Struktura složit#jších program#. Rozhraní. D#di#nost.

Obsah

Objektové modelování reality

- Kroky #ešení problému na po#íta#i - pár slov o SW inženýrství

Kroky #ešení reálného problému na po#íta#i

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

1. Zadání problému
2. Shromážd#ní informací o realit# a jejich analýza
3. Modelování reality na po#íta#i a implementace požadovaných operací nad modelovanou realitou

Vývoj software je proces...

(podle JS, SW inženýrství):

1. p#i n#mž jsou **uživatelovy pot#eby**
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 opera#ní použití.

Celkový rámec vývoje SW

V tomto p#edm#tu nás z toho bude zajímat jen n#co a jen #áste#n#:

1. **Specifikace** (tj. zadání a jeho formalizace)
2. **Vývoj** (tj. návrh a vlastní programování)
3. #áste#n# **Validace** (z ní p#edevším testování)

- 1. **Specifikace SW**: Je t#eba definovat funkcionalitu SW a opera#ní omezení.
  2. **Vývoj SW**: Je t#eba vytvo#it SW, který spl#uje požadavky kladené ve specifikaci.
  3. **Validace SW**: SW musí být validován („kolaudován“), aby bylo potvrzeno, že #eší prám# to, co požaduje uživatel.
  4. **Evoluce SW**: SW musí být dále rozvíjen, aby vyhov#l m#nícím se požadavk#m zákazníka.

Metodiky vývoje SW

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

Metodika (tvorby SW) je ucelený soubor inženýrských postup#, jak #ízeným zp#sobem, s odhadnutelnou spot#ebou zdroj# dosp#t k použitelnému SW produktu.

N#které skupiny metodik:

- strukturovaná
- objektová
- ...

Metodika typu "vodopád"

Nevracím se nikdy o více jak jednu úrove# zp#t:

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

Nyní zp#t k Jav# a jednoduchým program#m...

Srovnání Java - Pascal

Co bude odlišné oproti dosavadním programátorským zkušenostem?

Struktura a rozsah programu:

Pascal
: program m#l jeden nebo více zdrojových soubor# (soubor = modul) tvo#enými jednotlivými procedurami/fcemi, definicemi a deklaracemi (typ#, prom#nných...)

Java
: (a n#které další OO jazyky): program je obvykle tvo#en více soubory (soubor = popis jedné t#ídy) tvo#enými deklaracemi metod a prom#nných (p#ípadn# dalších prvk#) t#chto t#íd.

Organizace programových soubor#

- *v Pascalu*: zdrojové (`.pas` [http://www.google.com/search?q=.pas] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=.pas]) soubory, výsledný (jeden) spustitelný soubor (`.exe` [http://www.google.com/search?q=.exe] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=.exe]), resp. p#ložené kódy jednotek (`.tpu` [http://www.google.com/search?q=.tpu] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=.tpu])

- *v Jav#*: zdrojové (`.java` [http://www.google.com/search?q=.java] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=.java]) soubory, p#eložené soubory v bajtkódu (`.class` [http://www.google.com/search?q=.class] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=.class]) - jeden z nich spouštíme

Organizace zdrojových soubor#

v Pascalu nebyla (nutná)

v Jav# je nezbytná - zdrojové soubory organizujeme podle toho, ve kterých balících jsou t#ídy za#azeny

(p#eložené soubory se *implicitn#* ukládájí vedle zdrojových)

Shromážd#ní informací o realit#

Zjistíme, jaké *typy objekt#* se ve zkoumaném výseku reality vyskytují a které pot#ebujeme

- *#lov#k, pes, veteriná#*

Zjistíme a zachytíme vztahy mezi objekty našeho zájmu

- *#lov#k-chovatel vlastní psa*

Zjistíme, které #innosti objekty (akté#i, akto#i) provád#jí

- *veteriná# psa o#kuje, pes št#ká, kousne #lov#ka...*

Jak zachytíme tyto informace

Jak zachytíme tyto informace:

- neformálními prost#edky - tužkou na papí#e vlastními slovy v p#irozeném jazyce
- formáln# pomocí n#jakého vyjad#ovacího aparátu - nap#. grafického jazyka
- pomocí CASE nástroje p#ímo na po#íta#i

Zatím se p#idržíme neformálního zp#sobu...

