Struktura složitějších programů. Rozhraní. Dědičnost.

Obsah

Objektové modelování reality

- Kroky řešení problému na počítači - pár slov o SW inženýrství

Kroky řešení reálného problému na počítači

Generický (univerzální, obecný...) model postupu:

1. Zadání problému
2. Shromáždění informací o realitě a jejich analýza
3. Modelování reality na počítači a implementace požadovaných operací nad modelovanou realitou

Vývoj software je proces...

(podle JS, SW inženýrství):

1. při němž jsou **uživatelovy potřeby**
2. transformovány na **požadavky na software**,
3. tyto jsou transformovány na **návrh**,
4. návrh je implementován pomocí **kódu**,
5. kód je **testován**, **dokumentován** a **certifikován** pro operační použití.

Celkový rámec vývoje SW

V tomto předmětu nás z toho bude zajímat jen něco a jen částečně:

1. **Specifikace** (tj. zadání a jeho formalizace)

2. **Vývoj** (tj. návrh a vlastní programování)
3. částečně **Validace** (z ní především testování)

- 1. **Specifikace SW**: Je třeba definovat funkcionalitu SW a operační omezení.
  2. **Vývoj SW**: Je třeba vytvořit SW, který splňuje požadavky kladené ve specifikaci.
  3. **Validace SW**: SW musí být validován („kolaudován“), aby bylo potvrzeno, že řeší právě to, co požaduje uživatel.
  4. **Evoluce SW**: SW musí být dále rozvíjen, aby vyhověl měnícím se požadavkům zákazníka.

Metodiky vývoje SW

Tyto generické modely jsou dále rozpracovávány do podoby konkrétních *metodik*.

Metodika (tvorby SW) je ucelený soubor inženýrských postupů, jak řízeným způsobem, s odhadnutelnou spotřebou zdrojů dospět k použitelnému SW produktu.

Některé skupiny metodik:

- strukturovaná
- objektová
- ...

Metodika typu "vodopád"

Nevracím se nikdy o více jak jednu úroveň zpět:

1. Analýza (Analysis)
2. Návrh (Design)
3. Implementace (Implementation)
4. Testování (Testing)
5. Nasazení (Deployment)

Nyní zpět k Javě a jednoduchým programům...

Srovnání Java - Pascal

Co bude odlišné oproti dosavadním programátorským zkušenostem?

Struktura a rozsah programu:

Pascal program měl jeden nebo více zdrojových souborů (soubor = modul) tvořenými jednotlivými procedurami/fcemi, definicemi a deklaracemi (typů, proměnných...)

Java (a některé další OO jazyky): program je obvykle tvořen více soubory (soubor = popis jedné třídy) tvořenými deklaracemi metod a proměnných (případně dalších prvků) těchto tříd.

Organizace programových souborů

- *v Pascalu*: zdrojové (`.pas` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=.pas]) soubory, výsledný (jeden) spustitelný soubor (`.exe` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=.exe]), resp. přeložené kódy jednotek (`.tpu` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=.tpu])
- *v Javě*: zdrojové (`.java` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=.java]) soubory, přeložené soubory v bajtkódu (`.class` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=.class]) - jeden z nich spouštíme

Organizace zdrojových souborů

v Pascalu nebyla (nutná)

v Javě je nezbytná - zdrojové soubory organizujeme podle toho, ve kterých balících jsou třídy zařazeny

(přeložené soubory se *implicitně* ukládájí vedle zdrojových)

Shromáždění informací o realitě

Zjistíme, jaké *typy objektů* se ve zkoumaném výseku reality vyskytují a které potřebujeme

- *člověk, pes, veterinář*

Zjistíme a zachytíme vztahy mezi objekty našeho zájmu

- *člověk-chovatel vlastní psa*

Zjistíme, které činnosti objekty (aktéři, aktoři) provádějí

- *veterinář psa očkuje, pes štěká, kousne člověka...*

Jak zachytíme tyto informace

Jak zachytíme tyto informace:

- neformálními prostředky - tužkou na papíře vlastními slovy v přirozeném jazyce
- formálně pomocí nějakého vyjadřovacího aparátu - např. grafického jazyka
- pomocí CASE nástroje přímo na počítači

Zatím se přidržíme neformálního způsobu...

