokus` (v konstruktoru potomka se vždy volá konstruktor předka). Tj. počítáme vytvořené instance, které lze přetypovat na typ `Pokus`.

```
    public void finalize() {
        super.finalize();
        pocetInstanci--;
    }
    public static long getPocetInstanci() {
        return pocetInstanci;
    }
    // další části třídy
}
```

Uvedené kódy počítání instancí nefungují správně ve vláknech či při serializaci objektů – obě témata jsou však mimo rozsah těchto skript.