Modelování reality pomocí t#íd

Ur#ení základních **t#íd**, tj.

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

- `Person` [http://www.google.com/search?q=Person]

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

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

Rozhraní

Rozhraní

V Jav#, na rozdíl od C++ neexistuje vícenásobná d#di#nost -

- to nám ušet#í #adu komplikací
- ale je t#eba to n##ím nahradit

Pokud po t#íd# chceme, aby disponovala vlastnostmi z n#kolika r#zných množin (skupin), m#žeme ji deklarovat tak, že

- implementuje více rozhraní

Co je rozhraní

- Rozhraní je vlastn# *popis (specifikace) množiny vlastností*, *aniž* bychom tyto vlastnosti *ihned implementovali* Vlastnostmi zde rozumíme p#edevším *metody*.
- #íkáme, že ur#itá t#ída *implementuje rozhraní*, pokud implementuje (tedy *má* - p#ímo sama nebo pod#dí) všechny vlastnosti (tj. metody), které jsou daným rozhraním p#edepsány.
- Javové rozhraní je tedy *množina hlavi#ek metod ozna#ená identifikátorem* - názvem rozhraní. (a celých specifikací - tj. popisem, co p#esn# má metoda d#lat - vstupy/výstupy metody, její vedlejší efekty...)

Deklarace rozhraní

- Vypadá i umis#uje se do soubor# podobn# jako deklarace t#ídy
- Všechny metody v rozhraní musí být `public` [http://www.google.com/search?q=public] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=public] a v hlavi#ce se to ani nemusí uvád#t.
- T#la metod v deklaraci rozhraní se nepíší. (Metody v rozhraní tudíž vypadají velmi podobn# jako abstraktní metody ve t#ídách, ale nemusím psát `abstract`.)

P#íklad deklarace rozhraní

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

Implementace rozhraní

P#íklad

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

#teme: T#ída `Person` implementuje rozhraní `Informing`.

1. T#ída v hlavi#ce uvede `implements NázevRozhraní` [http://www.google.com/search?q=implements NázevRozhraní] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=implements NázevRozhraní]
2. T#ída implementuje všechny metody p#edepsané rozhraním

Využití rozhraní

1. Pot#ebujeme-li u jisté prom#nné práv# jen funkcionalitu popsanou ur#itým rozhraním,
2. tuto prom#nnou m#žeme pak deklarovat jako typu rozhraní - ne p#ímo objektu, který rozhraní implementuje.

P#íklad

```
Informing petr = new Person("Petr Novák", 1945);
petr.writeInfo(); // "petr" sta#í deklarovat jen jako Informing
                  // jiné metody než p#edepsané tímto intf.
                  // nepot#ebujeme!
```

Dv# t#ídy implementující totéž rozhraní

Totéž rozhraní m#že implementovat více t#íd, #asto konceptuáln# zcela nesouvisejících:

- Rozhraní `Going` ("jdoucí") implementují dv# t#ídy:
- `Car` (auto má schopnost "jít", tedy jet)
- `Clock` (hodiny také "jdou")

Viz p#íklad - projekt v BlueJ - `car_clock` [http://www.google.com/search?q=car_clock]

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

D#di#nost

D#di#nost

V realit# jsme #asto sv#dci toho, že t#ídy jsou **podt#ídami** jiných:

- tj. všechny objekty podt#ídy jsou zárove# objekty nadt#ídy, nap#. každý objekt typu (t#ídy) `ChovatelPsu` [http://www.google.com/search?q=ChovatelPsu] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=ChovatelPsu] je sou#asn# typu `Clovek` [http://www.google.com/search?q=Clovek] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Clovek] nebo
- nap#. každý objekt typu (t#ídy) `Pes` [http://www.google.com/search?q=Pes] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Pes] je sou#asn# typu `DomaciZvire` [http://www.google.com/search?q=DomaciZvire] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=DomaciZvire] (alespo# v našem výseku reality - existují i psi "nedomácí"...)