Modelování reality pomocí tříd

Určení základních **tříd**, tj.

skupin (kategorií) objektů, které mají podobné vlastnosti/schopnosti:

- `Person` WIKIPEDIA The Free Encyclopedia
  [http://cs.wikipedia.org/wiki/Speci%C3%A lln%C3%AD:Search?search=Person]
- `Account` WIKIPEDIA The Free Encyclopedia
  [http://cs.wikipedia.org/wiki/Speci%C3%A lln%C3%AD:Search?search=Account]
- ...

Rozhraní

Rozhraní

V Javě, na rozdíl od C++ neexistuje vícenásobná dědičnost -

- to nám ušetří řadu komplikací
- ale je třeba to něčím nahradit

Pokud po třídě chceme, aby disponovala vlastnostmi z několika různých množin (skupin), můžeme ji deklarovat tak, že

- implementuje více rozhraní

Co je rozhraní

- Rozhraní je vlastně *popis (specifikace) množiny vlastností*, *aniž* bychom tyto vlastnosti *ihned implementovali* Vlastnostmi zde rozumíme především *metody*.
- Říkáme, že určitá třída *implementuje rozhraní*, pokud implementuje (tedy *má* - přímo sama nebo podědí) všechny vlastnosti (tj. metody), které jsou daným rozhraním předepsány.
- Javové rozhraní je tedy *množina hlaviček metod označená identifikátorem* - názvem rozhraní. (a celých specifikací - tj. popisem, co přesně má metoda dělat - vstupy/výstupy metody, její vedlejší efekty...)

Deklarace rozhraní

- Vypadá i umisťuje se do souborů podobně jako deklarace třídy
- Všechny metody v rozhraní musí být `public`WIKIPEDIA

  [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=public] a v hlavičce se to ani nemusí uvádět.
- Těla metod v deklaraci rozhraní se nepíší. (Metody v rozhraní tudíž vypadají velmi podobně jako abstraktní metody ve třídách, ale nemusím psát `abstract`.)

Příklad deklarace rozhraní

```
public interface Informing {
   void writeInfo();
}
```

Implementace rozhraní

Příklad

```
public class Person implements Informing {
   ...
   public void writeInfo() {
      ...
```

```
    }
}
```

Čteme: Třída `Person` implementuje rozhraní `Informing`.

1. Třída v hlavičce uvede `implements NázevRozhraní` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=implements NázevRozhraní]
2. Třída implementuje všechny metody předepsané rozhraním

Využití rozhraní

1. Potřebujeme-li u jisté proměnné právě jen funkcionalitu popsanou určitým rozhraním,
2. tuto proměnnou můžeme pak deklarovat jako typu rozhraní - ne přímo objektu, který rozhraní implementuje.

Příklad

```
Informing petr = new Person("Petr Novák", 1945);
petr.writeInfo(); // "petr" stačí deklarovat jen jako Informing
                  // jiné metody než předepsané tímto intf.
                  // nepotřebujeme!
```

Dvě třídy implementující totéž rozhraní

Totéž rozhraní může implementovat více tříd, často konceptuálně zcela nesouvisejících:

- Rozhraní `Going` ("jdoucí") implementují dvě třídy:
- `Car` (auto má schopnost "jít", tedy jet)
- `Clock` (hodiny také "jdou")

Viz příklad - projekt v BlueJ - `car_clock` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=car_clock]

Dědičnost

Dědičnost

V realitě jsme často svědci toho, že třídy jsou **podtřídami** jiných:

- tj. všechny objekty podtřídy jsou zároveň objekty nadtřídy, např. každý objekt typu (třídy) `ChovatelPsu`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=ChovatelPsu] je současně typu `Clovek`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Clovek] nebo
- např. každý objekt typu (třídy) `Pes`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Pes] je současně typu `DomaciZvire`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=DomaciZvire] (alespoň v našem výseku reality - existují i psi "nedomácí"...)

