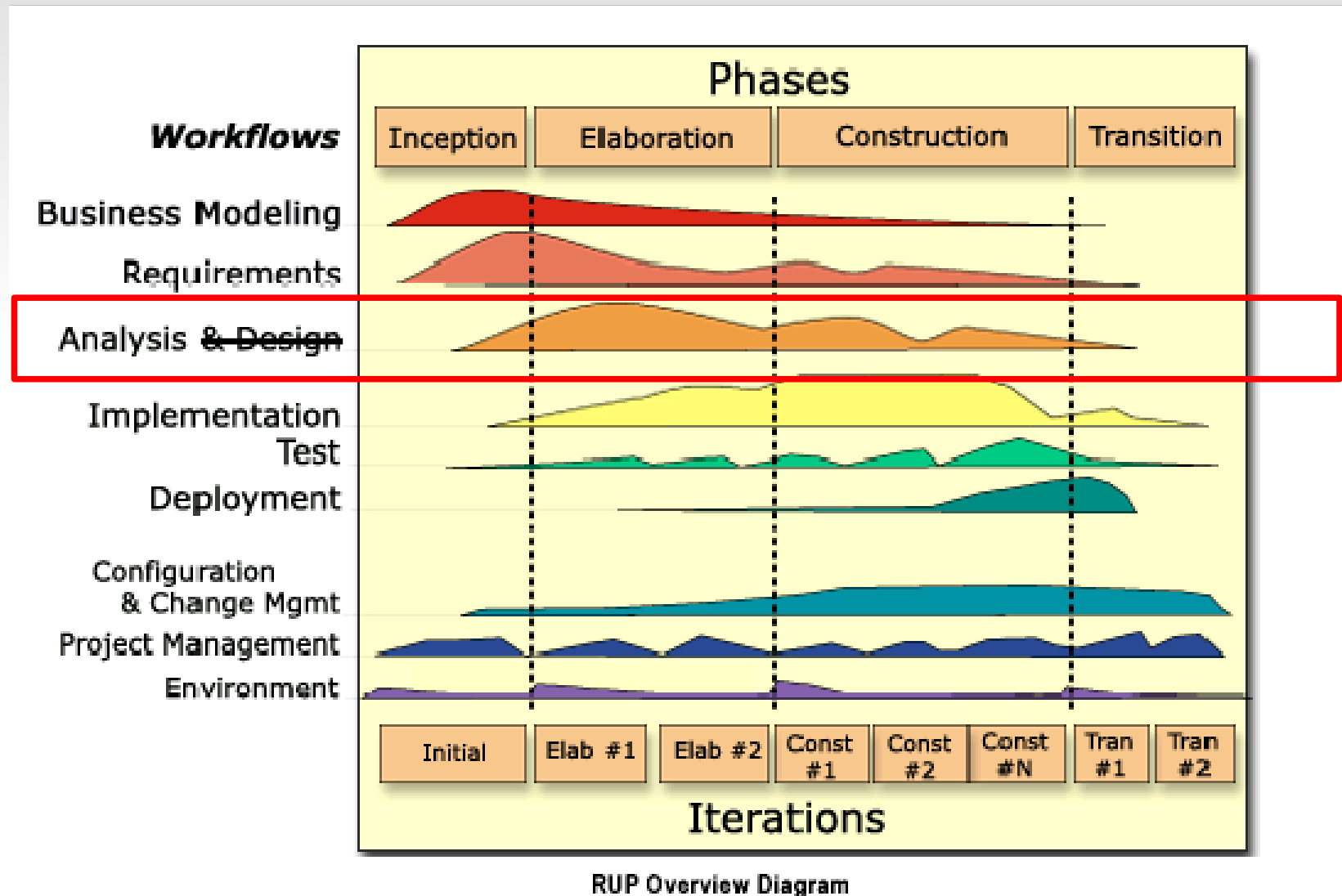

Analýza

analytické modely, objekty, třídy

© Radek Ošlejšek
Fakulta informatiky MU
oslejsek@fi.muni.cz

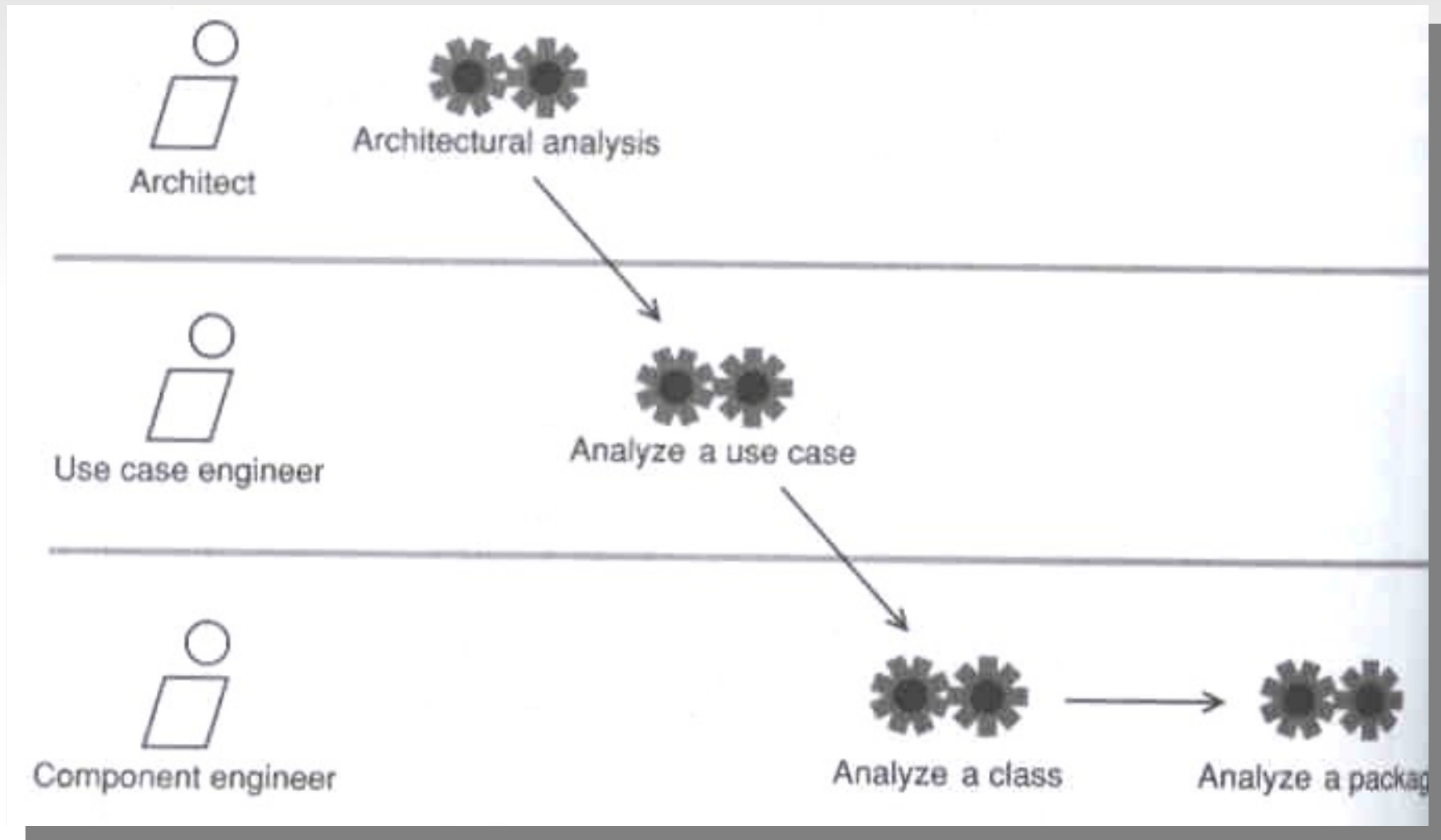
Analysis workflow

- Analytický model se zaměřuje na to, **co** bude systém dělat a ne **jak** to to bude dělat



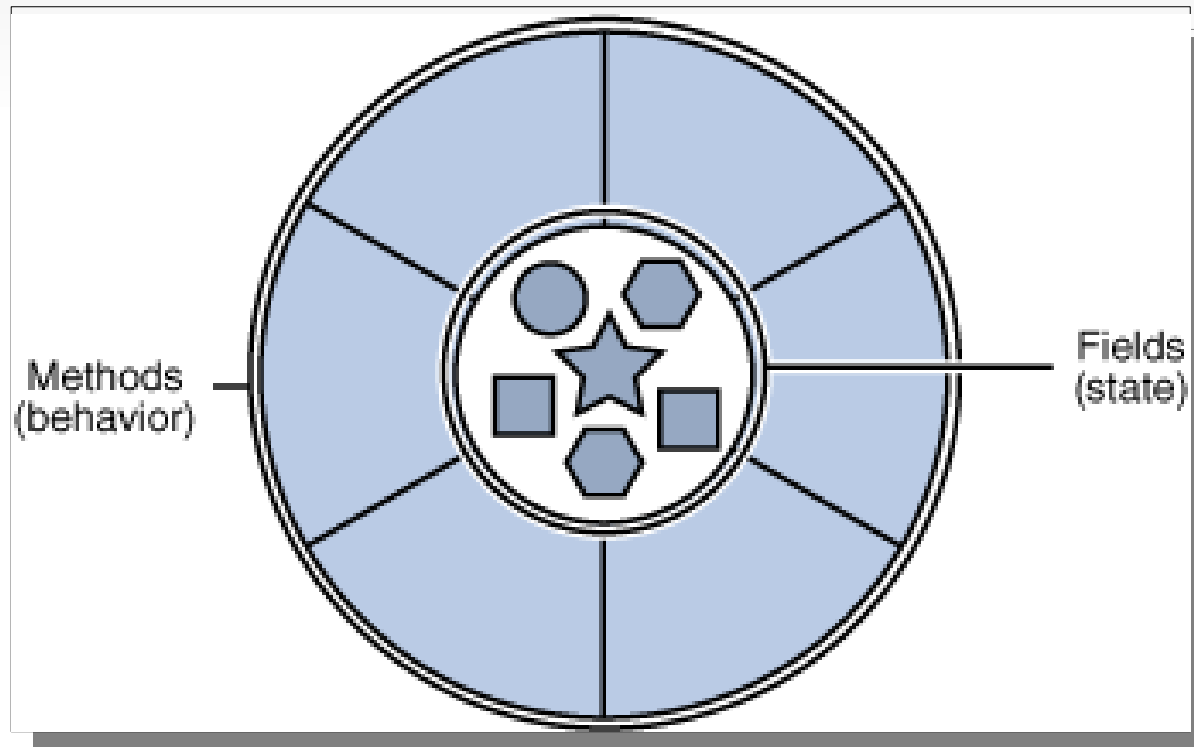
Analysis workflow detail

- Pro pochopení aktivit je třeba nejdříve pochopit objektové principy, proto se k aktivitám vrátíme později



Co jsou to objekty

- Objekty kombinují data a funkce do podoby uzavřené, soudržné jednotky
- Objekty ukrývají data za vrstvou funkcí (operací)
 - data jsou přístupná pouze skrze operace
 - zapouzdření



Co jsou to objekty

- Každý objekt má
 - Jednoznačnou identitu – odlišení objektu od ostatních (stejných) objektů.
 - Stav – hodnoty atributů uchovávají data objektu. Data objektu definují jeho stav.
 - Chování – operace, které je možné s objektem provádět. Operace často mění stav a často závisí na aktuálním stavu (na aktuálních datech).

Objekt: Stav

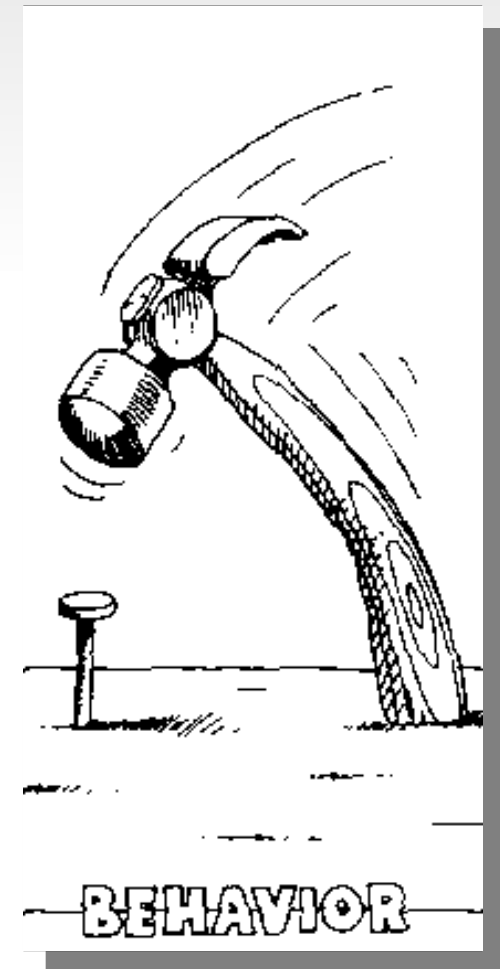
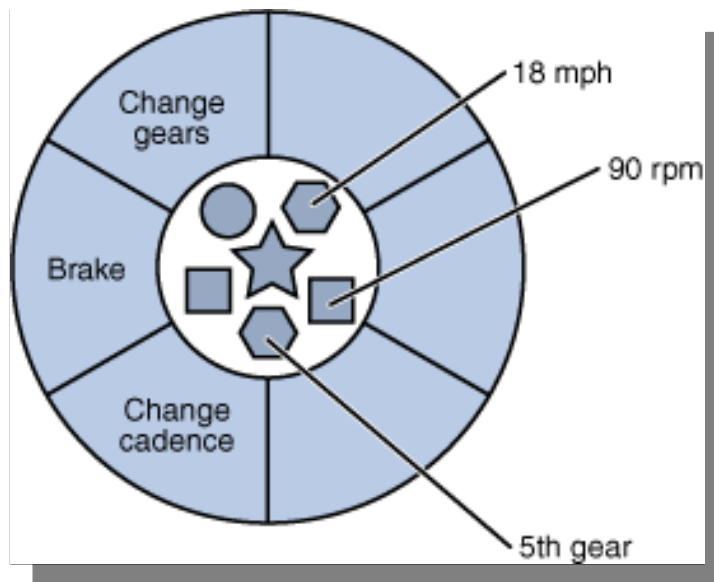
- Stav je určen ***hodnotami atributů*** a ***vazbami/vztahy na ostatní objekty*** v daném časovém okamžiku
- Př. pro tiskárnu:

Stav objektu	Název atributu	Hodnota atributu	Vazba
Zapnuto	power	on	N/A
Vypnuto	power	off	N/A
DošlaNáplň	inkCartridge	empty	N/A
Připojeno	N/A	N/A	Propojeno s objektem počítače
Odpojeno	N/A	N/A	Neprojeno s objektem počítače



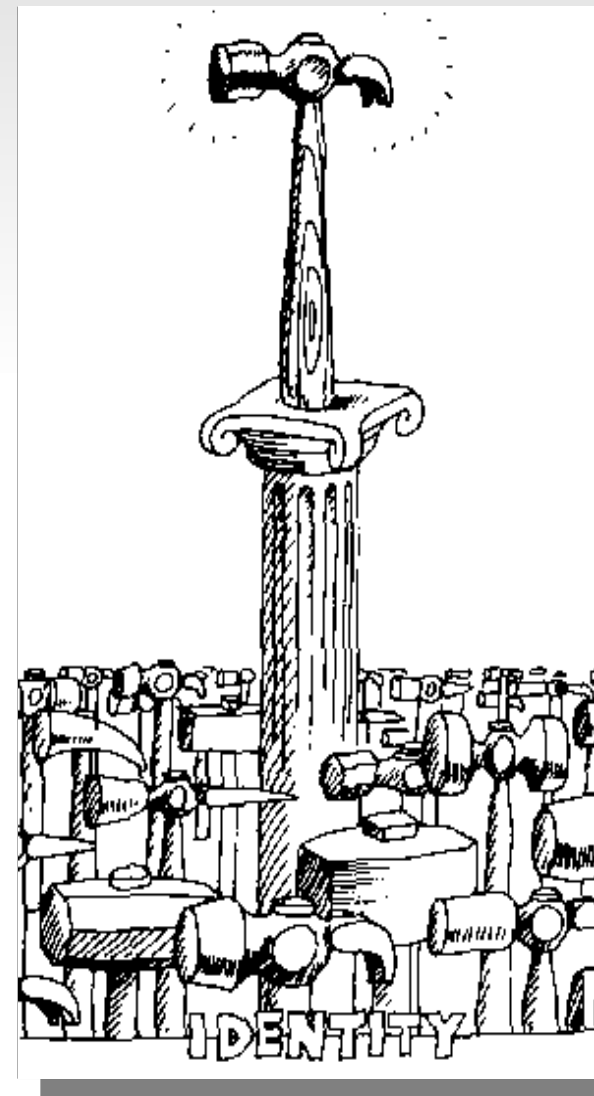
Objekt: Chování

- Chování vyjadřuje, jak objekt koná a reaguje
- Chování je specifikováno operacemi. Každá operace tedy definuje část chování objektu.
- Chování se může měnit v závislosti na stavu
 - Př: chování tiskárny při stavu „DošlaNáplň“
- Vyvolání operace může vést ke změně stavu
- Implementace operace se nazývá **metoda**



Objekt: Identita

- Každý objekt má unikátní výskyt v prostoru a čase
- Každý objekt je jednoznačně identifikovatelný
- Identita je určena
 - hodnotami atributů
 - adresou v paměti
 - ...

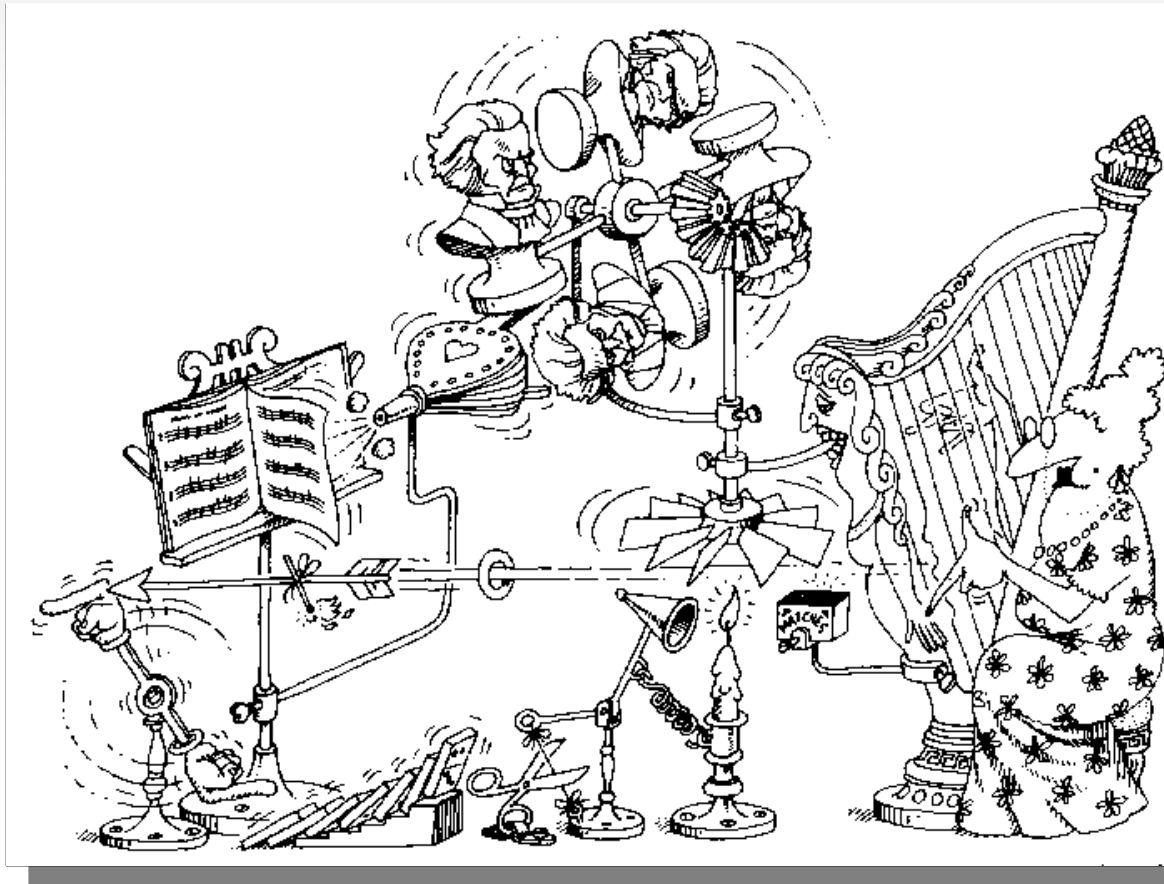


Identita – příklad

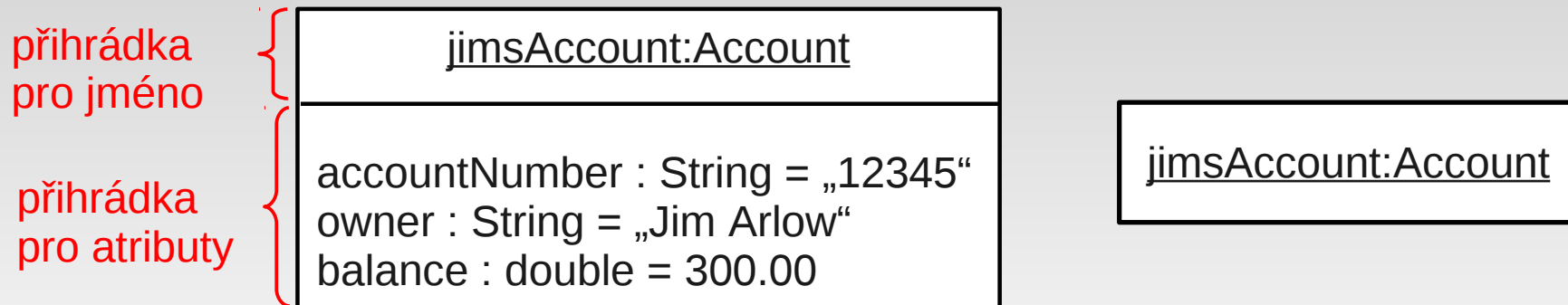
- Židle nabízené internetovým obchodem. Jak mohou být identifikovány?
- Adresa v paměti (odpovídá reálnému světu):
 - Dvě naprosto stejné reálné židle stojící vedle sebe jsou dva různé objekty (každá židle zabírá svůj prostor v daném čase)
 - Dvě naprosto stejné „virtuální“ židle jsou stejné, pokud jsou na různých adresách (zabírají každá svůj paměťový prostor v daném čase)
- Výrobní číslo (klonování v reálném světě?):
 - Dvě židle považujeme za jeden objekt, pokud mají stejné výrobní číslo, byť leží na různých adresách
- Na židli je možné posadit pouze jediného člověka. Jaký bude rozdíl v chování systému, pokud bude identita židle dána paměťovou adresou vs. výrobním číslem?

Spolupráce objektů, zasílání zpráv

- Objektový systém funguje na základě spolupráce jednotlivých objektů
- Objekty si posílají zprávy (vyvolávají operace na jiných objektech) a tím vytvářejí odezvu systému na požadavky uživatelů.
 - Celý běh objektového programu je dán stromem volání, kde kořenem je spustitelná třída



Objekty – notace UML



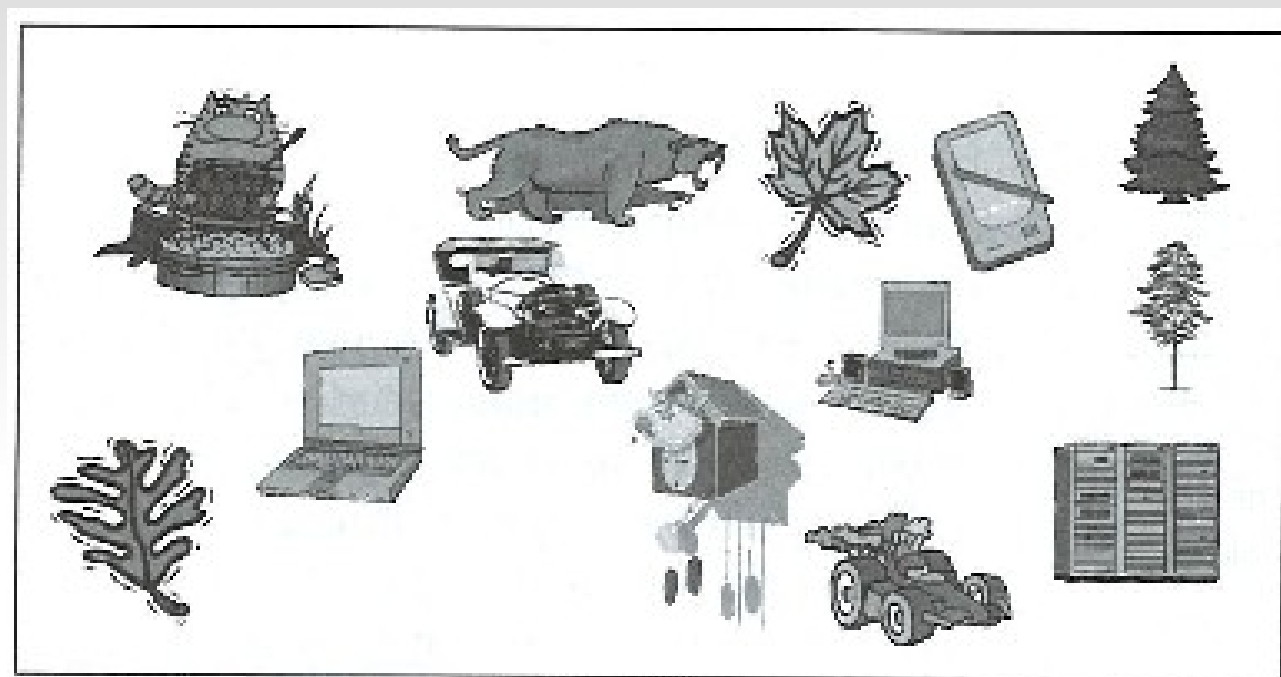
- *Syntaxe názvu: jméno objektu : jméno třídy [jména stavů]*
 - :Account – anonymní objekt dané třídy
 - jimsAccount – konkrétní objekt bez specifikované třídy, užitečné především v počátečních fázích analýzy
 - jimsAccount:Account – konkrétní instance třídy; nutné pokud je v diagramu více instancí jedné třídy
- Název je vždy podtržený, doporučuje se tzv. „*lowerCamelCase*“
- Formát pro atributy: **jméno : typ = hodnota**
 - typ se často vynechává, protože je uvedený u třídy
- Zobrazení atributů není povinné, záleží na účelu diagramu

Třída

- Každý objekt je instancí právě jedné třídy
- Třída popisuje vlastnosti množiny objektů
 - šablona objektu, která předepisuje všem instancím atributy, operace a vztahy k ostatním objektům/třídám
 - instance jedné třídy se ale mohou lišit stavem (aktuálními hodnotami atributů) a chováním předepsaných operací.