Podt#ída je tedy "zjemn#ním" nadt#ídy:

- p#ebírá její vlastnosti a zpravidla p#idává další, **rozši#uje** svou nadt#ídu/p#edka

V Jav# je *každá* uživatelem definovaná t#ída potomkem n#jaké jiné - neuvedeme-li p#edka explicitn#, je p#edkem vestav#ná t#ída `Object` [http://www.google.com/search?q=Object] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Object]

Terminologie d#di#nosti

Terminologie:

- Nadt#íd# (superclass) se také #íká "(bezprost#ední) p#edek", "rodi#ovská t#ída"
- Podt#íd# (subclass) se také #íká "(bezprost#ední) potomek", "dce#inná t#ída"

D#d#ní m#že mít i více "generací", nap#.

`Person` [http://www.google.com/search?q=Person] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Person] <- `Employee` [http://www.google.com/search?q=Employee] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Employee] <- `Manager` [http://www.google.com/search?q=Manager] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Manager] (osoba je rodi#ovskou t#ídou zam#stnance, ten je rodi#ovskou t#ídou manažera)

P#enesen# tedy p#edkem (nikoli bezprost#edním) manažera je #lov#k.

Jak zapisujeme d#d#ní

Klí#ovým slovem `extends` [http://www.google.com/search?q=extends]

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

```
public class Employee extends Person {
   ... popis vlastností (prom#nných, metod...) zam#stnance navíc oproti #lov#ku...
}
```

D#di#nost a vlastnosti t#íd

Jak víme, t#ídy popisují skupiny objekt# podobných vlastností

T#ídy mohou mít tyto skupiny **vlastností**:

- Metody - procedury/funkce, které pracují (p#edevším) s objekty této t#ídy
- Prom#nné - pojmenované datové prvky (hodnoty) uchovávané v každém objektu této t#ídy

Vlastnosti jsou ve t#íd# "schované", tzv. **zapouzd#ené** (encapsulated)

T#ída p#ipomíná pascalský záznam (record), ten však zapouzd#uje jen prom#nné, nikoli metody.

D#di#nost (alespo# v javovém smyslu) znamená, že dce#inná t#ída (podt#ída, potomek)

- má *všechny* vlastnosti (metody, prom#nné) nadt#ídy
- + vlastnosti uvedené p#ímo v deklaraci podt#ídy

P#íklad

Cíl: vylepšit t#ídu `Ucet` [http://www.google.com/search?q=Ucet] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Ucet]

Postup:

1. Zdokonalíme náš p#íklad s ú#tem tak, aby si ú#et "hlídal", kolik se z n#j p#evádí pen#z
2. Zdokonalenou verzi t#ídy `Account` [http://www.google.com/search?q=Account] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Account]nazveme `CreditAccount` [http://www.google.com/search?q=CreditAccount] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=CreditAccount]

P#íklad 1. P#íklad kompletního zdrojového kódu t#ídy

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
   // ... zd#dí se z nadt#ídy/p#edka "Account"
```

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

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

   // public void writeInfo() ... zd#dí se
   // public void transferTo(Account to, double amount) ... zd#dí se
   // ... p#edpokládejme, že v t#íd# "Ucet" používáme variantu:
   // add(-amount);
   // to.add(amount);
   // }
}
```

Vzorový zdroják sám o sob# nep#jde p#eložit, protože nemáme t#ídu, na níž závisí.

P#íklad - co tam bylo nového

- Klí#ové slovo `extends` [http://www.google.com/search?q=extends] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=extends] - zna#í, že t#ída `CreditAccount` [http://www.google.com/search?q=CreditAccount] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=CreditAccount] je potomkem/podt#ídou/rozší#ením/dce#innou t#ídou (*subclass*) t#ídy `Account` [http://www.google.com/search?q=Account] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Account].
- Konstrukce `super.metoda(...);` [http://www.google.com/search?q=super.metoda(...);] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=super.metoda(...);] zna#í, že je volána metoda rodi#ovské t#ídy/p#edka/nadt#ídy (*superclass*). *Kdyby se nevolala p#ekrytá metoda,* *`super`* *[http://www.google.com/search?q=super] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=super] by se neuvád#lo.*
- V#tvení `if() {...} else {...}` [http://www.google.com/search?q=if() {...} else {...}] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=if() {...} else {...}] - složené závorky se používají k uzav#ení p#íkaz# do sekvence - ve smyslu pascalského begin/end.

