Základní pojmy OOP. T#ída, objekt, jeho vlastnosti. Metody, prom#nné. Konstruktory.

Obsah

T#ída, objekt, jejich vlastnosti 2
Co je t#ída a objekt? 2
Vlastnosti objektu (1) 3
Vlastnosti objektu (2) 3
Deklarace a použití t#ídy 3
P#íklad - deklarace t#ídy Person 3
P#íklad - použití t#ídy Person (1) 4
P#íklad - použití t#ídy Person (2) 4
Vytvá#ení objekt# 5
Shrnutí 5
Prom#nné 5
Prom#nné - deklarace 5
Prom#nné - datový typ 5
Prom#nné - jmenné konvence 6
Prom#nné - použití 7
Prom#nné - modifikátory p#ístupu 7
Prom#nné - použití (2) 7
Metody 7
Metody 7
Metody - p#íklad 8
Volání metod 8
Volání metod - p#íklad 8
Parametry metod 9
P#edávání skute#ných parametr# metodám 9
P#íklad p#edávání parametr# - primitivní typy 9
P#íklad p#edávání parametr# - objektové typy (1) 10
P#íklad p#edávání parametr# - objektové typy (2) 10
Návrat z metody 11
Konstruktory 11
Konstruktory 11
Konstruktory (2) 12
P#et#žování metod 12
P#et#žování 12
P#et#žování - p#íklad 12
P#et#žování - p#íklad (2) 13
P#et#žování - p#íklad (3) 13
Odkazy na objekty (instance) 13
Odkazy na objekty (instance) 13
P#i#azování objektových prom#nných 13
Vracení odkazu na sebe 14
#et#zení volání 15
Statické prom#nné a metody 15
Prom#nné a metody t#ídy - statické 15
P#íklad statické prom#nné a metody 15

Úvod do objektového programování

- Pojmy: t#ída, objekt
- Deklarace a definice t#íd, jejich vlastnosti (prom#nné, metody)
- Vytvá#ení objekt# (deklarace sama objekt nevytvo#í...), prom#nné odkazující na objekt
- Jmenné konvence - jak tvo#it jména t#íd, prom#nných, metod
- Použití objekt#, volání metod, p#ístupy k prom#nným
- Modifikátory p#ístupu/viditelnosti (public, protected...)
- Konstruktory (dotvo#í/naplní prázdný objekt)
- P#et#žování metod (dv# metody se stejným názvem a jinými parametry)

T#ída, objekt, jejich vlastnosti

Co je t#ída a objekt?

T#ída (také pon#kud nep#esn# zvaná *objektový typ*) p#edstavuje skupinu objekt#, které nesou stejné vlastnosti

"stejné" je myšleno *kvalitativn#*, nikoli *kvantitativn#*, tj.

- nap#. všechny objekty t#ídy `Person` [http://www.google.com/search?q=Person] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Person] mají vlastnost `name` [http://www.google.com/search?q=name] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=name],
- tato vlastnost má však obecn# pro r#zné lidi r#zné hodnoty - lidi mají r#zná jména

Objekt je jeden konkrétní jedinec (reprezentant, entita) p#íslušné t#ídy

pro konkrétní objekt **nabývají vlastnosti** deklarované t#ídou **konkrétních hodnot**

P#íklad:

- T#ída `Person` [http://www.google.com/search?q=Person] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Person] má vlastnost `name` [http://www.google.com/search?q=name] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=name]
- Objekt `panProfesor` [http://www.google.com/search?q=panProfesor] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=panProfesor] typu `Person` [http://www.google.com/search?q=Person] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Person] má vlastnost `name` [http://www.google.com/search?q=name] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=name] s hodnotou `"Václav Klaus"` [http://www.google.com/search?q="Václav Klaus"] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search="Václav Klaus"].