Podtřída je tedy "zjemněním" nadtřídy:

- přebírá její vlastnosti a zpravidla přidává další, **rozšiřuje** svou nadtřídu/předka

V Javě je *každá* uživatelem definovaná třída potomkem nějaké jiné - neuvedeme-li předka explicitně, je předkem vestavěná třída `Object`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Object]

Terminologie dědičnosti

Terminologie:

- Nadtřídě (superclass) se také říká "(bezprostřední) předek", "rodičovská třída"
- Podtřídě (subclass) se také říká "(bezprostřední) potomek", "dceřinná třída"

Dědění může mít i více "generací", např.

`Person`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Person] <- `Employee`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Employee] <- `Manager`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Manager] (osoba je rodičovskou třídou zaměstnance, ten je rodičovskou třídou manažera)

Přeneseně tedy předkem (nikoli bezprostředním) manažera je člověk.

Jak zapisujeme dědění

Klíčovým slovem `extends` WIKIPEDIA

[http://cs.wikipedia.org/wiki/Speci%C3%A lln%C3%AD:Search?search= extends]:

```
public class Employee extends Person {
   ... popis vlastností (proměnných, metod...) zaměstnance navíc oproti člověku...
}
```

Dědičnost a vlastnosti tříd

Jak víme, třídy popisují skupiny objektů podobných vlastností

Třídy mohou mít tyto skupiny **vlastností**:

- Metody - procedury/funkce, které pracují (především) s objekty této třídy
- Proměnné - pojmenované datové prvky (hodnoty) uchovávané v každém objektu této třídy

Vlastnosti jsou ve třídě "schované", tzv. **zapouzdřené** (encapsulated)

Třída připomíná pascalský záznam (record), ten však zapouzdřuje jen proměnné, nikoli metody.

Dědičnost (alespoň v javovém smyslu) znamená, že dceřinná třída (podtřída, potomek)

- má *všechny* vlastnosti (metody, proměnné) nadtřídy
- + vlastnosti uvedené přímo v deklaraci podtřídy

Příklad

Cíl: vylepšit třídu `Ucet` WIKIPEDIA

[http://cs.wikipedia.org/wiki/Speci%C3%A lln%C3%AD:Search?search=Ucet]

Postup:

1. Zdokonalíme náš příklad s účtem tak, aby si účet "hlídal", kolik se z něj převádí peněz
2. Zdokonalenou verzi třídy `Account` WIKIPEDIA

   [http://cs.wikipedia.org/wiki/Speci%C3%A lln%C3%AD:Search?search=Account]nazveme `CreditAccount` WIKIPEDIA

   [http://cs.wikipedia.org/wiki/Speci%C3%A lln%C3%AD:Search?search=CreditAccount]

Příklad 1. Příklad kompletního zdrojového kódu třídy

ke stažení zde [http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/banka/KontokorentniUcet.java]

```
public class CreditAccount extends Account {
   // private double balance; znovu neuvádíme
   // ... zdědí se z nadtřídy/předka "Account"

   // kolik mohu "jít do mínusu"
   private double creditLimit;

   public void add(double amount) {
      if (balance + creditLimit + amount >= 0) {
         // zavoláme původní "neopatrnou" metodu
         super.add(amount);
      } else {
         System.err.println("Nelze odebrat částku " + (-amount));
      }
   }

   // public void writeInfo() ... zdědí se
   // public void transferTo(Account to, double amount) ... zdědí se
   // ... předpokládejme, že v třídě "Ucet" používáme variantu:
   // add(-amount);
   // to.add(amount);
   // }
}
```

Vzorový zdroják sám o sobě nepůjde přeložit, protože nemáme třídu, na níž závisí.

Příklad - co tam bylo nového

- Klíčové slovo `extends` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=extends] - značí, že třída `CreditAccount` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=CreditAccount] je potomkem/podtřídou/rozšířením/dceřinnou třídou (*subclass*) třídy `Account` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Account].
- Konstrukce `super.metoda(...);` WIKIPEDIA

[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=super.metoda(...);] značí, že je volána metoda rodičovské třídy/předka/nadtřídy (*superclass*). *Kdyby se nevolala překrytá metoda,* `super` *[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=super] by se neuvádělo.*

- Větvení `if() {...} else {...}` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=if() {...} else {...}] - složené závorky se používají k uzavření příkazů do sekvence - ve smyslu pascalského begin/end.