Další p#íklad

Demoprojekt `private_account` [http://www.google.com/search?q=private_account] http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=privat[e_account]:

- výchozí t#ída `Account`
- pod#díme do t#ídy `PrivateAccount` (osobní/privátní ú#et)

- zde p#ibude nová vlastnost - prom#nná "vlastník" nesoucí odkaz na osobu vlastnící tento ú#et.

Do t#etice - víceúrov#ová d#di#nost

Neplést s vícenásobnou - více úrovn#mi myslíme #astou situaci, kdy ze t#ídy odvodíme podt#ídu, z ní zase podt#ídu...

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

- výchozí t#ída `Account` (oby#ejný ú#et)
- pod#díme do t#ídy `PrivateAccount` (osobní/privátní ú#et)
- z ní pod#díme do t#ídy `CheckedPrivateAccount` (osobní ú#et s kontrolou minimálního z#statku)

Organizace t#íd do balík#

Zápis t#ídy do zdrojového souboru

Soubor `Person.java` [http://www.google.com/search?q=erson.java] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=erson.java] bude obsahovat (pozor na velká/malá písmena - v obsahu i názvu souboru):

```
public class Person {
   ... popis vlastností (prom#nných, metod...) osoby ...
}
```

`public` [http://www.google.com/search?q=public] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=public] zna#í, že t#ída je "ve#ejn#" použitelná, tj. i mimo balík

Organizace t#íd do balík#

T#ídy zorganizujeme do balík#.

V balíku jsou vždy umíst#ny *související* t#ídy.

Co znamená související?

- t#ídy, jejichž **objekty spolupracují**
- t#ídy na podobné **úrovni abstrakce**
- t#ídy ze **stejné #ásti reality**

Balíky

Balíky obvykle organizujeme do hierarchií, nap#.:

- `cz.muni.fi.pb162` [http://www.google.com/search?q=cz.muni.fi.pb162] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=cz.muni.fi.pb162]
- `cz.muni.fi.pb162.banking` [http://www.google.com/search?q=cz.muni.fi.pb162.banking] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=cz.muni.fi.pb162.banking]
- `cz.muni.fi.pb162.banking.credit` [http://www.google.com/search?q=cz.muni.fi.pb162.banking.credit] http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=cz.muni.fi.pb162.banking.cr [edit]

Neplatí však, že by

- t#ídy "dce#inného" balíku (nap#. `cz.muni.fi.pb162.banking.credit`)
- byly zárove# t#ídami balíku "rodi#ovského" (`cz.muni.fi.pb162.banking` [http://www.google.com/search?q=cz.muni.fi.pb162.banking])! [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=cz.muni.fi.pb162.banking]!!

Hierarchie balík# má tedy význam spíše pro srozumitelnost a logické #len#ní.

P#íslušnost t#ídy k balíku

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

- Uvádíme obvykle jako *první* deklaraci v zdrojovém souboru;
- P#íslušnost k balíku musíme sou#asn# *potvrdit správným umíst#ním* zdrojového souboru do adresá#ové struktury;
- nap#. zdrojový soubor t#ídy `Person` [http://www.google.com/search?q=Person] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Person] umístíme do podadresá#e `cz\muni\fi\pb162` [http://www.google.com/search?q=cz\muni\fi\pb162] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=cz\muni\fi\pb162]
- Neuvedeme-li p#íslušnost k balíku, stane se t#ída sou#ástí **implicitního balíku** - to však nelze pro jakékoli v#tší a/nebo znovupoužívané t#ídy #i dokonce programy doporu#it a zde nebude tolerováno!

