4. Třída versus objekt

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 `Clovek` mají vlastnost `jmeno`,
- 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 `Clovek` má vlastnost `jmeno`
- Objekt `panProfesor` typu `Clovek` má vlastnost `jmeno` s hodnotou `"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

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

Příklad - deklarace třídy Clovek

- deklarujme třídu objektů - lidí

```
public class Clovek {
    protected String jmeno;
    protected int rokNarozeni;
    public Clovek(String j, int rN) {
```

```
        jmeno = j;
        rokNarozeni = rN;
    }
    public void vypisInfo() {
        System.out.println("Clovek:");
        System.out.println("Jmeno="+jmeno);
        System.out.println("Rok
  narozeni="+rokNarozeni);
    }
}
```

- Použijme ji v programu -
    1. vytvořme instanci - objekt - typu `Clovek`
    2. vypišme informace o něm

Příklad - použití třídy Clovek (1)

Mějme deklarovánu třídu `Clovek`

Metoda `main` v následujícím programu:

1. vytvoří 2 lidi (pomocí `new Clovek`)
2. zavolá jejich metody `vypisInfo()`

```
3.public class TestLidi {
4.    public static void main(String[] args) {
5.        Clovek ales = new Clovek("Ales Necas", 1966);
6.        Clovek beata = new Clovek("Beata Novakova", 1970);
7.        ales.vypisInfo();
8.        beata.vypisInfo();
9.    }
10.}
```

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

Nyní podrobněji k ***proměnným*** objektů.

Příklad - použití třídy Clovek (2)

Ve výše uvedeném programu znamenalo na řádku:

```
        Clovek ales = new Clovek("Ales Necas", 1966);
```

Clovek ales: pouze deklarace (tj. určení typu) proměnné `ales` - bude typu `Clovek`.

ales = new Clovek ("Ales Necas", 1966): vytvoření objektu `Clovek` 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.

Proměnné - deklarace

Položky `jmeno` a `rokNarozeni` v předchozím příkladu jsou **proměnné** objektu `Clovek`.

Jsou deklarovány v těle deklarace třídy `Clovek`.

Deklarace proměnné objektu má tvar:

modifikátoryTypjméno;

např.:

```
protected int rokNarozeni;
```

Proměnné - datový typ

Výše uvedená proměnná `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. Podrobněji viz Datové typy v Javě

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 `Clovek`.
- Existují třídy definované přímo v Javě, v knihovně Java Core API.
- Nenajdeme-li tam třídu, kterou potřebujeme, můžeme si ji nadefinovat sami - viz `Clovek`.

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 FIXME):

- 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;
```

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

```
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 TestLidi2 {
    public static void main(String[] args) {
        ...
        Clovek ales = new Clovek("Ales Necas", 1966); // vytvoření
objektu
        ...
        System.out.println(ales.jmeno); // přístup k (čtení) jeho
proměnné
        ...
        ales.jmeno = "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 jmeno;
```

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 TestLidi3 {
    public static void main(String[] args) {
        ...
        Clovek ales = new Clovek("Ales Necas", 1966);   // vytvoření
prvniho objektu
        Clovek petr = new Clovek("Petr Svoboda", 1968); // vytvoření
druheho objektu
        ...
        System.out.println(ales.jmeno); // přístup k (čtení) proměnné
prvniho
        System.out.println(petr.jmeno); // přístup k (čtení) proměnné
prvniho
    }
}
```

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

Vytváření objektů

Ve výše uvedených příkladech jsme objekty vytvářeli voláními **new Clovek(...)** bezděčně jsme tak použili

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

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 Clovek měla metodu na výpis informací o daném objektu/člověku:

```
public class Clovek {
    protected String jmeno;
    protected int rokNarozeni;
    public Clovek(String j, int rN) {
        jmeno = j;
        rokNarozeni = rN;
    }

    // Metoda vypisInfo() na výpis informací o člověku:
    public void vypisInfo() {
        System.out.println("Clovek:");
        System.out.println("Jmeno="+jmeno);
        System.out.println("Rok narozeni="+rokNarozeni);
    }
}
```

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 `vypisInfo`.

```
public class TestLidi {
    public static void main(String[] args) {

        Clovek ales = new Clovek("Ales Necas", 1966);
        Clovek beata = new Clovek("Beata Novakova", 1970);
```

```
        ales.vypisInfo();  // volání metody objektu ales
        beata.vypisInfo(); // volání metody objektu beata
    }
}
```

Vytvoří se dva objekty Clovek a vypíší se informace o nich.