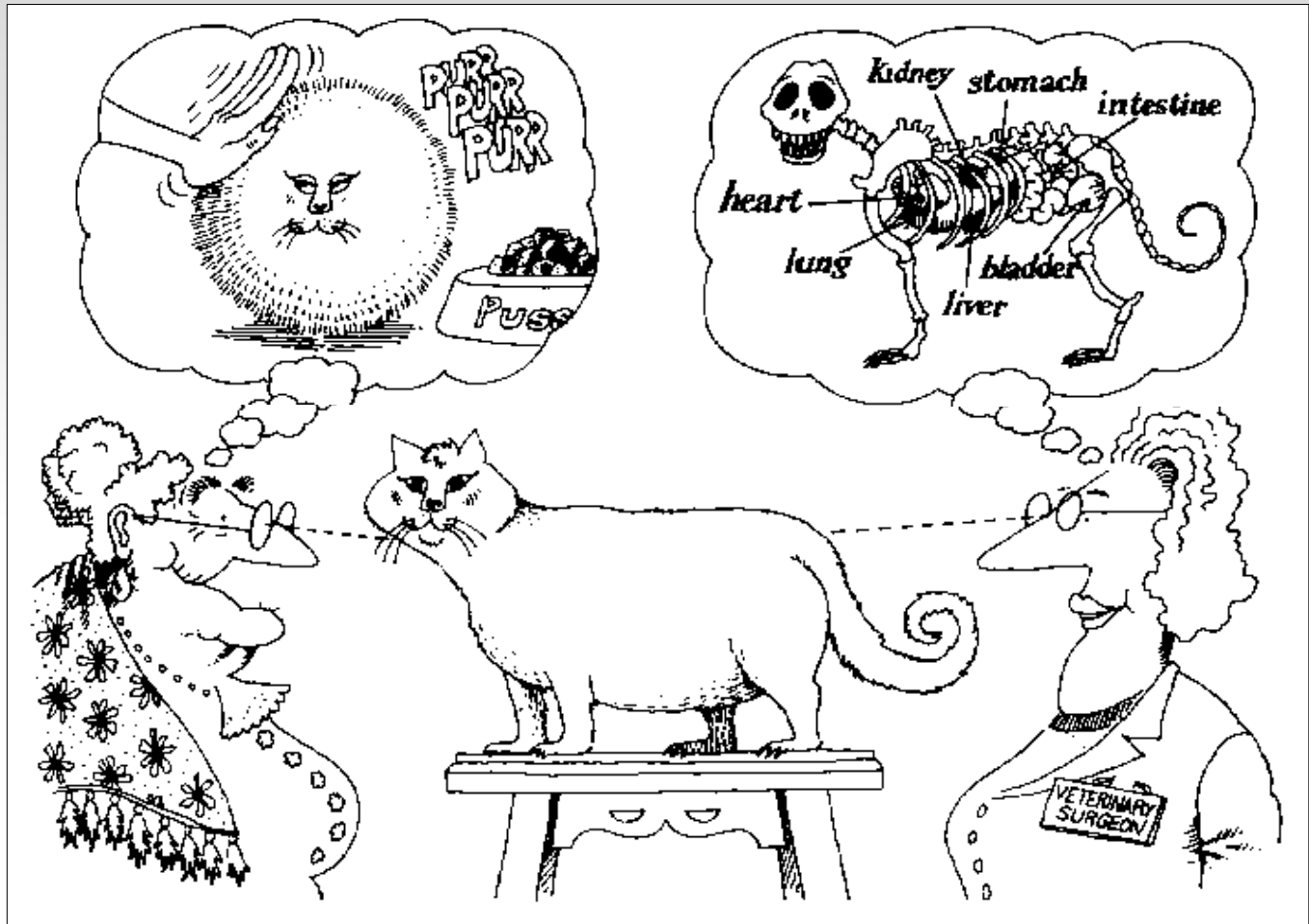
Abstrakce (klasifikace objektů a tříd)

- Kolik tříd je schováno na obrázku?



- třída stromů, třída listů, ...
- třída výpočetních zařízení vs. třída mobilních výpočetních zařízení
- třída koček vs. třída rychle se pohybujících objektů – kam patří kočkovitá šelma?
- atd., atd.

Abstrakce (klasifikace objektů a tříd)



- Nalezení vhodného klasifikačního schématu je jeden z klíčových úkolů objektově orientované analýzy

Třídy – notace UML

Window

potlačené detaily

Window

size: Area
visibility: Boolean

display ()
hide ()

detaily analytické úrovně
(typy a parametry mohou být také potlačeny)

<<boundary>>

Window

{abstract,
author=Joe,
status=tested}

+size:Area = (100,100)
#visibility:Boolean = invisible
+default-size:Rectangle
#maximum-size:Rectangle
-xptr: XWindow*
+name:String {frozen}
-colors: Color [3]

+*display ()*
+*hide ()*
+create ()
-attachXWindow(xwin:Xwindow*)

detaily implementační úrovně

info o
třídě,
název

oddíl
atributů

oddíl
operací
(metod)

Třídy – notace UML (II)

Jméno třídy:

tzv. UpperCamelCase, jednotné číslo, nezkracovat

Struktura atributů:

viditelnost název: typ [násobnost] = počátečníHodnota

nepovinné

povinné

nepovinné

Viditelnost:

+ public
- private
protected
~ package

Počáteční hodnota:

vyjadřuje důležité omezení
v analýze řídký jev

Násobnost:

[0..1] optional
[*] array, list

Typ:

Integer
UnlimitedNatural
Boolean
String
Real
Třída

celé číslo
celé číslo ≥ 0 , nekonečno = *
true nebo *false*
řetězec v uvozovkách
reálné číslo
odkaz na jinou třídu

<<boundary>>

Window

{abstract,
author=Joe,
status=tested}

+size:Area = (100,100)
#visibility:Boolean = invisible
+default-size:Rectangle
#maximum-size:Rectangle
-xptr: XWindow*
+name:String {frozen}
-colors: Color [3]

+display ()
+hide ()
+create ()
-attachXWindow(xwin:Xwindow*)

Třídy – notace UML (III)

Struktura operací:

viditelnost název (směr názevArg : typArg = počHodnota, ...): návratovýTyp

signatura operace

Směr parametrů:

in, *inout*, *out*

return - více návratových hodnot (Python)

implicitně *in*

Působnost atributů a operací (scope):

“instance scope” - nejčastější, v rámci objektu

“class scope” - společné pro všechny instance třídy (static)

<<boundary>>

Window

{abstract,
author=Joe,
status=tested}

+size:Area = (100,100)

#visibility:Boolean = invisible

+default-size:Rectangle

#maximum-size:Rectangle

-xptr: XWindow*

+name:String {frozen}

-colors: Color [3]

+display ()

+hide ()

+create ()

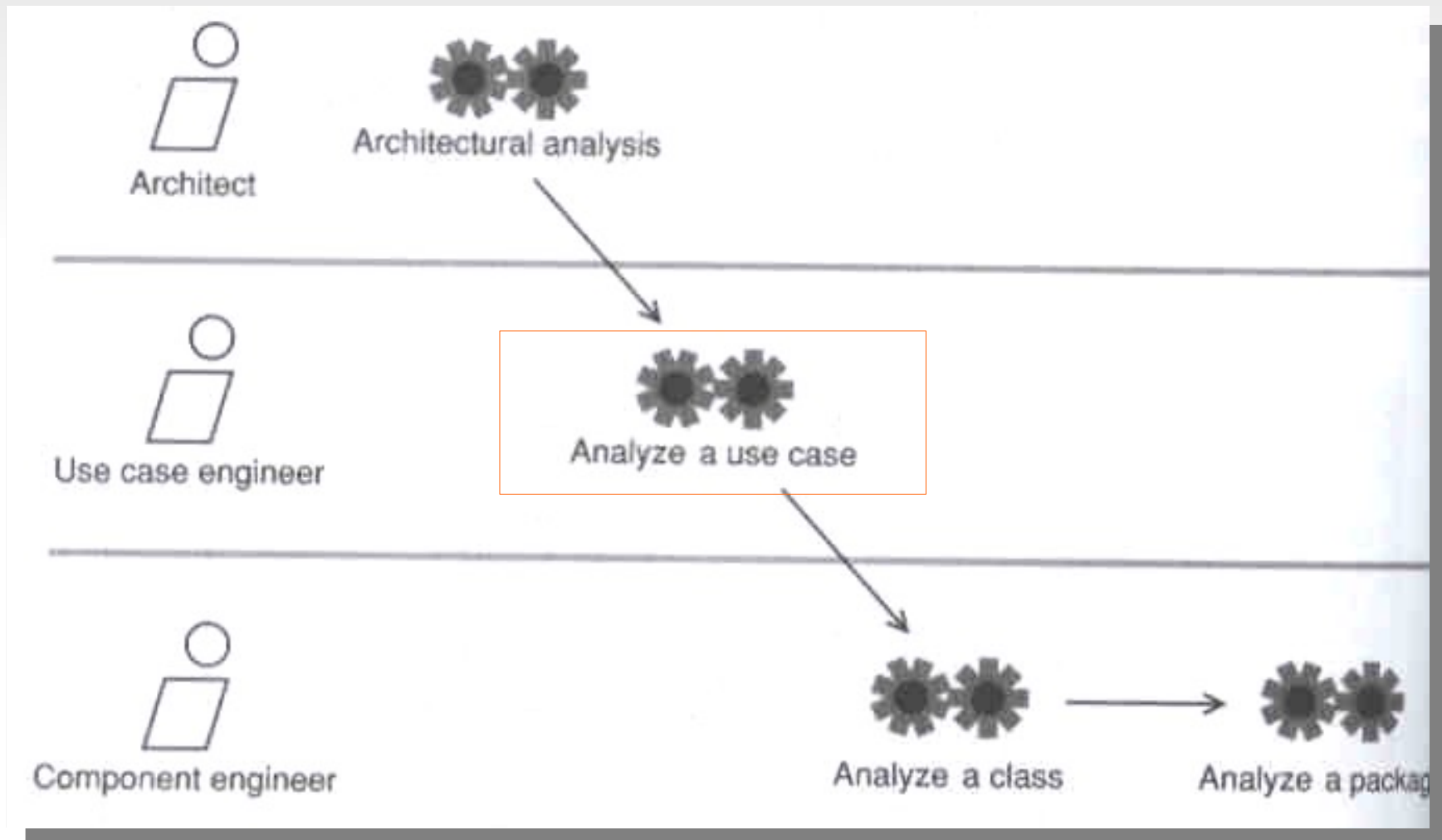
-attachXWindow(xwin:Xwindow*)

Rozsah platnosti atributů a operací

- *Angl. Scope*
- Instance scope
 - každý objekt (instance) má svoji hodnotu atributu
 - operace fungují nad konkrétním objektem a mohou záviset na jeho stavu
 - většina atributů a metod
- Class scope
 - hodnota atributu je společná všem instancím
 - změna hodnoty atributu jednou instancí se projeví i u všech ostatních instancí
 - operace nepracují s konkrétním objektem (nemají přístup k „instance scope“ atributům a operacím)
 - konstruktor, statické operace a metody v Javě a C++

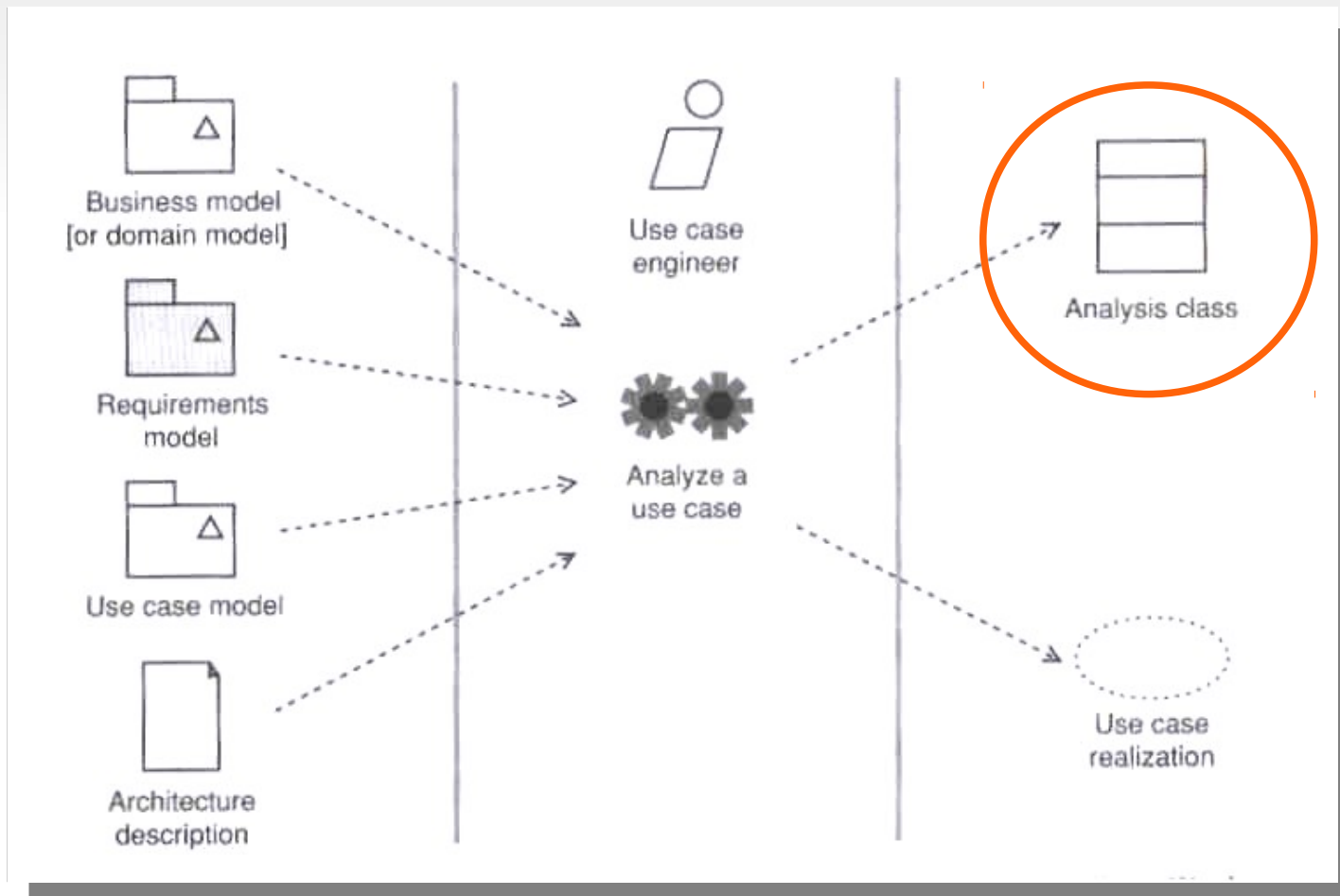
Analysis workflow detail

- Nyní se vraťme zpět k aktivitám RUP
- Architektura ovlivňuje výslednou strukturu a podobu systému. Proto se základní architektonická rozhodnutí dělají na začátku analýzy. My se ale k architektuřám dostaneme až později.