Deklarace `import NázevT#ídy` [http://www.google.com/search?q=import NázevT#ídy] http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:S

earch?search=import NázevT#ídy]

Deklarace import nesouvisí s d#d#ním, ale s organizací t#íd programu do balík#:

- Umožní odkazovat se *v rámci kódu jedné t#ídy na ostatní t#ídy*
- Syntaxe: `import názevt#ídy;` [http://www.google.com/search?q=import názevt#ídy;] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=import názevt#ídy;]
- kde *názevt#ídy* je uveden v#etn# názvu balíku
- Píšeme obvykle ihned po deklaraci p#íslušnosti k balíku (`package názevbalíku;` [http://www.google.com/search?q=package názevbalíku;] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=package názevbalíku;])

Import není nutné deklarovat mezi t#ídami téhož balíku!

Deklarace `import názevbalíku.*` [http://www.google.com/search?q=import názevbalíku.*] http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:S [earch?search=import názevbalíku.*]

Pak lze používat všechny t#ídy z uvedeného balíku

Doporu#uje se "import s hv#zdi#kou" nepoužívat:

- jinak nevíme nikdy s jistotou, ze kterého balíku se daná t#ída použila;
- i profesionálové to však n#kdy používají :-)
- lze tolerovat tam, kde používáme z ur#itého balíku v#tšinu t#íd;
- v tomto úvodním kurzu v#tšinou tolerovat nebudeme!

"Hv#zdi#kou" *nezp#ístupníme t#ídy z podbalík#*, nap#.

- `import cz.*` [http://www.google.com/search?q=import cz.*] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=import cz.*] nezp#ístupní t#ídu `cz.muni.fi.pb162.Person` [http://www.google.com/search?q=cz.muni.fi.pb162.Person] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=cz.muni.fi.pb162.Person]

P#ístupová práva (viditelnost)

P#ístupová práva

P#ístup ke t#ídám i jejim prvk#m lze (podobn# jako nap#. v C++) regulovat:

- P#ístupem se rozumí jakékoli použití dané t#ídy, prvku...

- Omezení p#ístupu je kontrolováno hned p#i p#ekladu -> není-li p#ístup povolen, nelze program ani p#eložit.
- Tímto zp#sobem lze regulovat p#ístup staticky, mezi celými t#ídami, nikoli pro jednotlivé objekty
- Jiný zp#sob zabezpe#ení p#edstavuje tzv. *security manager*, který lze aktivovat p#i spušt#ní JVM.

Granularita omezení p#ístupu

P#ístup je v Jav# regulován *jednotliv# po prvcích*

ne jako v C++ po blocích

Omezení p#ístupu je ur#eno uvedením jednoho z tzv. *modifikátoru p#ístupu* (*access modifier*) nebo naopak *neuvedením žádného*.

Typy omezení p#ístupu

- Existují #ty#i možnosti:
  - `public` [http://www.google.com/search?q=public] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=public] = ve#ejný
  - `protected` [http://www.google.com/search?q=protected] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=protected] = chrán#ný
  - *modifikátor neuveden* = #íká se *lokální v balíku* nebo *chrán#ný v balíku* nebo "p#átelský"
  - `private` [http://www.google.com/search?q=private] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=private] = soukromý

Kde jsou která omezení aplikovatelná?

Pro *t#ídy*:

- ve#ejné - public
- neve#ejné - lokální v balíku

Pro *vlastnosti t#íd* = prom#nné/metody:

- ve#ejné - public
- chrán#né - protected

- neve#ejné - lokální v balíku
- soukromé - private

P#íklad - `public` [http://www.google.com/search?q=public] http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=public]

`public` [http://www.google.com/search?q=public] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=public] => p#ístupné odevšad

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

t#ída `Account` [http://www.google.com/search?q=Account] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Account]je ve#ejná = lze nap#.

