Základní pojmy OOP. Třída, objekt, jeho vlastnosti. Metody, proměnné. Konstruktory.

Obsah

Úvod do objektového programování

- Pojmy: třída, objekt
- Deklarace a definice tříd, jejich vlastnosti (proměnné, metody)
- Vytváření objektů (deklarace sama objekt nevytvoří...), proměnné odkazující na objekt
- Jmenné konvence - jak tvořit jména tříd, proměnných, metod
- Použití objektů, volání metod, přístupy k proměnným
- Modifikátory přístupu/viditelnosti (public, protected...)
- Konstruktory (dotvoří/naplní prázdný objekt)
- Přetěžování metod (dvě metody se stejným názvem a jinými parametry)

Třída, objekt, jejich vlastnosti

Co je třída a objekt?

Třída (také poněkud nepřesně zvaná *objektový typ*) představuje skupinu objektů, které nesou stejné vlastnosti

"stejné" je myšleno *kvalitativně*, nikoli *kvantitativně*, tj.

- např. všechny objekty třídy `Person`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Person] mají vlastnost `name`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=name],
- tato vlastnost má však obecně pro různé lidi různé hodnoty - lidi mají různá jména

Objekt je jeden konkrétní jedinec (reprezentant, entita) příslušné třídy

pro konkrétní objekt **nabývají vlastnosti** deklarované třídou **konkrétních hodnot**

Příklad:

- Třída `Person`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Person] má vlastnost `name`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=name]
- Objekt `panProfesor`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=panProfesor] typu `Person`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Person] má vlastnost `name`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=name] s hodnotou `"Václav Klaus"`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search="Václav Klaus"].

Vlastnosti objektu (1)

Vlastnostmi objektů jsou:

- *proměnné*
- *metody*

Vlastnosti objektů - proměnné i metody - je třeba *deklarovat*.

viz Sun Java Tutorial / Trail: Learning the Java Language: Lesson: Classes and Inheritance [http://java.sun.com/docs/books/tutorial/java/javaOO/classes.html]

Vlastnosti objektu (2)

Proměnné

1. jsou nositeli "pasivních" vlastností; jakýchsi *atributů*, *charakteristik* objektů
2. de facto jde o datové hodnoty svázané (zapouzdřené) v objektu

Metody

1. jsou nositeli "výkonných" vlastností; "dovedností" objektů
2. de facto jde o funkce (procedury) pracující (převážně) nad proměnnými objektu

Deklarace a použití třídy

Příklad - deklarace třídy Person

- deklarujme třídu objektů - lidí

```
public class Person {

   protected String name;
   protected int age;

   public Person(String n, int a) {
      name = n;
      age = a;
   }

   public void writeInfo() {
      System.out.println("Person:");
      System.out.println("Name "+name);
      System.out.println("Age  "+age);
   }
}
```

- Použijme ji v programu -
  1. vytvořme instanci - objekt - typu `Person`WIKIPEDIA
     [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Person]
  2. vypišme informace o něm

Příklad - použití třídy Person (1)

Mějme deklarovánu třídu *Person*

Metoda *main* v následujícím programu `Demo`WIKIPEDIA
[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Demo]:

1. vytvoří 2 lidi (pomocí *new Person*)
2. zavolá jejich metody
   `vypisInfo()`WIKIPEDIA
   [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=vypisInfo()]

```
public class Demo {
   public static void main(String[] args) {
      Person ales = new Person("Ales Necas", 38);
      Person beata = new Person("Beata Novakova", 36);
      ales.writeInfo();
      beata.writeInfo();
   }
}
```

Tedy: vypíší se informace o obou vytvořených objektech - lidech.

Nyní podrobněji k *proměnným* objektů.

Příklad - použití třídy Person (2)

Ve výše uvedeném programu znamenalo na řádku:

```
Person ales = new Person("Ales Necas", 38);
```

Person **ales**WIKIPEDIA

[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Person ales]: pouze deklarace (tj. určení typu) proměnné *ales* - bude typu *Person*.

ales = new Person ("Ales Necas", 38)WIKIPEDIA

[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=ales = new Person ("Ales Necas", 38)]: vytvoření objektu *Person* se jménem `Ales Necas`.

Lze napsat zvlášť do dvou řádků nebo (tak jak jsme to udělali) na řádek jeden.

Každý příkaz i deklaraci ukončujeme středníkem.