UP aktivita: Analýza případů užití

- *Architecture description* – důležité aspekty softwarové architektury v podobě UML modelů, popisného textu apod. Architekturami se budeme zabývat později
- *Analysis class* – jednoduchý diagram tříd se základními doménovými třídami a jejich vztahy



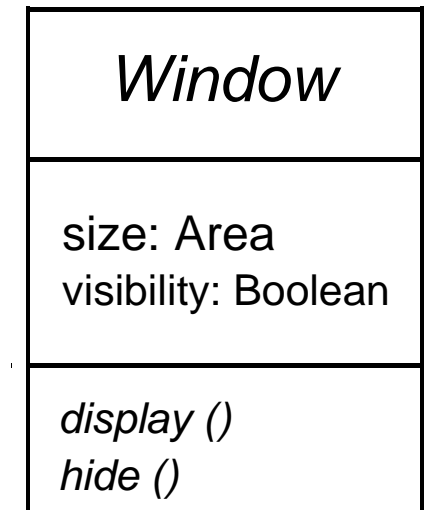
Nalezení analytických tříd

Pravidla pro vytváření analytického modelu

- Zobrazujte pouze třídy, které jsou součástí slovníku problémové oblasti.
- Vytvářejte modely, které „popisují příběh“. Každý diagram by měl objasňovat nějakou důležitou část požadovaného chování systému.
- Mějte nadhled, nezabývejte se detaily.
- Jasně rozlišujte mezi problémovou oblastí (problem domain – business requirements) a doménou řešení (solution domain – detailní návrhová rozhodnutí). Např. třídy *Customer* a *Order* jsou OK, ale třídy pro přístup k databázi ne.
- Minimalizujte propojení mezi třídami.
- Dědičnost jen pokud je jasná hierarchie abstrakcí.
- Stále se ptejte „*Je model užitečný pro všechny zúčastněné?*“. Není nic horšího, než když je model ignorován zadavateli, nebo návrháři a vývojáři.
- Udržujte model jednoduchý!

Analytické třídy

- Zachycují doménovou oblast, později jsou upřesněny/rozloženy na návrhové třídy
- Obsahují pouze klíčové atributy a operace nutné pro popis základní zodpovědnosti třídy
 - Jméno: je nezbytné
 - Atributy: pouze podmnožina budoucích atributů, typy se vynechávají
 - Operace: parametry a návratová hodnota jen pokud je to důležité
 - Viditelnost: nezobrazuje se



Vlastnosti dobré analytické třídy

- Pojmenování třídy odráží její význam
- Malá, jasně definovaná množina zodpovědností
 - zodpovědnost = typ služeb, kterou třída nabízí ostatním třídám
 - Př. zodpovědností pro třídu *NakupniKosik*:
 - OK: přidej položku do košíku, odeber položku z košíku, ukaž položky v košíku
 - ještě lépe (ve fázi analýzy): spravuj položky v košíku
 - špatně: ověř kreditní kartu, akceptuj platbu, vytiskni účet
- Malý počet vazeb na ostatní třídy
 - pro řešení své zodpovědnosti vyžaduje třída spolupráci pouze malého počtu dalších tříd
 - rovnoměrné rozložení zodpovědností mezi třídy vede k malému počtu propojení

Nalezení analytických tříd

- Analýza podstatných jmen a sloves
 - analýza textů (specifikace systému, model požadavků, dokumentace případů užití, apod.)
 - podstatná jména v textu často vedou na třídy nebo atributy
 - slovesa a slovesné fráze často indikují zodpovědnosti nebo operace
 - pozor na synonyma a homonyma
- CRC analýza
 - *Class, Responsibilities and Collaborators analysis*
 - “brainstorming” metoda s lepícími papírky
 - co papírek, to jedna třída
 - každý papírek obsahuje jméno, zodpovědnosti a spolupracující třídy
 - místo seznamu spolupracujících tříd se mohou nalepit na tabuli a propojit čárami
 - používá se spolu s analýzou podstatných jmen a sloves

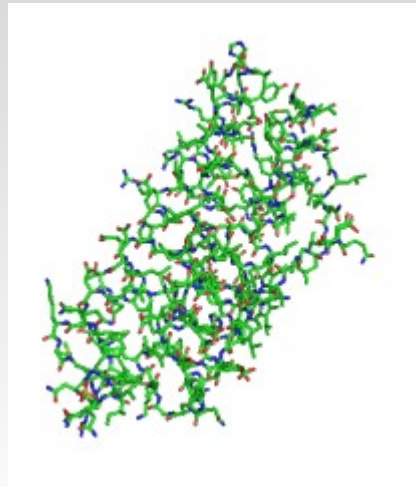
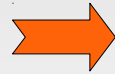
Nalezení analytických tříd (II)

- Další zdroje tříd
 - fyzické objekty (lidé, pracoviště, ...), dokumenty (formuláře, objednávky, ...), známá rozhraní (terminály, periferní zařízení), ...

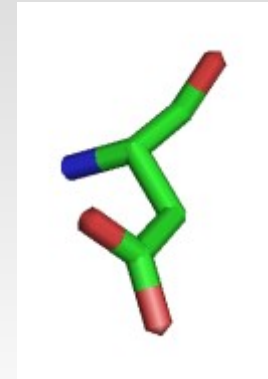
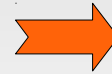
Případová studie: Molekuly



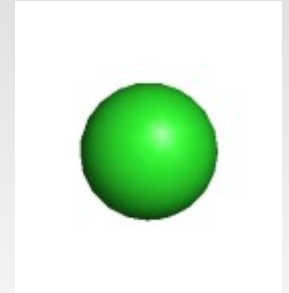
Structure



Molecule



Residue



Atom

Structure
-name

Molecule

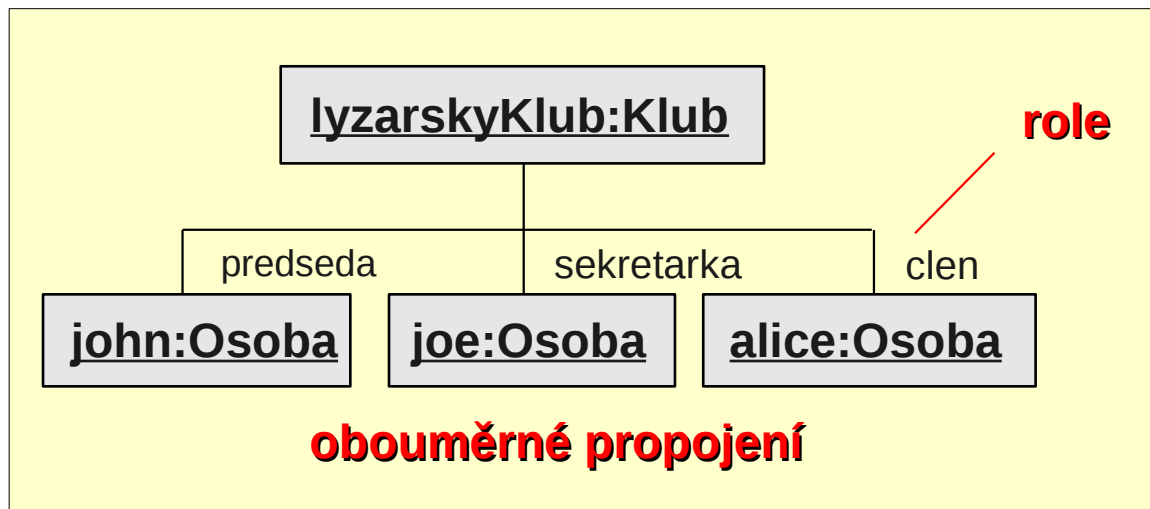
Residue

Atom
-atomicNumber

Modelování analytických tříd a objektů

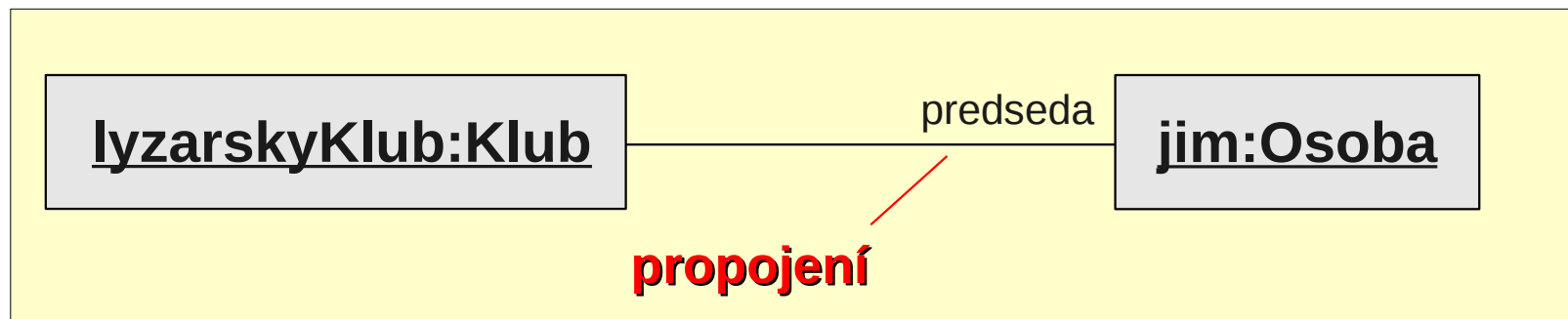
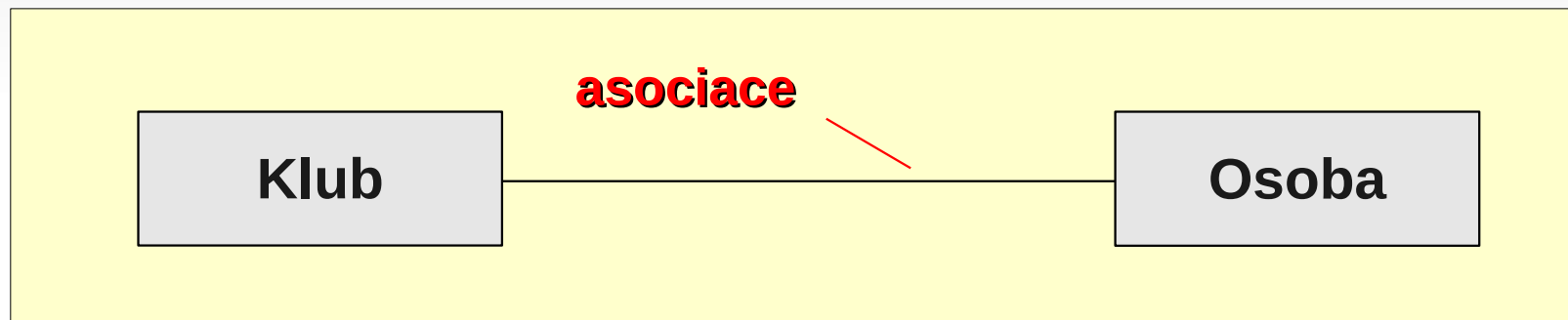
Diagram objektů

- Snímek v časovém bodě, který ukazuje podmnožinu objektů, jejich stavů, hodnot atributů a propojení.
- Vhodné zejména pro komunikaci se zákazníky (konkrétní příklad hierarchie, časový snímek systému, ...)
- Propojení (link)
 - „spárování“ objektů tak, aby spolu mohly komunikovat.
 - Role určuje význam objektu v propojení.
 - Šipka může upřesnit směr řízení – viz směr řízení u asociací



Propojení vs. asociace

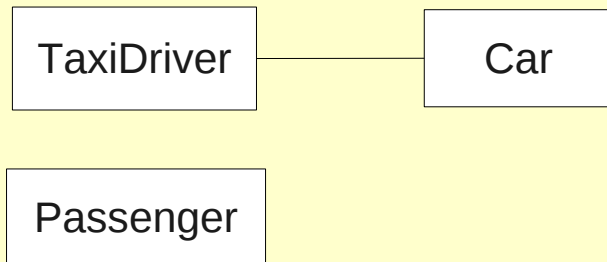
- Asociace je vztah mezi třídami (obdoba propojení u objektů)
- Asociace mezi třídami na diagramu tříd je nutnou podmínkou pro vytváření propojení na příslušném diagramu objektů