Parametry metod

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

Syntaxe:

```
modifikatorytypVraceneHodnotynazevMetody
(
seznamFormalnichParametru
) {


tělo (výkonný kód) metody


}
```

seznamFormalnichParametru je tvaru: ***typParametrunazevFormalnihoParametru, ...***

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 `Clovek` o metodu `zakric` s parametry:

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

```
    ...
```

Při zavolání:

```
    ...
    zakric(10);
    ...
```

tato metoda vypíše

```
Kricim 10krat UAAAA!
```

Příklad předávání parametrů - objektové typy (1)

Následující třída Ucet modeluje jednoduchý bankovní účet s možnostmi:

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

```
public class Ucet {

    // stav (zustatek) penez uctu
    protected double zustatek;

    public void pridej(double castka) {
        zustatek += castka;
    }
    public void vypisZustatek() {
        System.out.println(zustatek);
    }
    public void prevedNa(Ucet kam, double castka) {
        zustatek -= castka;
        kam.pridej(castka);
    }
}
```

Metoda `prevedNa` pracovat nejen se svým "mateřským" objektem, ale i s objektem `kam` předaným do metody... opět přes tečkovou notaci.

Příklad předávání parametrů - objektové typy (2)

Příklad použití třídy `Ucet`:

```
    ...
    public static void main(String[] args) {
        Ucet petruvUcet = new Ucet();
        Ucet ivanuvUcet = new Ucet();
        petruvUcet.pridej(100);
        ivanuvUcet.pridej(220);
        petruvUcet.prevedNa(ivanuvUcet, 50);
        petruvUcet.vypisZustatek();
        ivanuvUcet.vypisZustatek();
    }
```

Návrat z metody

Kód metody skončí, tj. předá řízení zpět volající metodě (nebo systému - v případě startovní metody `main`), jakmile

- dokončí poslední příkaz v těle metody nebo
- dospěje k příkazu `return`

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

- Vrácenou hodnotu musíme uvést za příkazem **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`**.

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

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` k vytvoření nové instance třídy - nového objektu, evt. volat z jiného konstruktoru

Syntaxe (viz výše):

```
public class Clovek {
    protected String jmeno;
    protected int rokNarozeni;

    // konstruktor se dvěma parametry
    // – inicializuje hodnoty proměnných ve vytvořeném objektu
    public Clovek(String j, int rN) {
        jmeno = j;
        rokNarozeni = rn;
    }
    ...
}
```

Příklad využití tohoto konstruktoru:

```
    ...
    Clovek pepa = new Clovek("Pepa z Hongkongu", 1899);
    ...
```

Toto volání vytvoří objekt pepa a naplní ho jménem a rokem narození.

Konstruktory (2)

Jak je psát a co s nimi lze dělat?

- nemají návratový typ (ani `void` - ***to už vůbec ne***!!!)
- mohou mít parametry
- mohou volat konstruktor rodičovské třídy - ale jen jako svůj první příkaz

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` přetížíme metodu `prevedNa`.

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

```
public void prevedNa(Ucet u) {
    u.pridej(zustatek);
    zustatek = 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 prevedNa(Ucet u) {
    prevedNa(u, zustatek);
}
```

- 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 prevedVseNa(Ucet u) {
    prevedNa(u, zustatek);
}
```

- 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).

Shrnutí

Objekty:

- jsou instance "své" třídy
- vytváříme je operátorem **new** - voláním konstruktoru
- vytvořené objekty ukládáme do proměnné stejného typu (nebo typu předka či implementovaného rozhraní - o tom až později)

Odkazy na objekty

Deklarace proměnné objektového typu ještě žádný objekt nevytváří.

To se děje až příkazem - operátorem - **new**.

- Proměnné objektového typu jsou vlastně **odkaz*y na dynamicky vytvořené objekty***.
- Přiřazením takové proměnné zkopírujeme pouze odkaz, nikoli 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;**) 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);**