Vlastnosti objektu (1)

Vlastnostmi objekt# jsou:

- *prom#nné*
- *metody*

Vlastnosti objekt# - prom#nné i metody - je t#eba *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)

Prom#nné

1. jsou nositeli "pasivních" vlastností; jakýchsi *atribut#*, *charakteristik* objekt#
2. de facto jde o datové hodnoty svázané (zapouzd#ené) v objektu

Metody

1. jsou nositeli "výkonných" vlastností; "dovedností" objekt#
2. de facto jde o funkce (procedury) pracující (p#evážn#) nad prom#nnými objektu

Deklarace a použití t#ídy

P#íklad - deklarace t#ídy Person

- deklarujme t#ídu objekt# - lidí

```
public class Person {

   private String name;
   private 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. vytvo#me instanci - objekt - typu `Person` [http://www.google.com/search?q=Person] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Person]
2. vypišme informace o n#m

P#íklad - použití t#ídy Person (1)

M#jme deklarovánu t#ídu *Person*

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

1. deklaruje dv# lokální prom#nné typu Person - budou obsahovat odkazy na následn# vytvo#ené objekty - lidi
2. vytvo#í tyto dva lidi (pomocí *new Person*)
3. zavolá jejich metody

`vypisInfo()` [http://www.google.com/search?q=vypisInfo()] [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 vytvo#ených objektech - lidech.

Nyní podrobn#ji k *prom#nným* objekt#.

P#íklad - použití t#ídy Person (2)

Ve výše uvedeném programu znamenalo na #ádku:

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

Person ales [http://www.google.com/search?q=Person ales] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Person ales]: pouze deklarace (tj. ur#ení typu) prom#nné *ales* - bude typu *Person*.

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

Lze napsat zvláš# do dvou #ádk# nebo (tak jak jsme to ud#lali) na #ádek jeden.

Každý p#íkaz i samostatn# stojící deklaraci ukon#ujeme st#edníkem.

Vytvá#ení objekt#

Ve výše uvedených p#íkladech jsme objekty vytvá#eli voláními **new Person(...)** **[http://www.google.com/search?q=new Person(...)]** [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=new Person(...)] bezd##n# jsme tak použili

- **operátor new**, který *vytvo#í prázdný objekt typu Person* a
- volání **konstruktoru**, který prázdný objekt *naplní po#áte#ními údaji* (daty).

Shrnutí

Objekty:

- jsou instance "své" t#ídy
- vytvá#íme je operátorem **new [http://www.google.com/search?q=new]** [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=new] - voláním konstruktoru
- vytvo#ené objekty ukládáme do prom#nné stejného typu (nebo typu p#edka #i implementovaného rozhraní - o tom až pozd#ji)

Prom#nné

Prom#nné - deklarace

Položky `name` `[http://www.google.com/search?q=name]` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=name] a `age` `[http://www.google.com/search?q=age]` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=age] v p#edchozím p#íkladu jsou **prom#nné** objektu `Person` `[http://www.google.com/search?q=Person]` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Person].

Jsou deklarovány v t#le deklarace t#ídy *Person*.

Deklarace prom#nné objektu má tvar:

`modifikátory TypProm#nné jménoProm#nné;` `[http://www.google.com/search?q=modifikátory TypProm#nné jménoProm#nné;]` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=modifikátory TypProm#nné jménoProm#nné;]

nap#.:

`private int age;` `[http://www.google.com/search?q=private int age;]` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=private int age;]

Prom#nné - datový typ

Výše uvedená prom#nná `rokNarozeni [http://www.google.com/search?q=rokNarozeni]` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=rokNarozeni] m#la datový typ *int* (32bitové celé #íslo). Tedy:

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

Krom# celých #ísel *int* nabízí Java celou #adu dalších primitivních datových typ#. Primitivní typy jsou dané napevno, programátor je jen používá, nedefinuje.

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

- Objektovými typy v Jav# jsou **t#ídy** (class) a **rozhraní** (interface). T#ídy už jsme vid#li v p#íkladu *Person*.
- Existují t#ídy definované p#ímo v Jav#, v knihovn# Java Core API.
- Nenajdeme-li tam t#ídu, kterou pot#ebujeme, m#žeme si ji nadefinovat sami - viz *Person*.

Prom#nné - jmenné konvence

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

- jména za#ínají malým písmenem
- nepoužíváme diakritiku (problémy s editory, p#enositelností a kódováním znak#) - a to p#esto, že Java ji i v identifikátorech povoluje
- (rad#ji ani #eský jazyk, angli#tin# rozumí "každý")
- je-li to složenina více slov, pak je *nespojujeme podtržítkem*, ale další za#ne velkým písmenem (tzv. "CamelCase")

nap#.:

`private int rokNarozeni; [http://www.google.com/search?q= private int rokNarozeni;]` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= private int rokNarozeni;]

je identifikátor se správn# (vhodn#) utvo#eným jménem, zatímco:

`private int RokNarozeni; [http://www.google.com/search?q= private int RokNarozeni;]` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search= private int RokNarozeni;]

není vhodný identifikátor prom#nné (za#íná velkým písmenem)

Dodržování t#chto jmenných konvencí je základem psaní srozumitelných program# a bude vyžadováno, sledováno a hodnoceno v odevzdávaných úlohách i písemkách.

Prom#nné - použití

Prom#nné objektu odkazujeme pomocí "te#kové notace":