- vytvo#it objekt typu `Account` [http://www.google.com/search?q=Account] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Account] i v metod# jiné t#ídy
- deklarovat podt#ídu t#ídy `Account` [http://www.google.com/search?q=Account] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Account] ve stejném i jiném balíku

P#íklad - `protected` [http://www.google.com/search?q=protected] http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=protected]

`protected` [http://www.google.com/search?q=protected] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=protected] => p#ístupné jen *z podt#íd* a *ze t#íd stejného balíku*

```
public class Account {
   // chrán#ná prom#nná
   protected float creditLimit;
}
```

používá se jak pro metody (velmi #asto), tak pro prom#nné (mén# #asto)

P#íklad - p#átelský

lokální v balíku = p#átelský=> p#ístupné jen*ze t#íd stejného balíku,*už ale ne z podt#íd, jsou-li v jiném balíku

```
public class Account {
   Date created; // p#átelská prom#nná
}
```

- používá se spíše u prom#nných než metod, ale dost #asto se vyskytuje z lenosti programátora, kterému se nechce psát `protected [http://www.google.com/search?q=protected]` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=protected]
- osobn# moc nedoporu#uji, protože svazuje p#ístupová práva s organizací do balík# (-> a ta se m#že p#ece jen m#nit #ast#ji než nap#. vztah `nadt#ída-podt#ída [http://www.google.com/search?q=nadt#ída-podt#ída]` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=nadt#ída-podt#ída].)
- Mohlo by mít význam, je-li práce rozd#lena na více lidí na jednom balíku pracuje jen jeden #lov#k - pak si m#že p#átelským p#ístupem chránit své neve#ejné prvky/t#ídy -> nesmí ovšem nikdo jiný chtít mé t#ídy rozši#ovat a používat p#itom p#átelské prvky.
- Používá se relativn# #asto pro neve#ejné t#ídy definované v jednom zdrojovém souboru se t#ídou ve#ejnou.

P#íklad - private

`private [http://www.google.com/search?q=private]` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=private] => p#ístupné jen v rámci t#ídy, ani v podt#ídách - používá se #ast#ji pro prom#nné než metody

ozna#ením private prvek *zneviditelníme i p#ípadným podt#ídám*!

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

- prom#nná `owner [http://www.google.com/search?q=owner]` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=owner] je soukromá = nelze k ní p#ímo p#istoupit ani v podt#íd# - je tedy t#eba zp#ístupnit prom#nnou pro "vn#jší" pot#eby jinak, nap#.
- p#ístupovými metodami `setOwner(String m) [http://www.google.com/search?q=setOwner(String m)]` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=setOwner(String m)] a `String getOwner() [http://www.google.com/search?q=String getOwner()]` [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=String getOwner()]

Když si nevíte rady

Nastavení p#ístupových práv k t#íd# pomocí modifikátor# se d#je na úrovni t#íd, tj. vztahuje se pak na *všechny objekty p#íslušné t#ídy* i na její *statické vlastnosti* (prom#nné, metody) atd.

Nastavení musí vycházet z povahy doty#né prom#nné #i metody.

Nevíme-li si rady, jaká práva p#id#lit, #ídíme se následujícím:

- metoda by m#la být `public` [http://www.google.com/search?q=public] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=public], je-li užite#ná i mimo t#ídu #i balík - "navenek"
- jinak `protected` [http://www.google.com/search?q=protected] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=protected]
- máme-li záruku, že metoda bude v p#ípadných podt#ídách nepot#ebná, m#že být `private` [http://www.google.com/search?q=private] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=private] - *ale kdy tu záruku máme???*
- prom#nná by m#la být `private` [http://www.google.com/search?q=private] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=private], nebo `protected` [http://www.google.com/search?q=protected] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=protected], je-li pot#eba p#ímý p#ístup v podt#íd#
- tém## nikdy bychom nem#li deklarovat prom#nné jako `public` [http://www.google.com/search?q=public] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=public]!

P#ístupová práva a umíst#ní deklarací do soubor#

- T#ídy deklarované jako *ve#ejné* (`public` [http://www.google.com/search?q=public] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=public]) musí být umíst#né do soubor# s názvem totožným s názvem t#ídy (+p#ípona `.java` [http://www.google.com/search?q=.java] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=.java]) i na systémech Windows (v#. velikosti písmen)
- krom# takové t#ídy však m#že být v tomtéž souboru i libovolný po#et deklarací neve#ejných t#íd
- `private` [http://www.google.com/search?q=private] [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=private] nemají význam, ale *p#átelské* ano