Další příklad

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

- výchozí třída `Account`
- podědíme do třídy `PrivateAccount` (osobní/privátní účet)
- zde přibude nová vlastnost - proměnná "vlastník" nesoucí odkaz na osobu vlastnící tento účet.

Do třetice - víceúrovňová dědičnost

Neplést s vícenásobnou - více úrovněmi myslíme častou situaci, kdy ze třídy odvodíme podtřídu, z ní zase podtřídu...

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

- výchozí třída `Account` (obyčejný účet)
- podědíme do třídy `PrivateAccount` (osobní/privátní účet)
- z ní podědíme do třídy `CheckedPrivateAccount` (osobní účet s kontrolou minimálního zůstatku)

Přístupová práva (viditelnost)

Přístupová práva

Přístup ke třídám i jejim prvkům lze (podobně jako např. v C++) regulovat:

- Přístupem se rozumí jakékoli použití dané třídy, prvku...

- Omezení přístupu je kontrolováno hned při překladu -> není-li přístup povolen, nelze program ani přeložit.
- Tímto způsobem lze regulovat přístup staticky, mezi celými třídami, nikoli pro jednotlivé objekty
- Jiný způsob zabezpečení představuje tzv. *security manager*, který lze aktivovat při spuštění JVM.

Granularita omezení přístupu

Přístup je v Javě regulován *jednotlivě po prvcích*

ne jako v C++ po blocích

Omezení přístupu je určeno uvedením jednoho z tzv. *modifikátoru přístupu* (*access modifier*) nebo naopak *neuvedením žádného*.

Typy omezení přístupu

- Existují čtyři možnosti:
  - `public`WIKIPEDIA
    [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=public] = veřejný
  - `protected`WIKIPEDIA
    [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=protected] = chráněný
  - *modifikátor neuveden* = říká se *lokální v balíku* nebo *chráněný v balíku* nebo "přátelský"
  - `private`WIKIPEDIA
    [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=private] = soukromý

Kde jsou která omezení aplikovatelná?

Pro *třídy*:

- veřejné - public
- neveřejné - lokální v balíku

Pro *vlastnosti tříd* = proměnné/metody:

- veřejné - public
- chráněné - protected
- neveřejné - lokální v balíku
- soukromé - private

Příklad - `public` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=public]

`public` [http://cs.wikipedia.org/wiki/Speci%C3%A lln%C3%AD:Search?search=public] => přístupné odevšad

```
public class Account {
   ...
}
```

třída `Account` [http://cs.wikipedia.org/wiki/Speci%C3%A lln%C3%AD:Search?search=Account]je veřejná = lze např.

- vytvořit objekt typu `Account` [http://cs.wikipedia.org/wiki/Speci%C3%A lln%C3%AD:Search?search=Account] i v metodě jiné třídy
- deklarovat podtřídu třídy `Account` [http://cs.wikipedia.org/wiki/Speci%C3%A lln%C3%AD:Search?search=Account] ve stejném i jiném balíku

Příklad - `protected` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=protected]

`protected` [http://cs.wikipedia.org/wiki/Speci%C3%A lln%C3%AD:Search?search=protected] => přístupné jen *z podtříd* a *ze tříd stejného balíku*

```
public class Account {
   // chráněná proměnná
```

```
   protected float creditLimit;
}
```

používá se jak pro metody (velmi často), tak pro proměnné (méně často)

Příklad - přátelský

lokální v balíku = přátelský=> přístupné jen*ze tříd stejného balíku,*už ale ne z podtříd, jsou-li v jiném balíku

```
public class Account {
   Date created; // přátelská proměnná
}
```

- používá se spíše u proměnných než metod, ale dost často se vyskytuje z lenosti programátora, kterému se nechce psát `protected`WIKIPEDIA

  [http://cs.wikipedia.org/wiki/Speci%C3%A lln%C3%AD:Search?search=protected]

- osobně moc nedoporučuji, protože svazuje přístupová práva s organizací do balíků (-> a ta se může přece jen měnit častěji než např. vztah `nadtřída-podtřída`WIKIPEDIA

  [http://cs.wikipedia.org/wiki/Speci%C3%A lln%C3%AD:Search?search=nadtřída-podtřída].)