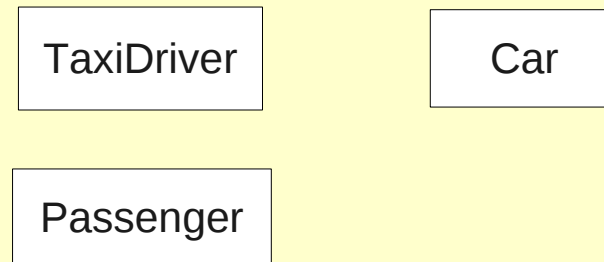
Vlastnosti propojení a asociací

- Propojení/asociace modelují dlouhodobé spárování objektů/tříd.
 - K tomu, aby dva objekty mohly komunikovat, není vždy nutné jejich dlouhodobé „spárování“. Pokud např. druhý objekt dostaneme jako parametr metody, jedná se o dočasné krátkodobé „spárování“, které se nemodeluje pomocí propojení/asociací!
 - PŘ. 1: Taxikář má přidělené konkrétní auto, dispečer dá pouze pokyn aby odvezl zákazníka
 - PŘ. 2: Dispečer řekne taxikáři: vezmi si toto auto a odvez zákazníka

```
public class TaxiDriver {  
    private Car car;  
    public void go(Passenger p) {}  
}
```

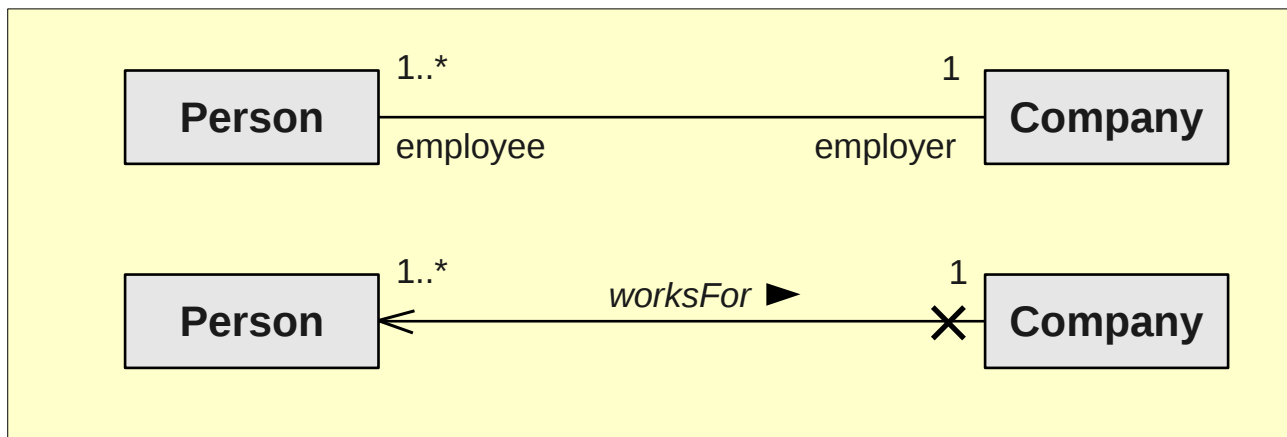


```
public class TaxiDriver {  
    public void go(Car c, Passenger p) {}  
}
```



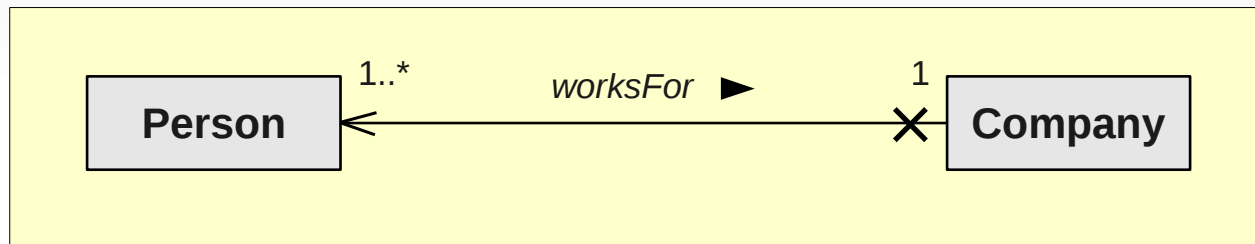
Asociace mezi třídami

- Zobrazují se jako plná čára mezi dvěma třídami
- Měly by být pojmenovány, šipka u jména indikuje směr čtení
- Role určuje způsob, jakým se na asociaci podílí
 - pojmenování role není povinné
 - alternativa k pojmenování asociace
- Násobnost definuje počet objektů ve vztahu (je definována **na úrovni instancí**)
- Jsou obousměrné (výhodné pro analýzu) pokud neupřesníme směr (většinou při návrhu)
- Modelujeme pouze dlouhodobé asociace



Směr asociací (směr řízení)

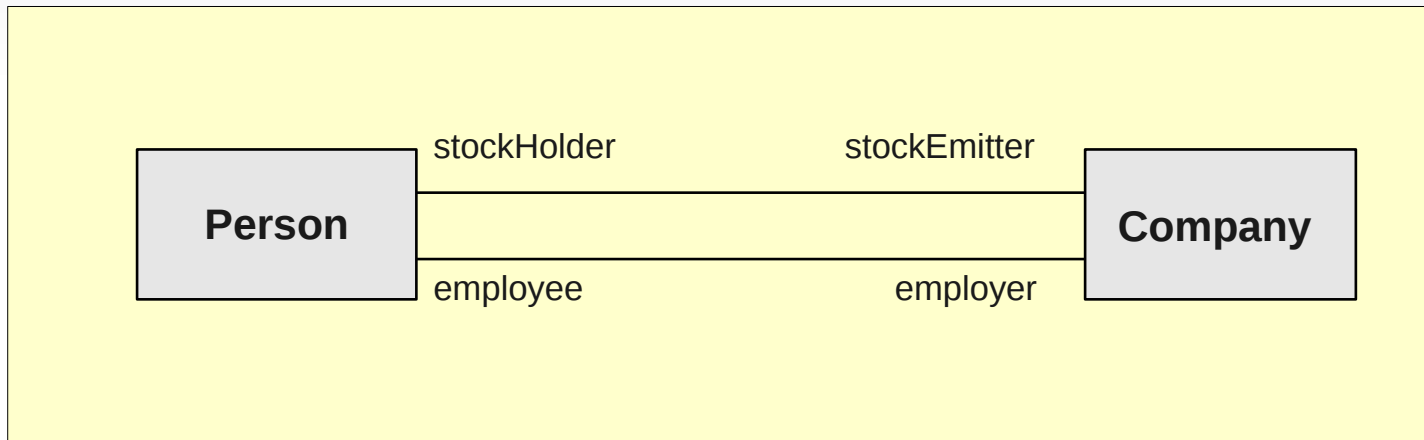
- I když je asociace jednosměrná, většinou je možné ji “procházet i v zakázaném směru” za cenu vyšší výpočetní náročnosti
- Př: Osoba neví nic o firmách, ve kterých je zaměstnána. Přesto můžeme firmy zaměstnávající osobu najít tak, že procházíme všechny firmy a hledáme ty, které zaměstnávají danou osobu.



- Ekvivalent jednosměrných ulic: Snadno se dostaneme z jedno konce na druhý. Pokud se chceme dostat zpět, musíme nějak objet celý blok.

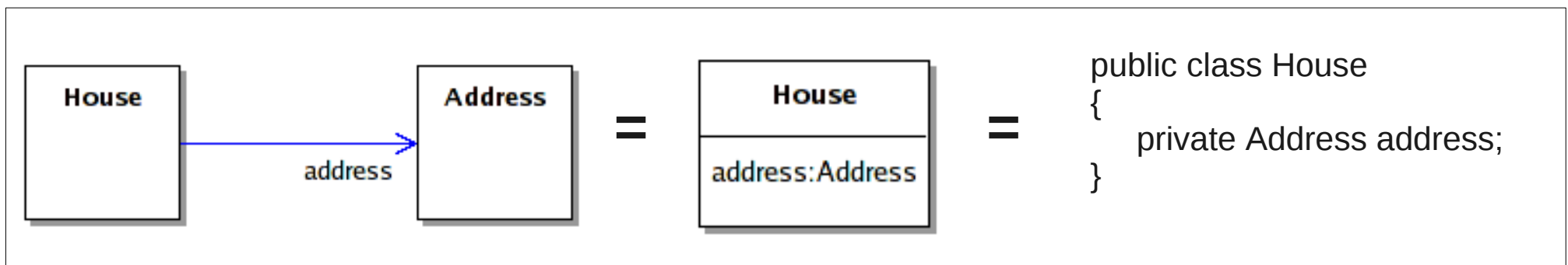
Násobné asociace

- Mezi instancemi mohou vznikat různé typy statických vazeb
 - Př: „zaměstnanecký vztah“ vs. „akcionářský vztah“ mezi osobou a firmou



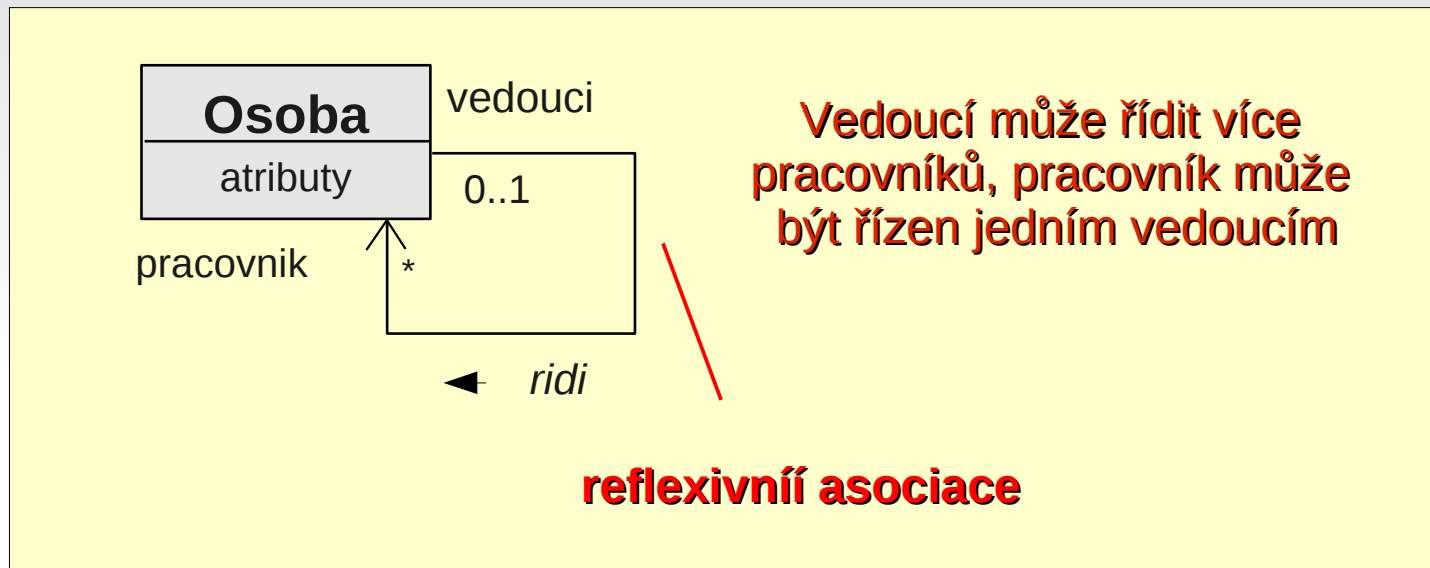
Asociace a atributy

- Mezi asociacemi tříd a atributy tříd existuje velmi těsná vazba.
- Vazba 1:1 odpovídá odkazu atributem
- Vazba 1:N odpovídá odkazu polem atributů nebo jedné kolekci
- Vazba M:N se řeší v návrhu
- Role v asociaci se často stává základem pojmenování atributu
- Směr asociace udává, která třída obsahuje atribut
- Obousměrné asociace se upřesňují při návrhu



Reflexivní asociace

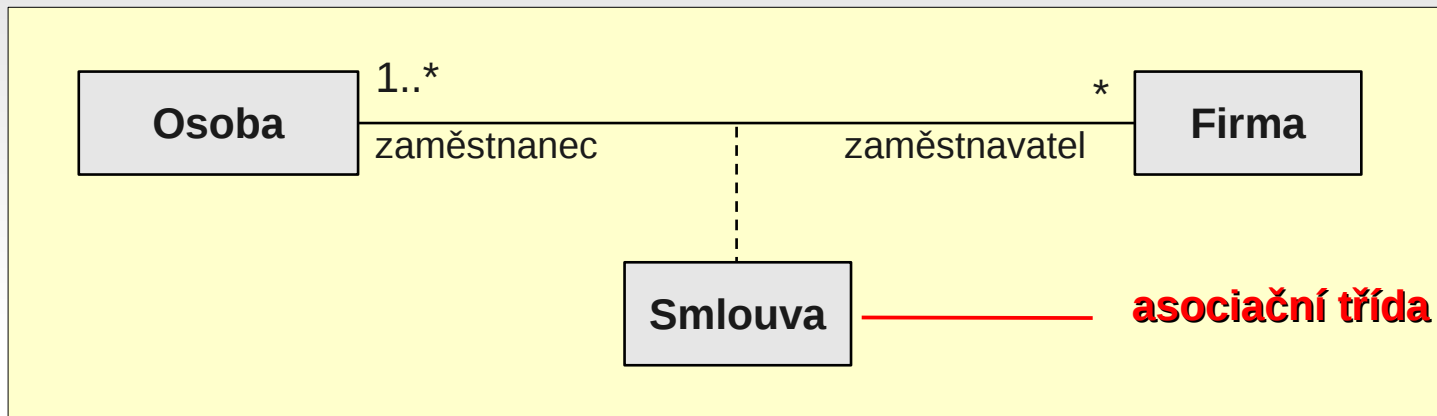
- Angl.: *Reflexive association*
- Měly by mít vždy role, jinak vzniká chaos