```
    ...
    public static void main(String[] args) {
        Ucet petruvUcet = new Ucet();
        Ucet ivanuvUcet = new Ucet();
        Ucet u;
        petruvUcet.pridej(100);
        ivanuvUcet.pridej(220);
        u = petruvUcet;
        u.prevedNa(ivanuvUcet, 50); // odečte se z Petrova účtu
        petruvUcet.vypisZustatek(); // vypíše 50
        ivanuvUcet.vypisZustatek();
    }
```

Vracení odkazu na sebe

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

`return this;`

Příklad - upravený `Ucet` s metodou `prevedNa` vracející odkaz na sebe

```
public class Ucet {
    float zustatek;
    public void pridej(float castka) {
        zustatek += castka;
    }
    public void vypisZustatek() {
        System.out.println(zustatek);
    }
    public Ucet prevedNa(Ucet u, float castka) {
        zustatek -= castka; // nebo také vhodné je: pridej(-castka);
        u.pridej(castka);
        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) {
        Ucet petruvUcet = new Ucet();
        Ucet ivanuvUcet = new Ucet();
        Ucet igoruvUcet = new Ucet();
        petruvUcet.pridej(100);
        ivanuvUcet.pridej(100);
        igoruvUcet.pridej(100);

        // budeme řetězit volání:
        petruvUcet.prevedNa(ivanuvUcet, 50).prevedNa(igoruvUcet, 20);

        petruvUcet.vypisZustatek(); // vypíše 30
        ivanuvUcet.vypisZustatek(); // vypíše 150
        igoruvUcet.vypisZustatek(); // vypíše 120
    }
```

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. Ttakové metody a proměnné nazýváme **statické** a označujeme v deklaraci modifikátorem `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 `Clovek` doplnit:

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

```
public class Clovek {
    protected String jmeno;
    protected int rokNarozeni;
    protected static int pocetLidi = 0;
    public Clovek(String j, int rN) {
        jmeno = j;
        rokNarozeni = rn;
        pocetLidi++;
    }
    ...
    public static int kolikMamLidi() {
        return pocetLidi;
```

```
    }
    ...
}
```

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

Úloha do cvičení 2.

Druhé cvičení vás pomůže nacvičit práci s více jednoduchými objekty několika tříd.

Vše, co vytvoříte i překopírujete, budete umisťovat do balíku `cz.muni.fi.{vaslogin}.banka` a odevzdáte podle pokynů cvičícího. ***Cvičící si může zadání upravit - toto je jen vzor.***

Úkolem bude:

Příklad 3.1.

1. Ze slidů této přednášky vzít a **"přivlastnit"** (tj. umístit) do svého balíku třídy `Clovek` a `Ucet`.
2. Třídu Ucet upravit tak, že bude mít **další proměnnou** `majitel` typu `Clovek`. Tato proměnná ponese odkaz na vlastníka účtu.

   Kromě toho bude mít třída Ucet **konstruktor** se dvěma parametry: ***majitelem účtu a počátečním stavem/zůstatkem***. Konstruktor si odkaz na majitele účtu pochopitelně zapamatuje v příslušné proměnné a zůstatek nastaví.

   Do třídy dále přidejte metodu **vypisInfo**, aby vypisovala informace o zůstatku a o vlastníkovi účtu.
3. Vytvořte dále třídu Banka s metodou **public Ucet vytvorUcet(Clovek maj, double pocatecni)**. Tato metoda vytvoří pro budoucího majitele maj nový účet a dá do něj počáteční vklad pocatecni. Současně přičte jedničku k celkového počtu zřízených účtů a tento celkový počet vypíše (přes `System.out.println`). Vytvořený účet vrátí.
4. Na základě výše uvedených deklarací napište do třídy Banka hlavní metodu programu (**`main`**, viz vzor minule) takovou, aby:
   - vytvořila jednu banku (např. b1)
   - vytvořila člověka (Petr Novotný, 1949) a (Jan Veselý, 1970)
   - v metodě main banka b1 vytvořila Petrovi voláním metody `vytvorUcet` dva účty (pu1: zůstatek 1000, pu2: zůstatek 50000) a Janovi jeden (ju: zůstatek 3000)
   - z Petrova druhého účtu se převede 1000 na Janův účet
   - z Janova účtu naopak 500 na Petrův první účet
   - vypíší se zůstatky na všech účtech