- Mohlo by mít význam, je-li práce rozdělena na více lidí na jednom balíku pracuje jen jeden člověk - pak si může přátelským přístupem chránit své neveřejné prvky/třídy -> nesmí ovšem nikdo jiný chtít mé třídy rozšiřovat a používat přitom přátelské prvky.

- Používá se relativně často pro neveřejné třídy definované v jednom zdrojovém souboru se třídou veřejnou.

Příklad - private

`private`WIKIPEDIA

[http://cs.wikipedia.org/wiki/Speci%C3%A lln%C3%AD:Search?search=private] => přístupné jen v rámci třídy, ani v podtřídách - používá se častěji pro proměnné než metody

označením private prvek *zneviditelníme i případným podtřídám*!

```
public class Account {
   private String owner;
   ...
}
```

- proměnná `owner`WIKIPEDIA

[http://cs.wikipedia.org/wiki/Speci%C3%A lln%C3%AD:Search?search=owner] je soukromá = nelze k ní přímo přistoupit ani v podtřídě - je tedy třeba zpřístupnit proměnnou pro "vnější" potřeby jinak, např.

- přístupovými metodami `setOwner(String m)` WIKIPEDIA

  [http://cs.wikipedia.org/wiki/Speci%C3%A lln%C3%AD:Search?search=setOwner(String m)] a `String getOwner()` WIKIPEDIA

  [http://cs.wikipedia.org/wiki/Speci%C3%A lln%C3%AD:Search?search=String getOwner()]

Když si nevíte rady

Nastavení přístupových práv k třídě pomocí modifikátorů se děje na úrovni tříd, tj. vztahuje se pak na *všechny objekty příslušné třídy* i na její *statické vlastnosti* (proměnné, metody) atd.

Nastavení musí vycházet z povahy dotyčné proměnné či metody.

Nevíme-li si rady, jaká práva přidělit, řídíme se následujícím:

- metoda by měla být `public` WIKIPEDIA

  [http://cs.wikipedia.org/wiki/Speci%C3%A lln%C3%AD:Search?search=public], je-li užitečná i mimo třídu či balík - "navenek"

- jinak `protected` WIKIPEDIA

  [http://cs.wikipedia.org/wiki/Speci%C3%A lln%C3%AD:Search?search=protected]

- máme-li záruku, že metoda bude v případných podtřídách nepotřebná, může být `private` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A lln%C3%AD:Search?search=private] - *ale kdy tu záruku máme???*

- proměnná by měla být `private` WIKIPEDIA

  [http://cs.wikipedia.org/wiki/Speci%C3%A lln%C3%AD:Search?search=private], nebo `protected` WIKIPEDIA

  [http://cs.wikipedia.org/wiki/Speci%C3%A lln%C3%AD:Search?search=protected], je-li potřeba přímý přístup v podtřídě

- téměř nikdy bychom neměli deklarovat proměnné jako `public` WIKIPEDIA

  [http://cs.wikipedia.org/wiki/Speci%C3%A lln%C3%AD:Search?search=public]!