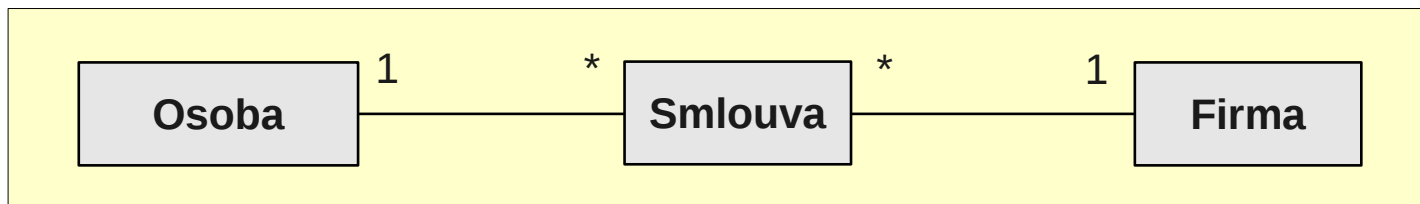
- Další “ozdoby” asociací:
 - » uspořádání: {ordered}
 - » měnitelnost: {addOnly}
 - » viditelnost: +, #, -
 - » navigace: směrové šipky
 - » omezení: {union}, {subset}, {unique}

Asociační třídy

- Angl: *Association class*
- Asociační třída vzniká jako produkt vztahu
- Nalezení: *Kam patří plat?*

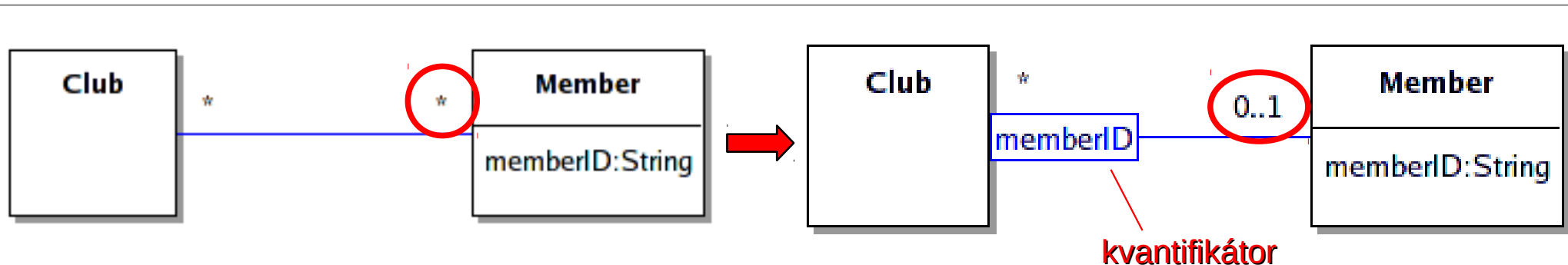


- Jedna osoba může mít s jednou firmou pouze jedinou smlouvu
 - může existovat pouze jediné propojení mezi objekty třídy Osoba a Firma v daném čase
- Pokud může být smluv více, tak:



Asociace s kvantifikátorem

- Angl: *Quantified association*
- Redukce asociace typu M:N na typ N:1
 - slouží k výběru jednoho člena cílové množiny
 - představuje “vyhledávací klíč”
 - kvantifikátor se obvykle odkazuje na atribut cílové třídy
 - kvalifikátor je součástí asociace, nikoliv třídy!

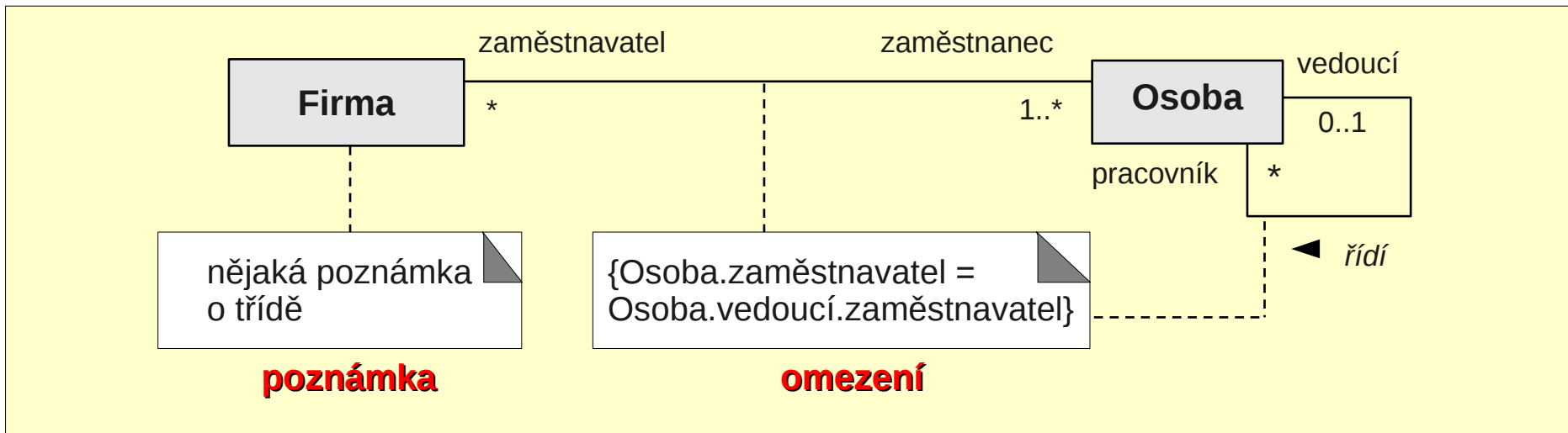
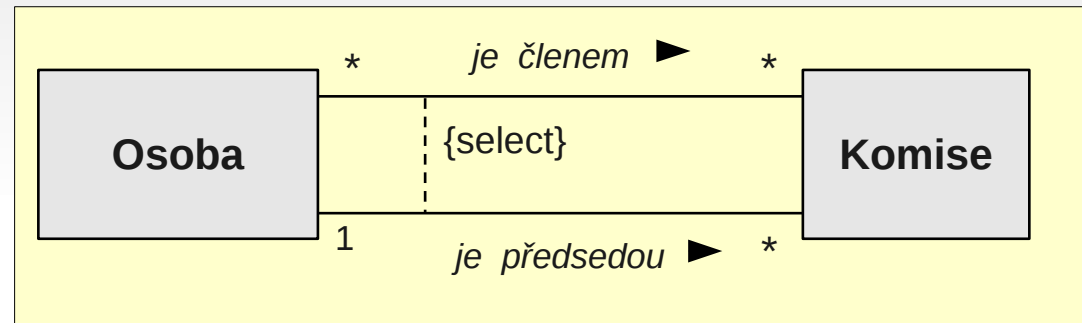
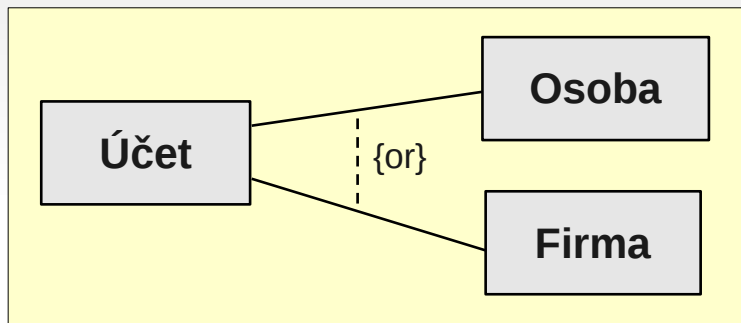


Máme např. objekt typu *Club* spojený s množinou objektů typu *Member*. Jak řídit jednu konkrétní instanci třídy *Member*?

Kombinace {*Club*, *memberID*} specifikuje jedinečný cíl

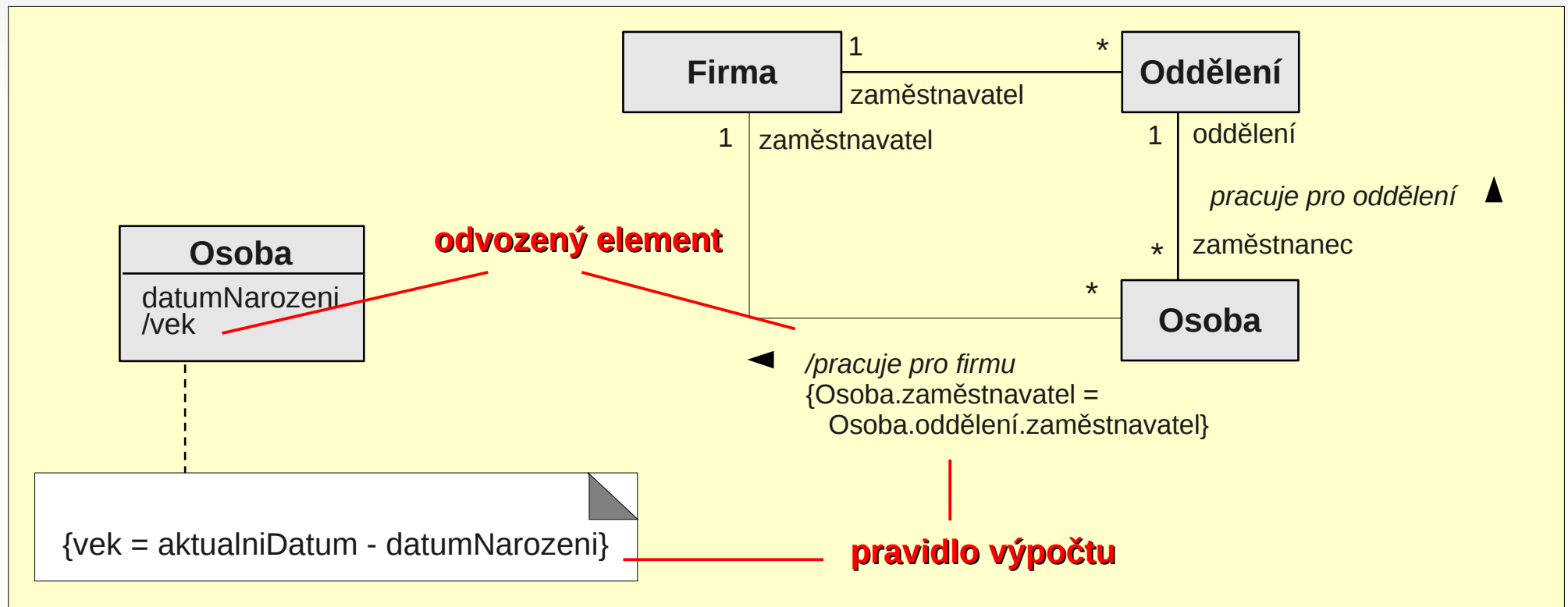
Omezující podmínky na asociacích

- Omezení specifikuje podmínky a tvrzení, které musí být udržovány jako pravdivé
 - umístěno bezprostředně za elementem, na který je aplikováno (např. atribut), připojeno čárkovanou čarou, nebo v poznámce
 - vždy v {}



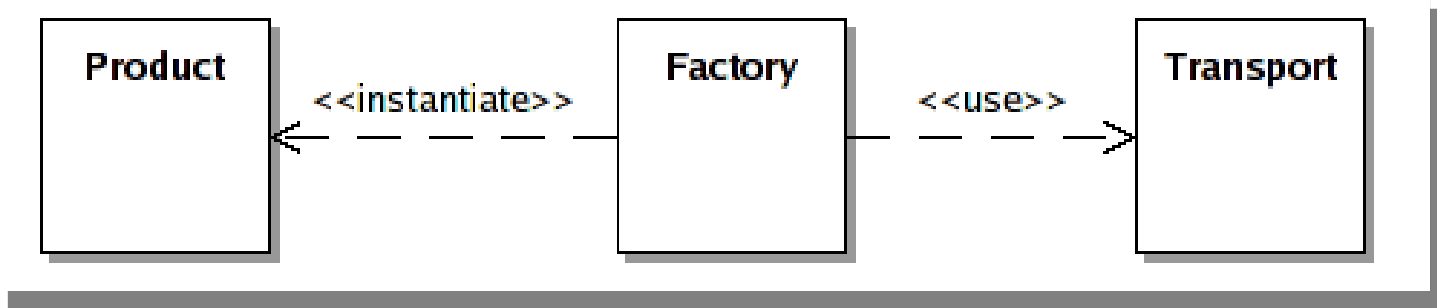
Odvozené atributy a asociace

- Odvozený element (angl. *derived*) je takový, který může být odvozen z jiného elementu
 - je ukázán pro názornost (analytická úroveň)
 - je zahrnut z implementačních (např. efektivita) důvodů (úroveň návrhu)
 - nepřidává žádnou sémantickou informaci**



Vztah závislosti

- **Vztah závislosti** (angl. *dependency*) je nejobecnější vazba. Závislost mezi dvěma elementy (např. třídami) indikuje, že změna v cílovém elementu (např. rušení) může vyžadovat změnu ve zdrojovém elementu, ale ne naopak.
- Př. 1: Objekt jedné třídy je předán jako parametr metodě jiné třídy. Nejedná se o asociaci, přesto uvnitř metody komunikujeme s cizím objektem a tedy závisíme na jeho rozhraní.
- Př. 2: Instanciace třídy (vytvoření objektu v paměti konstruktorem) je vztah závislosti mezi třídou a jejími objekty.