```
public class Demo2 {
   public static void main(String[] args) {
      ...
      Person ales = new Person("Ales Necas", 38); // vytvo#ení objektu ...
      System.out.println(ales.name); // p#ístup k (#tení) jeho prom#nné ...
      ales.name = "Aleš Novák"; // modifikace (zápis do) jeho prom#nné
   }
}
```

Prom#nné - modifikátory p#ístupu

P#ístup k prom#nným (i metodám) m#že být #ízen uvedením tzv. *modifikátor#* p#ed deklaraci prvku, viz výše:

```
// private = p#ístup pouze z této t#ídy:
private String name;
```

Modifikátor# je více typ#, nejb#žn#jší jsou práv# zmín#né *modifikátory p#ístupu* (p#ístupových práv)

Prom#nné - použití (2)

Objekt# (tzv. *instancí*) stejného typu (tj. stejné t#ídy) si m#žeme postupn# vytvo#it více:

```
public class Demo3 {
    public static void main(String[] args) {
       ...
       Person ales = new Person("Ales Necas", 38); // vytvo#ení prvniho objektu
       Person petr = new Person("Petr Svoboda", 36); // vytvo#ení druheho objektu
       System.out.println(ales.name); // p#ístup k (#tení) prom#nné - prvnímu obje
       System.out.println(petr.name); // p#ístup k (#tení) prom#nné - druhému obje
    }
}
```

Existují tady dva objekty, každý má své (obecn# r#zné) hodnoty prom#nných - nap#. jsou r#zná jména obou lidí.

Metody

Metody

Nad *existujícími* (vytvo#enými) objekty m#žeme volat jejich *metody*. Metoda je:

- podprogram (funkce, procedura), který *primárn# pracuje s prom#nnými "mate#ského" objektu*,
- m#že mít další *parametry*
- m#že ve svém kódu (t#le) deklarovat *lokální prom#nné* - v našem p#íkladu metoda `main` deklarovala prom#nné `ales`, `petr`.
- m#že *vracet hodnotu* podobn# jako v Pascalu *funkce*.

Každá metoda se musí ve své t#íd# *deklarovat*.

Poznámka

V Jav# *neexistují metody deklarované mimo t#ídy* (tj. Java nezná žádné "globální" metody).

Metody - p#íklad

Výše uvedená t#ída `Person` [http://www.google.com/search?q=Person] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Person] m#la metodu na výpis informací o daném objektu/#lov#ku:

```
public class Person {

   private String name;
   private 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 prom#nných) "*te#kovou notací*", viz dále.
- Volání lze provést, jen je-li metoda z místa volání p#ístupná - "viditelná". P#ístupnost regulují pdobn# jako u prom#nných modifikátory p#ístupu.

Volání metod - p#íklad

Vracíme se k prvnímu p#íkladu: vytvo#íme dva lidi a zavoláme postupn# 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
   }
}
```

Vytvo#í se dva objekty `Person [http://www.google.com/search?q=Person]` [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 hlavi#ce tzv. *formální parametry*.

Syntaxe:

modifikatorytypVraceneHodnotynazevMetody(*seznamFormalnichParametru*) {
  t#lo (výkonný kód) metody
}

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

Podobn# jako v jiných jazycích parametr p#edstavuje v rámci metody *lokální prom#nnou*.

P#i volání metody jsou f. p. nahrazeny *skute#nými parametry*.

P#edávání skute#ných parametr# metodám

Hodnoty primitivních typ# - #ísla, logické hodnoty, znaky

- se p#edávají **hodnotou**, tj. hodnota se nakopíruje do lokální prom#nné metody

Hodnoty objektových typ# - všechny ostatní (tj. v#. všech uživatelem definovaných typ#)

- se p#edávají **odkazem**, tj. do lokální prom#nné metody se nakopíruje **odkaz na objekt - skute#ný parametr**

  Pozn: ve skute#nosti se tedy parametry *vždy p#edávají hodnotou*, protože v p#ípad# objektových parametr# se p#edává *hodnota odkazu na objekt - skute#ný parametr*.

V Jav# tedy nemáme jako programáto#i moc na výb#r, jak parametry p#edávat

- to je ale spíše výhoda!

P#íklad p#edávání parametr# - primitivní typy

Rozši#me definici t#ídy *Person* o metodu *scream* s parametry:

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

P#i zavolání:

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

tato metoda vypíše