Vytváření objektů

Ve výše uvedených příkladech jsme objekty vytvářeli voláními **new Person(...)**WIKIPEDIA

[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=new Person(...)] bezděčně jsme tak použili

- **operátor new**, který *vytvoří prázdný objekt typu Person* a
- volání **konstruktoru**, který prázdný objekt *naplní počátečními údaji* (daty).

Shrnutí

Objekty:

- jsou instance "své" třídy
- vytváříme je operátorem **new**WIKIPEDIA

  [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=new] - voláním konstruktoru
- vytvořené objekty ukládáme do proměnné stejného typu (nebo typu předka či implementovaného rozhraní - o tom až později)

Proměnné

Proměnné - deklarace

Položky `name`WIKIPEDIA

[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=name] a `age`WIKIPEDIA

[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=age] v předchozím příkladu jsou **proměnné** objektu `Person`WIKIPEDIA

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

Jsou deklarovány v těle deklarace třídy *Person*.

Deklarace proměnné objektu má tvar:

`modifikátory TypProměnné jménoProměnné;`WIKIPEDIA

[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=modifikátory TypProměnné jménoProměnné;]

např.:

`protected int age;`WIKIPEDIA

[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=protected int age;]

Proměnné - datový typ

Výše uvedená proměnná `rokNarozeni`WIKIPEDIA

[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=rokNarozeni] měla datový typ *int* (32bitové celé číslo). Tedy:

- proměnná takového typu nese *jednu hodnotu typu celé číslo* (v rozsahu -2^31.. 2^31-1);
- nese-li jednu hodnotu, pak se jedná o tzv. **primitivní datový typ**.

Kromě celých čísel *int* nabízí Java celou řadu dalších primitivních datových typů. Primitivní typy jsou dané napevno, programátor je jen používá, nedefinuje.

Tam, kde nestačí diskrétní hodnoty (tj. primitivní typy), musíme použít typy *složené*, **objektové**.

- Objektovými typy v Javě jsou **třídy** (class) a **rozhraní** (interface). Třídy už jsme viděli v příkladu *Person*.
- Existují třídy definované přímo v Javě, v knihovně Java Core API.
- Nenajdeme-li tam třídu, kterou potřebujeme, můžeme si ji nadefinovat sami - viz *Person*.

Proměnné - jmenné konvence

Na jméno (identifikátor) proměnné sice Java neklade žádná speciální omezení (tedy mimo omezení platná pro jakýkoli identifikátor), ale přesto bývá velmi dobrým zvykem dodržovat při pojmenovávání následující pravidla (blíže viz podrobný rozbor na http://java.sun.com/docs/codeconv/):

- jména začínají malým písmenem
- nepoužíváme diakritiku (problémy s editory, přenositelností a kódováním znaků)
- (raději ani český jazyk, angličtině rozumí "každý")
- je-li to složenina více slov, pak je *nespojujeme podtržítkem*, ale další začne velkým písmenem (tzv. "CamelCase")

např.:

```
protected                    int                    rokNarozeni;
```

[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= protected int rokNarozeni;]

je identifikátor se správně (vhodně) utvořeným jménem, zatímco:

```
protected                    int                    RokNarozeni;
```

[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= protected int RokNarozeni;]

není vhodný identifikátor proměnné (začíná velkým písmenem)

Dodržování těchto jmenných konvencí je základem psaní srozumitelných programů a bude vyžadováno, sledováno a hodnoceno v odevzdávaných úlohách i písemkách.

Proměnné - použití

Proměnné objektu odkazujeme pomocí "tečkové notace":

```
public class Demo2 {
   public static void main(String[] args) {
      ...
      Person ales = new Person("Ales Necas", 38); // vytvoření objektu ...
      System.out.println(ales.name); // přístup k (čtení) jeho proměnné ...
      ales.name = "Aleš Novák"; // modifikace (zápis do) jeho proměnné
   }
}
```

Proměnné - modifikátory přístupu

Přístup k proměnným (i metodám) může být řízen uvedením tzv. *modifikátorů* před deklaraci prvku, viz výše:

```
// protected = přístup pouze z třídy ve stejném balíku nebo z podtřídy:
protected String name;
```

Modifikátorů je více typů, nejběžnější jsou právě zmíněné *modifikátory přístupu* (přístupových práv)

Proměnné - použití (2)

Objektů (tzv. *instancí*) stejného typu (tj. stejné třídy) si můžeme postupně vytvořit více:

```
public class Demo3 {
    public static void main(String[] args) {
       ...
       Person ales = new Person("Ales Necas", 38); // vytvoření prvniho objektu
       Person petr = new Person("Petr Svoboda", 36); // vytvoření druheho objektu
       System.out.println(ales.name); // přístup k (čtení) proměnné - prvnímu obje
       System.out.println(petr.name); // přístup k (čtení) proměnné - druhému obje
    }
}
```

Existují tady dva objekty, každý má své (obecně různé) hodnoty proměnných - např. jsou různá jména obou lidí.