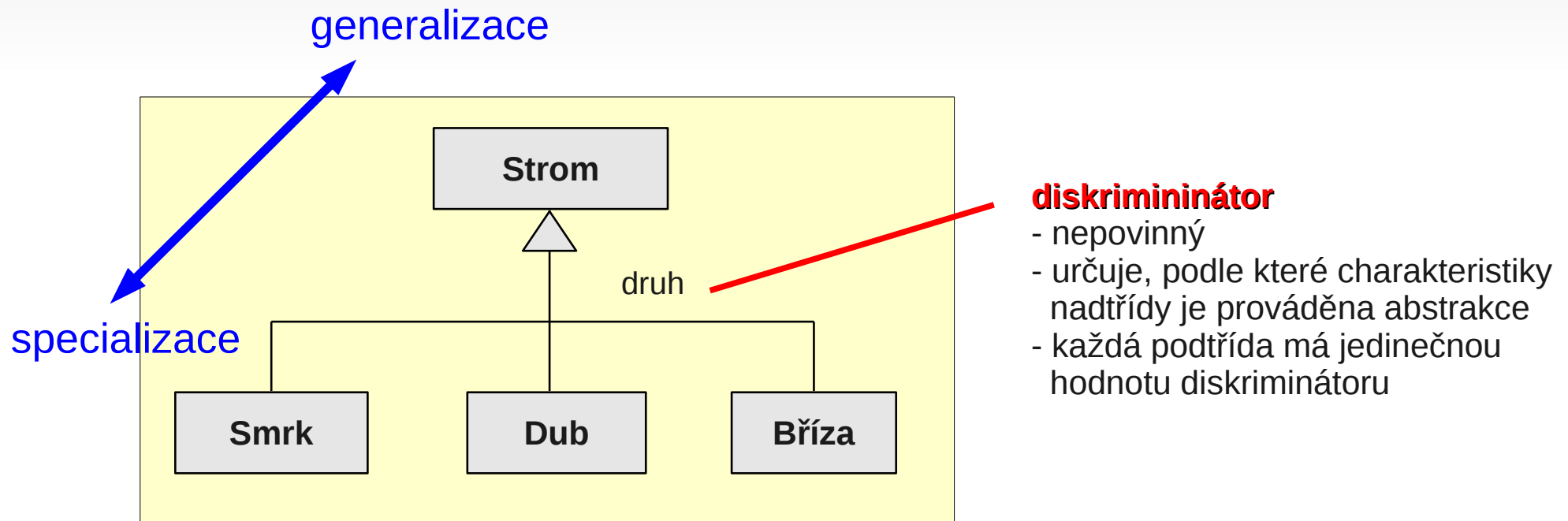


Některé předdefinované stereotypy:

- Zdrojový element (Factory) nepracuje bez cílového elementu (Transport): <<use>>
- Historicky odlišné elementy (nový = zdroj, starý = cíl) na různých úrovních abstrakce, ale reprezentující shodný koncept: <<trace>>
- Zdrojový element vytváří instance cílového elementu: <<instantiate>>
- Přátelský vztah ala C++: <<permit>>

Generalizace / specializace

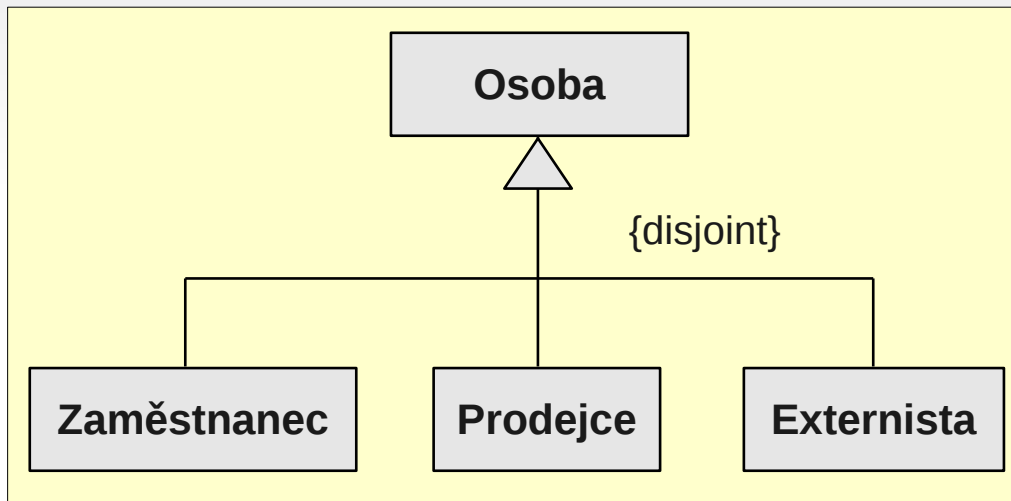
- Generalizace je vztah mezi obecným a specifitějším elementem (třídy, případy užití, rozhraní), který:
 - je **plně konzistentní** s obecným elementem
 - a přidává **dodatečnou informaci**



Generalizace - ozdůbky

■ Ozdůbky:

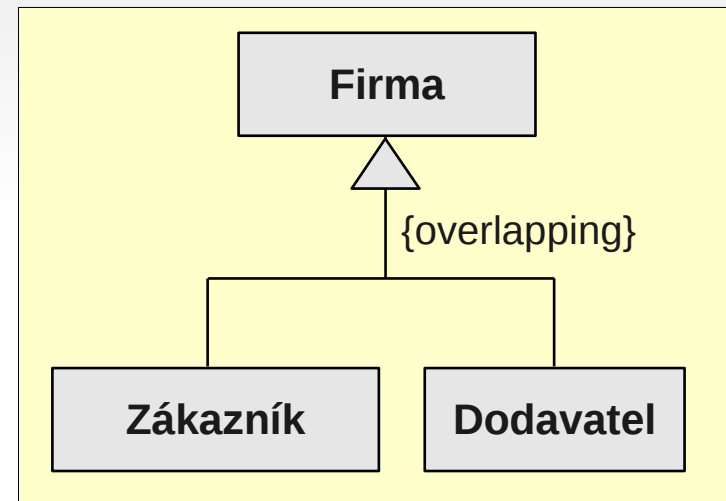
- » {overlapping} nebo {disjoint}
- » {incomplete} nebo {complete} – možnost rozšiřovat hierarchii o nové třídy
- » implicitně je {disjoint} {incomplete}



výlučná specializace

Osoba je:

- buď zaměstnanec, nebo prodejce, nebo externista.

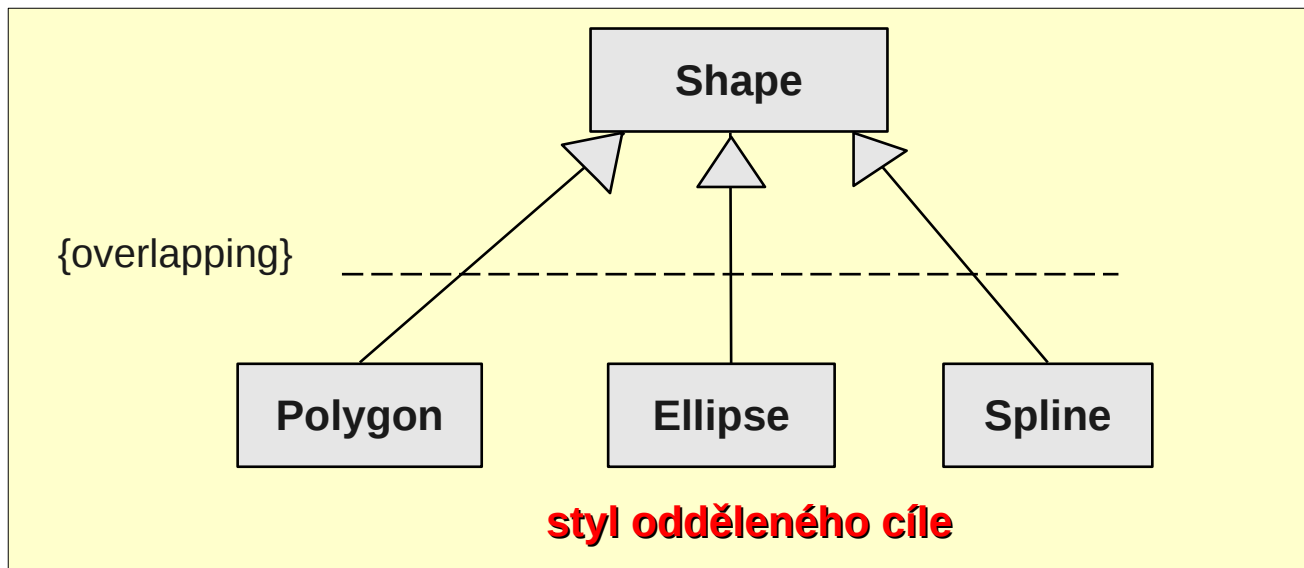
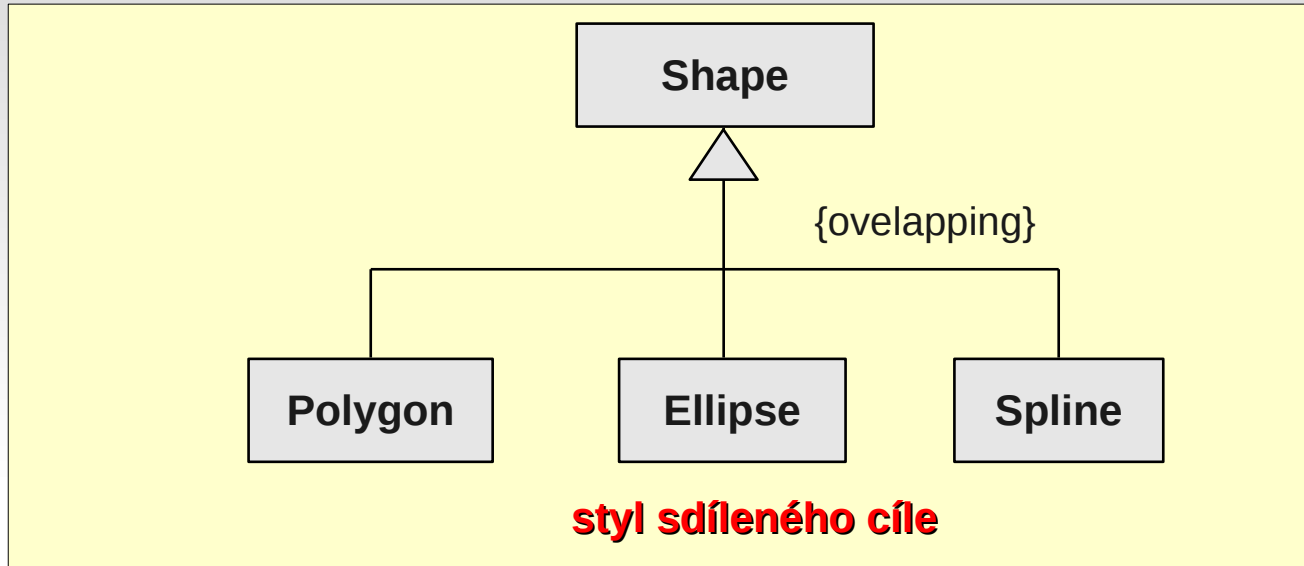


nevýlučná specializace

Firma je:

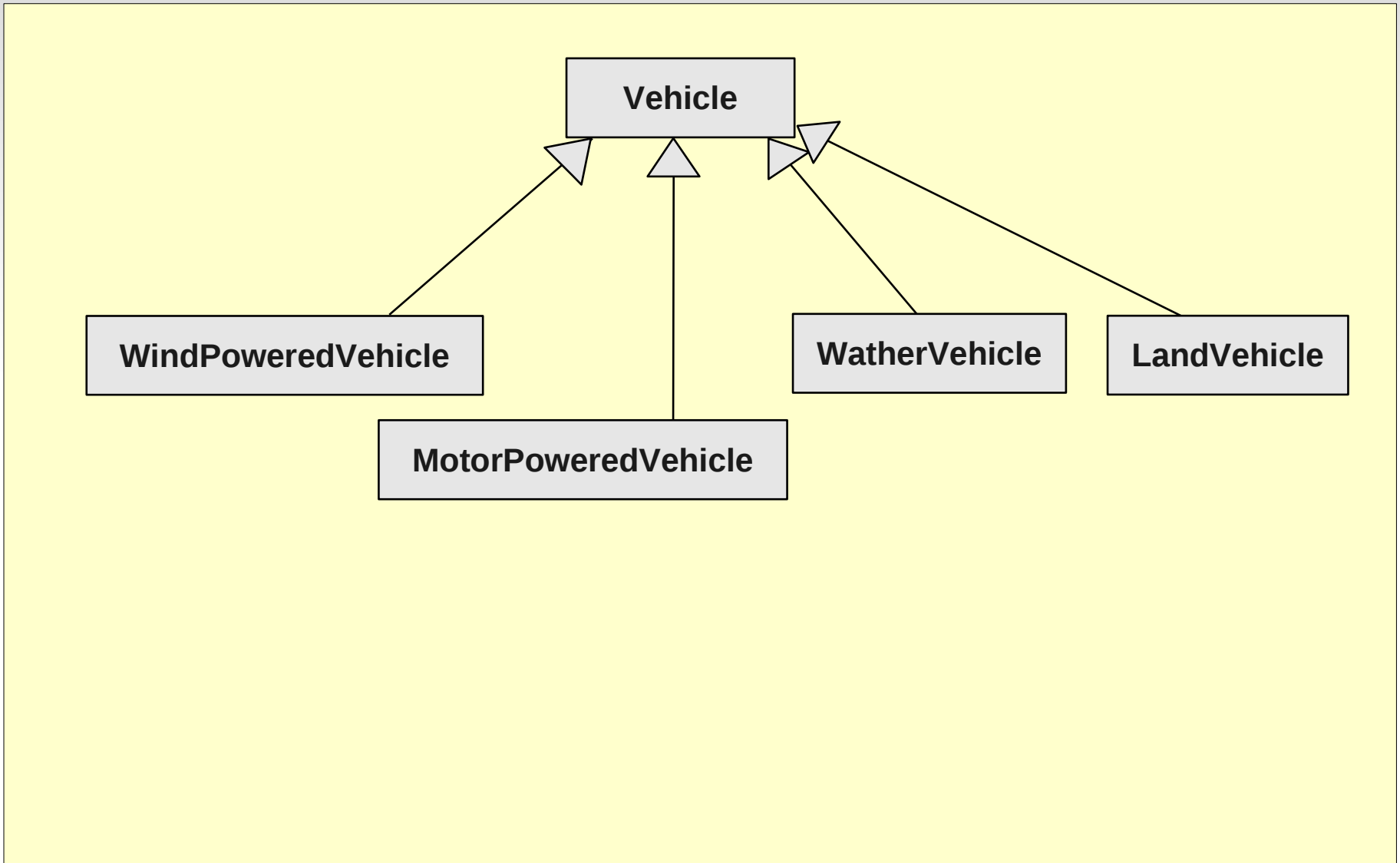
- zákazníkem, nebo dodavatelem, nebo obojím.