```
Kricim 10krat UAAAA!
```

P#íklad p#edávání parametr# - objektové typy (1)

Následující t#ída `Account` [http://www.google.com/search?q=Account] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Account] modeluje jednoduchý bankovní ú#et s možnostmi:

- p#idávat na ú#et/odebírat z ú#tu
- vypisovat z#statek na n#m
- p#evád#t na jiný ú#et

```
public class Account {
    // stav (zustatek) penez uctu
    private 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 "mate#ským" objektem, ale i s objektem *whereTo* p#edaným do metody... op#t p#es te#kovou notaci.

P#íklad p#edávání parametr# - objektové typy (2)

P#íklad použití t#ídy *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 skon#í, tj. p#edá #ízení zp#t volající metod# (nebo systému - v p#ípad# startovní metody `main` [http://www.google.com/search?q=main] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=main]), jakmile

- dokon#í poslední p#íkaz v t#le metody nebo
- dosp#je k p#íkazu `return` [http://www.google.com/search?q=return] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=return]

Metoda m#že p#i návratu *vrátit hodnotu* - tj. chovat se jako *funkce* (ve pascalském smyslu):

- Vrácenou hodnotu musíme uvést za p#íkazem **return [http://www.google.com/search?q=return]** [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=return]. V tomto p#ípad# tedy nesmí return chyb#t!
- Typ vrácené hodnoty musíme *v hlavi#ce metody deklarovat.*
- Nevrací-li metoda nic, pak musíme namísto typu vracené hodnoty psát `void` [http://www.google.com/search?q=void] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=void].

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

Konstruktory

Konstruktory

Co a k #emu jsou konstruktory?

- Konstruktury jsou speciální *metody* volané p#i *vytvá#ení nových instancí* dané t#ídy.
- Typicky se v konstruktoru *naplní (inicializují) prom#nné objektu.*
- Konstruktory lze volat jen ve spojení s operátorem `new` [http://www.google.com/search?q=new] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=new] k vytvo#ení nové instance t#ídy - nového objektu, evt. volat z jiného konstruktoru

Syntaxe (viz výše):

```
public class Person {
   private String name;
   private int age;
   // konstruktor se dv#ma parametry
   // - inicializuje hodnoty prom#nných ve vytvo#eném objektu
   public Person(String n, int a) {
      name = n;
```

```
      age = a;
   }
   ...
}
```

P#íklad využití tohoto konstruktoru:

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

Toto volání vytvo#í objekt pepa a naplní ho jménem a v#kem.

Konstruktory (2)

Jak je psát a co s nimi lze d#lat?

- nemají návratový typ (ani `void` [http://www.google.com/search?q=void] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=void] - *to už v#bec ne*!!!)
- mohou mít parametry
- mohou volat konstruktor rodi#ovské t#ídy - ale jen jako sv#j první p#íkaz

P#et#žování metod

P#et#žování

Jedna t#ída m#že mít:

- Více metod se *stejnými názvy, ale r#znými parametry*.
- Pak hovo#íme o tzv. *p#etížené* (overloaded) metod#.
- Nelze p#etížit metodu *pouze zm#nou typu návratové hodnoty*.

P#et#žování - p#íklad

Ve t#íd# `Ucet` [http://www.google.com/search?q=Ucet] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Ucet] p#etížíme metodu `prevedNa` [http://www.google.com/search?q=prevedNa] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=prevedNa].

- P#etížená metoda p#evede na ú#et p#íjemce celý z#statek z ú#tu odesílatele:

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

Ve t#íd#

Ucet

koexistují dv# r#zné metody se stejným názvem, ale jinými parametry.