Metody

Metody

Nad *existujícími* (vytvořenými) objekty můžeme volat jejich *metody*. Metoda je:

- podporgram (funkce, procedura), který *primárně pracuje s proměnnými "mateřského" objektu*,
- může mít další *parametry*

- může *vracet hodnotu* podobně jako v Pascalu *funkce*.

Každá metoda se musí ve své třídě *deklarovat*.

V Javě *neexistují metody deklarované mimo třídy* (tj. Java nezná žádné "globální" metody).

Metody - příklad

Výše uvedená třída `Person`WIKIPEDIA
The Free Encyclopedia
[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Person] měla metodu na výpis informací o daném objektu/člověku:

```
public class Person {

   protected String name;
   protected int age;

   public Person(String n, int a) {
      name = n;
      age = a;
   }

   public void writeInfo() {
      System.out.println("Person:");
      System.out.println("Name "+name);
      System.out.println("Age  "+age);
   }
}
```

Volání metod

- *Samotnou deklarací* (napsáním kódu) metody *se žádný kód neprovede*.
- Chceme-li vykonat kód metody, musíme ji *zavolat*.
- Volání se realizuje (tak jako u proměnných) "*tečkovou notací*", viz dále.
- Volání lze provést, jen je-li metoda z místa volání přístupná - "viditelná". Přístupnost regulují pdobně jako u proměnných modifikátory přístupu.

Volání metod - příklad

Vracíme se k prvnímu příkladu: vytvoříme dva lidi a zavoláme postupně jejich metodu *writeInfo*.

```
public class TestLidi {
   public static void main(String[] args) {
      Person ales = new Person("Ales Necas", 38);
      Person beata = new Person("Beata Novakova", 36);
      ales.writeInfo(); // volání metody objektu ales
      beata.writeInfo(); // volání metody objektu beata
   }
}
```

Vytvoří se dva objekty `Person`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Person] a vypíší se informace o nich.

Parametry metod

V deklaraci metody uvádíme v její hlavičce tzv. *formální parametry*.

Syntaxe:

modifikatory typVraceneHodnoty nazevMetody(*seznamFormalnichParametru*) {
  tělo (výkonný kód) metody
}

seznamFormalnichParametru je tvaru: `typParametru nazevFormalnihoParametru, ...`
WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=typParametru nazevFormalnihoParametru, ...]

Podobně jako v jiných jazycích parametr představuje v rámci metody *lokální proměnnou*.

Při volání metody jsou f. p. nahrazeny *skutečnými parametry*.

Předávání skutečných parametrů metodám

Hodnoty primitivních typů - čísla, logické hodnoty, znaky

- se předávají **hodnotou**, tj. hodnota se nakopíruje do lokální proměnné metody

Hodnoty objektových typů - všechny ostatní (tj. vč. všech uživatelem definovaných typů)

- se předávají **odkazem**, tj. do lokální proměnné metody se nakopíruje **odkaz na objekt - skutečný parametr**

  Pozn: ve skutečnosti se tedy parametry *vždy předávají hodnotou*, protože v případě objektových parametrů se předává *hodnota odkazu na objekt - skutečný parametr*.

V Javě tedy nemáme jako programátoři moc na výběr, jak parametry předávat

- to je ale spíše výhoda!

Příklad předávání parametrů - primitivní typy

Rozšiřme definici třídy *Person* o metodu *scream* s parametry:

```
...
public void scream(int kolikrat) {
   System.out.println("Kricim " + kolikrat + "krat UAAAA!");
}
...
```

Při zavolání:

```
...
scream(10);
...
```

tato metoda vypíše

```
Kricim 10krat UAAAA!
```

Příklad předávání parametrů - objektové typy (1)

Následující třída `Account` WIKIPEDIA The Free Encyclopedia [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Account] modeluje jednoduchý bankovní účet s možnostmi:

- přidávat na účet/odebírat z účtu
- vypisovat zůstatek na něm
- převádět na jiný účet

```
public class Account {
    // stav (zustatek) penez uctu
    protected double balance;

    public void add(double amount) {
        balance += amount;
    }
```

```
    public void writeBalance() {
        System.out.println(balance);
    }

    public void transferTo(Account whereTo, double amount) {
        balance -= amount;
        whereTo.add(amount);
    }
}
```

Metoda *transferTo* pracovat nejen se svým "mateřským" objektem, ale i s objektem *whereTo* předaným do metody... opět přes tečkovou notaci.