Přístupová práva a umístění deklarací do souborů

- Třídy deklarované jako *veřejné* (`public` WIKIPEDIA

[http://cs.wikipedia.org/wiki/Speci%C3%A lln%C3%AD:Search?search=public]) musí být umístěné do souborů s názvem totožným s názvem třídy (+přípona `.java`WIKIPEDIA

[http://cs.wikipedia.org/wiki/Speci%C3%A lln%C3%AD:Search?search=.java]) i na systémech Windows (vč. velikosti písmen)

- kromě takové třídy však může být v tomtéž souboru i libovolný počet deklarací neveřejných tříd
- `private`WIKIPEDIA

  [http://cs.wikipedia.org/wiki/Speci%C3%A lln%C3%AD:Search?search=private] nemají význam, ale *přátelské* ano

Organizace tříd do balíků

Zápis třídy do zdrojového souboru

Soubor `Person.java`WIKIPEDIA

[http://cs.wikipedia.org/wiki/Speci%C3%A lln%C3%AD:Search?search=erson.java] bude obsahovat (pozor na velká/malá písmena - v obsahu i názvu souboru):

```
public class Person {
   ... popis vlastností (proměnných, metod...) osoby ...
}
```

`public`WIKIPEDIA

[http://cs.wikipedia.org/wiki/Speci%C3%A lln%C3%AD:Search?search=public] značí, že třída je "veřejně" použitelná, tj. i mimo balík

Organizace tříd do balíků

Třídy zorganizujeme do balíků.

V balíku jsou vždy umístěny *související* třídy.

Co znamená související?

- třídy, jejichž **objekty spolupracují**
- třídy na podobné **úrovni abstrakce**
- třídy ze **stejné části reality**

Balíky

Balíky obvykle organizujeme do hierarchií, např.:

- `cz.muni.fi.pb162`WIKIPEDIA
  [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=cz.muni.fi.pb162]
- `cz.muni.fi.pb162.banking`WIKIPEDIA
  [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=cz.muni.fi.pb162.banking]
- `cz.muni.fi.pb162.banking.credit`WIKIPEDIA
  [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=cz.muni.fi.pb162.banking.credit]

Neplatí však, že by

- třídy "dceřinného" balíku (např. `cz.muni.fi.pb162.banking.credit`)
- byly zároveň třídami balíku "rodičovského" (`cz.muni.fi.pb162.banking`WIKIPEDIA
)
  !
  !
  [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=cz.muni.fi.pb162.banking]!

Hierarchie balíků má tedy význam spíše pro srozumitelnost a logické členění.

Příslušnost třídy k balíku

Deklarujeme ji syntaxí: **`package názevbalíku;`**WIKIPEDIA
[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=package názevbalíku;]

- Uvádíme obvykle jako *první* deklaraci v zdrojovém souboru;
- Příslušnost k balíku musíme současně *potvrdit správným umístěním* zdrojového souboru do adresářové struktury;
- např. zdrojový soubor třídy `Person`WIKIPEDIA
  [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Person] umístíme do podadresáře `cz\muni\fi\pb162`WIKIPEDIA
  [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=cz\muni\fi\pb162]
- Neuvedeme-li příslušnost k balíku, stane se třída součástí **implicitního balíku** - to však nelze pro jakékoli větší a/nebo znovupoužívané třídy či dokonce programy doporučit a zde nebude tolerováno!

Deklarace `import NázevTřídy` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=import NázevTřídy]

Deklarace import nesouvisí s děděním, ale s organizací tříd programu do balíků:

- Umožní odkazovat se *v rámci kódu jedné třídy na ostatní třídy*
- Syntaxe: `import názevtřídy;` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=import názevtřídy;]
- kde *názevtřídy* je uveden včetně názvu balíku
- Píšeme obvykle ihned po deklaraci příslušnosti k balíku (`package názevbalíku;` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=package názevbalíku;])

Import není nutné deklarovat mezi třídami téhož balíku!

Deklarace `import názevbalíku.*` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=import názevbalíku.*]

Pak lze používat všechny třídy z uvedeného balíku

Doporučuje se "import s hvězdičkou" nepoužívat:

- jinak nevíme nikdy s jistotou, ze kterého balíku se daná třída použila;
- i profesionálové to však někdy používají :-)
- lze tolerovat tam, kde používáme z určitého balíku většinu tříd;
- v tomto úvodním kurzu většinou tolerovat nebudeme!

"Hvězdičkou" *nezpřístupníme třídy z podbalíků*, např.

- `import cz.*` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=import cz.*] nezpřístupní třídu `cz.muni.fi.pb162.Person` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=cz.muni.fi.pb162.Person]