Pozn: I když jsou to teoreticky dv# úpln# r#zné metody, pak *když už se jmenují stejn#, m#ly by d#lat n#co podobného*.

P#et#žování - p#íklad (2)

- #asto p#etížená metoda volá jinou "verzi" metody se stejným názvem:

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

- Toto je *jednodušší, p#ehledn#jší*, ud#lá se tam potenciáln# mén# chyb.

  Lze doporu#it. Je to p#esn# postup divide-et-impera, rozd#l a panuj, d#lba práce mezi metodami!

P#et#žování - p#íklad (3)

- Je ale otázka, zdali p#evod celého z#statku rad#ji nenapsat jako *nep#etíženou*, samostatnou metodu, nap#.:

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

- Je to o n#co instruktivn#jší, ale p#ibude další identifikátor - název metody - k zapamatování.

  Což m#že být výhoda (je to výstižné) i nevýhoda (musíme si pamatovat další).

Odkazy na objekty (instance)

Odkazy na objekty (instance)

Deklarace prom#nné objektového typu ješt# žádný objekt nevytvá#í.

To se d#je až p#íkazem - operátorem - **new [http://www.google.com/search?q=new]** [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=new].

- Prom#nné objektového typu jsou vlastn# **odkazy***na dynamicky vytvo#ené objekty*.
- P#i#azením takové prom#nné zkopírujeme pouze odkaz. Nezduplikujeme celý objekt!

P#i#azování objektových prom#nných

V následující ukázce vytvo#íme dva ú#ty.

- Odkazy na n# budou primárn# v prom#nných *petruvUcet* a *ivanuvUcet*.
- V prom#nné u nebude primárn# odkaz na žádný ú#et.
- Pak do ní p#i#adíme (**u = petruvUcet; [http://www.google.com/search?q=u = petruvUcet;]** [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 m#žeme s ú#tem *petruvUcet* manipulovat p#es odkaz (prom#nnou) *u*.

  Což se také d#je: **u.prevedNa(ivanuvUcet, 50); [http://www.google.com/search?q=u.prevedNa(ivanuvUcet, 50);]** [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); //  ode#te se z Petrova ú#tu
     petruvUcet.writeBalance(); // vypíše 50
     ivanuvUcet.writeBalance();
  }
```

Vracení odkazu na sebe

Metoda m#že vracet odkaz na objekt, nad nímž je volána pomocí

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

P#íklad - upravený `Account` [http://www.google.com/search?q=Account] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Account] s metodou `transferTo` [http://www.google.com/search?q=transferTo] [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;
   }
}
```

#et#zení volání

Vracení odkazu na sebe (tj. na objekt, na n#mž se metoda volala) lze s výhodou využít k "#et#zení" 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 #et#zit volání:
      petruvUcet.transferTo(ivanuvUcet, 50).transferTo(igoruvUcet, 20);
      petruvUcet.writeBalance(); // vypíše 30
      ivanuvUcet.writeBalance(); // vypíše 150
      igoruvUcet.writeBalance(); // vypíše 120
   }
```

Statické prom#nné a metody

Prom#nné a metody t#ídy - statické

Dosud jsme zmi#ovali **prom#nné a metody** (tj. souhrnn# *prvky* - *members*) **objektu**.

Lze deklarovat také metody a prom#nné pat#ící *celé t#íd#*, tj. skupin# všech objekt# daného typu - statická prom#nná existuje pro jednu *t#ídu* jen jednou!

Takové metody a prom#nné nazýváme **statické** a ozna#ujeme v deklaraci modifikátorem `static` [http://www.google.com/search?q=static]
[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=static]

P#íklad statické prom#nné a metody

P#edstavme si, že si budeme pamatovat, kolik lidí se nám b#hem chodu programu vytvo#ilo a vypisovat tento po#et.

Budeme tedy pot#ebovat do t#ídy *Person* doplnit:

- jednu prom#nnou `peopleCount` spole#nou pro celou t#ídu `Person` - každý #lov#k ji p#i svém vzniku zvýší o jedna.
- jednu metodu `howManyPeople`, která vrátí po#et dosud vytvo#ených lidí.

```
public class Person {

   private String name;
```

```
   private int age;
   private static int peopleCount = 0;

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

Pozn: Všimn#te si v obou p#ípadech modifikátoru/klí#ového slova *static*.