Příklad předávání parametrů - objektové typy (2)

Příklad použití třídy *Account*:

```
...
public static void main(String[] args) {
   Account petrsAccount = new Account();
   Account ivansAccount = new Account();
   petrsAccount.add(100);
   ivansAccount.add(220);
   petrsAccount.transferTo(ivansAccount, 50);
   petrsAccount.writeBalance();
   ivansAccount.writeBalance();
}
```

Návrat z metody

Kód metody skončí, tj. předá řízení zpět volající metodě (nebo systému - v případě startovní metody `main`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=main]), jakmile

- dokončí poslední příkaz v těle metody nebo
- dospěje k příkazu `return`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=return]

Metoda může při návratu *vrátit hodnotu* - tj. chovat se jako *funkce* (ve pascalském smyslu):

- Vrácenou hodnotu musíme uvést za příkazem **return**WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=return]. V tomto případě

tedy nesmí return chybět!

- Typ vrácené hodnoty musíme *v hlavičce metody deklarovat*.
- Nevrací-li metoda nic, pak musíme namísto typu vracené hodnoty psát `void`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=void].

Pozn.: I když metoda něco vrátí, my to nemusíme použít, ale je to trochu divné...

Konstruktory

Konstruktory

Co a k čemu jsou konstruktory?

- Konstruktury jsou speciální *metody* volané při *vytváření nových instancí* dané třídy.
- Typicky se v konstruktoru *naplní (inicializují) proměnné objektu.*
- Konstruktory lze volat jen ve spojení s operátorem `new`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=new] k vytvoření nové instance třídy - nového objektu, evt. volat z jiného konstruktoru

Syntaxe (viz výše):

```
public class Person {
   protected String name;
   protected int age;
   // konstruktor se dvěma parametry
   // - inicializuje hodnoty proměnných ve vytvořeném objektu
   public Person(String n, int a) {
      name = n;
      age = a;
   }
   ...
}
```

Příklad využití tohoto konstruktoru:

```
...
Person pepa = new Person("Pepa z Hongkongu", 105);
...
```

Toto volání vytvoří objekt pepa a naplní ho jménem a věkem.

Konstruktory (2)

Jak je psát a co s nimi lze dělat?

- nemají návratový typ (ani `void`WIKIPEDIA
  [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=void] - *to už vůbec ne*!!!)
- mohou mít parametry
- mohou volat konstruktor rodičovské třídy - ale jen jako svůj první příkaz

Přetěžování metod

Přetěžování

Jedna třída může mít:

- Více metod se *stejnými názvy, ale různými parametry*.
- Pak hovoříme o tzv. *přetížené* (overloaded) metodě.
- Nelze přetížit metodu *pouze změnou typu návratové hodnoty*.

Přetěžování - příklad

Ve třídě `Ucet`WIKIPEDIA
[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Ucet] přetížíme metodu `prevedNa`WIKIPEDIA
[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=prevedNa].

- Přetížená metoda převede na účet příjemce celý zůstatek z účtu odesílatele:

```
public void transferTo(Account whereTo) {
   whereTo.add(balance);
   balance = 0;
}
```

Ve třídě

Ucet

koexistují dvě různé metody se stejným názvem, ale jinými parametry.

Pozn: I když jsou to teoreticky dvě úplně různé metody, pak *když už se jmenují stejně, měly by dělat něco podobného*.

Přetěžování - příklad (2)

- Často přetížená metoda volá jinou "verzi" metody se stejným názvem:

```
public void transferTo(Account whereTo) {
   transferTo(whereTo, balance);
}
```

- Toto je *jednodušší, přehlednější*, udělá se tam potenciálně méně chyb.

  Lze doporučit. Je to přesně postup divide-et-impera, rozděl a panuj, dělba práce mezi metodami!

Přetěžování - příklad (3)

- Je ale otázka, zdali převod celého zůstatku raději nenapsat jako *nepřetíženou*, samostatnou metodu, např.:

```
public void transferAllMoneyTo(Account whereTo) {
   transferTo(whereTo, balance);
}
```

- Je to o něco instruktivnější, ale přibude další identifikátor - název metody - k zapamatování.

  Což může být výhoda (je to výstižné) i nevýhoda (musíme si pamatovat další).

Odkazy na objekty (instance)

Odkazy na objekty (instance)

Deklarace proměnné objektového typu ještě žádný objekt nevytváří.

To se děje až příkazem - operátorem - **new**WIKIPEDIA The Free Encyclopedia [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=new].

- Proměnné objektového typu jsou vlastně **odkazy** *na dynamicky vytvořené objekty*.
- Přiřazením takové proměnné zkopírujeme pouze odkaz. Nezduplikujeme celý objekt!

Přiřazování objektových proměnných

V následující ukázce vytvoříme dva účty.

- Odkazy na ně budou primárně v proměnných *petruvUcet* a *ivanuvUcet*.
- V proměnné u nebude primárně odkaz na žádný účet.
- Pak do ní přiřadíme (**u = petruvUcet;** WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=u = petruvUcet;]) odkaz na objekt skrývající se pod odkazem *petruvUcet*.
- Od této chvíle můžeme s účtem *petruvUcet* manipulovat přes odkaz (proměnnou) *u*.

  Což se také děje: **u.prevedNa(ivanuvUcet, 50);** WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=u.prevedNa(ivanuvUcet, 50);]

```
...
  public static void main(String[] args) {
     Account petruvUcet = new Account();
     Account ivanuvUcet = new Account();
     Account u;
     petruvUcet.add(100);
     ivanuvUcet.add(220);
     u = petruvUcet;
     u.transferTo(ivanuvUcet, 50); //  odečte se z Petrova účtu
     petruvUcet.writeBalance(); // vypíše 50
     ivanuvUcet.writeBalance();
  }
```

Vracení odkazu na sebe

Metoda může vracet odkaz na objekt, nad nímž je volána pomocí

`return this;` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= return this;]

Příklad - upravený `Account` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Account] s metodou `transferTo` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=transferTo] vracející odkaz na sebe

```
public class Account {

   private double balance;
```

```
   public void add(double amt) {
      balance += amt;
   }

   public void writeBalance() {
      System.out.println(balance);
   }

   public Account transferTo(Account whereTo, double a) {
      add(-a);
      u.add(a);
      return this;
   }
}
```

Řetězení volání

Vracení odkazu na sebe (tj. na objekt, na němž se metoda volala) lze s výhodou využít k "řetězení" volání:

```
 ...
   public static void main(String[] args) {
      Account petruvUcet = new Account();
      Account ivanuvUcet = new Account();
      Account igoruvUcet = new Account();
      petruvUcet.add(100);
      ivanuvUcet.add(100);
      igoruvUcet.add(100);

// budeme řetězit volání:
      petruvUcet.transferTo(ivanuvUcet, 50).transferTo(igoruvUcet, 20);
      petruvUcet.writeBalance(); // vypíše 30
      ivanuvUcet.writeBalance(); // vypíše 150
      igoruvUcet.writeBalance(); // vypíše 120
   }
```

Statické proměnné a metody

Proměnné a metody třídy - statické

Dosud jsme zmiňovali **proměnné a metody** (tj. souhrnně *prvky* - *members*) **objektu**.

Lze deklarovat také metody a proměnné patřící *celé třídě*, tj. skupině všech objektů daného typu. Takové

metody a proměnné nazýváme **statické** a označujeme v deklaraci modifikátorem `static` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=static]

Příklad statické proměnné a metody

Představme si, že si budeme pamatovat, kolik lidí se nám během chodu programu vytvořilo a vypisovat tento počet.

Budeme tedy potřebovat do třídy *Person* doplnit:

- jednu proměnnou pocetLidi společnou pro celou třídu Person - každý člověk ji při svém vzniku zvýší o jedna.
- jednu metodu howManyPeople, která vrátí počet dosud vytvořených lidí.

```
public class Person {

   protected String name;
   protected int a;
   protected static int peopleCount = 0;

   public Person(String n, int a) {
      name = n;
      age = a;
      peopleCount++;
   }
   ...
   public static int howManyPeople() {
      return peopleCount;
   }
   ...
}
```

Pozn: Všimněte si v obou případech modifikátoru/klíčového slova *static*.