Generalizace – styl zápisu

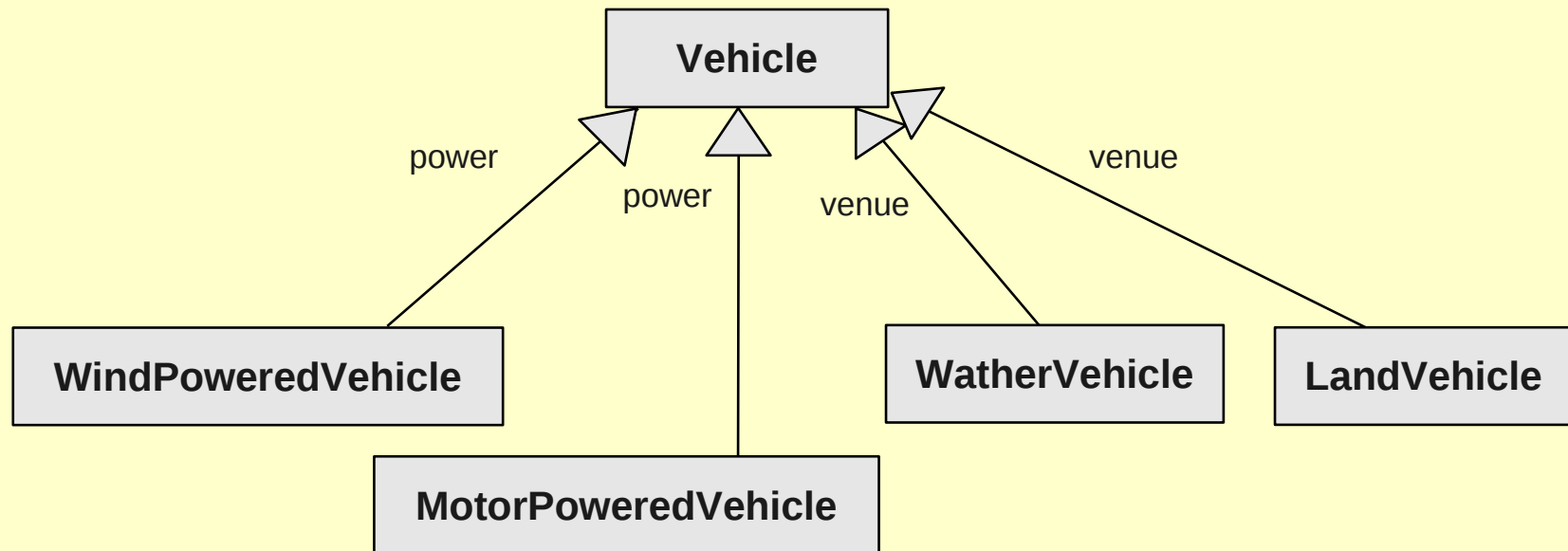


Příklad „zneužití“ dědičnosti (I)

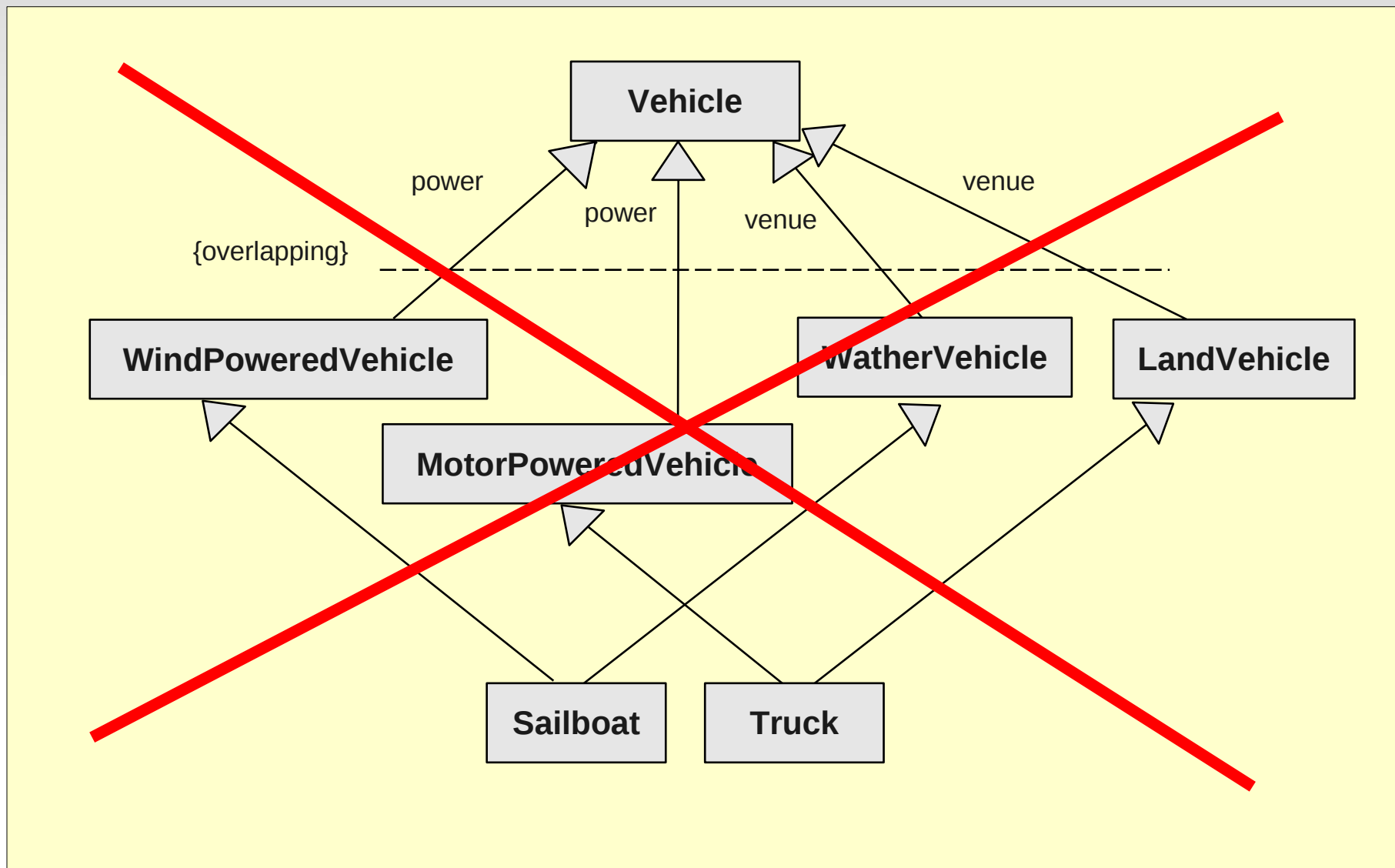
- Příklad chybné volby abstrakce



Příklad „zneužití“ dědičnosti (II)



Příklad „zneužití“ dědičnosti (III)



Vlastnosti dědičnosti a asociace

- Dědičnost

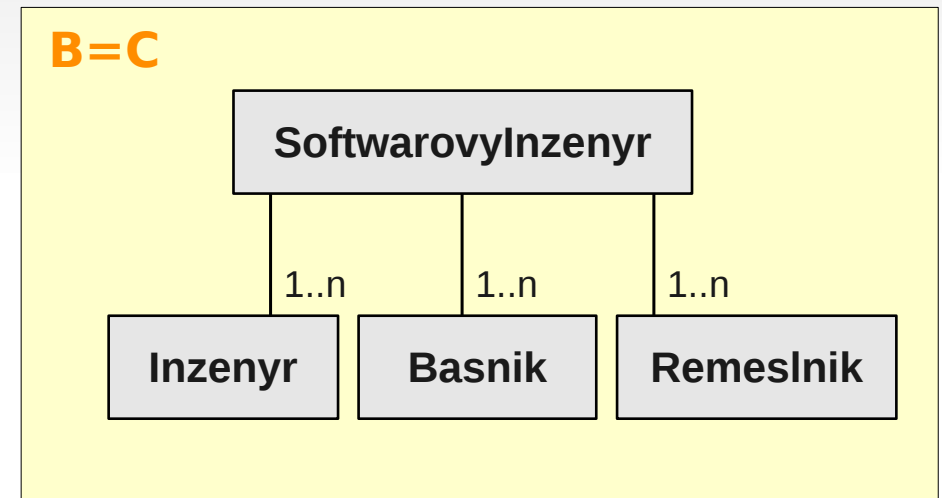
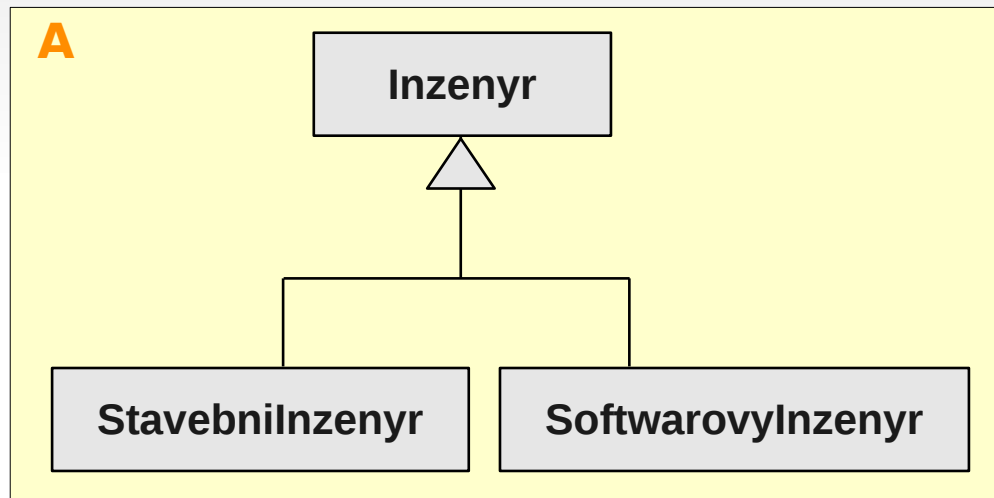
- je velmi snadné přidat novou třídu do modelu
 - neovlivní se tím zbytek modelu
- může zjednodušit model
 - atributy, metody a asociace jsou děděny z nadtříd

- Asociace

- změny na diagramu tříd mohou být obtížné
 - např. přidání/zrušení asociace, změna kardinality, ...
 - význam a použití asociace je zakódován v metodách třídy, změny tedy vedou ke změnám kódu
- jsou velmi dynamické na úrovni objektů (instancí)
 - za běhu mohou vznikat a zanikat konkrétní propojení (v rámci omezení daných definicí asociace na diagramu tříd)

„Would you rather buy or inherit ?“

- A. Každý softwarový inženýr je inženýr.
- B. V každém softwarovém inženýrovi je inženýr.
- C. Každý softwarový inženýr má inženýrskou komponentu.



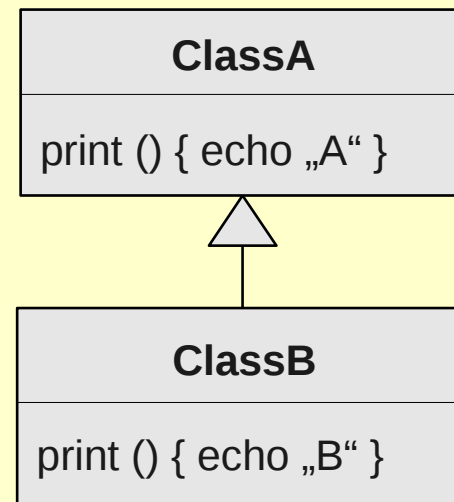
Nepoužívejte dědičnost pro popis vnímaného vztahu „je něčím“, pokud odpovídající objektové komponenty se **mohou změnit za běhu**.

Polymorfismus

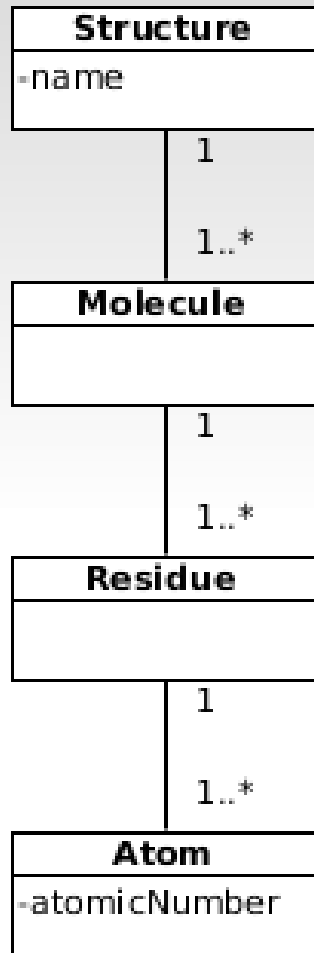
- Koncept z teorie typů, kdy jedno jméno může označovat různé věci:
 - „+“ znamená „stejnou věc“ pro real a integer
 - „+“ je implementováno odlišně pro real a integer
- Polymorfismus je důsledkem interakce mezi dědičností a dynamickou vazbou
 - (pod)třída dědí jméno operace
 - vazba implementované metody na toto jméno nastává až při provádění

Co vypíše následující program?

```
ClassA object;  
object = new ClassB();  
object.print();
```



Případová studie: Základní doménový model



Doménový model

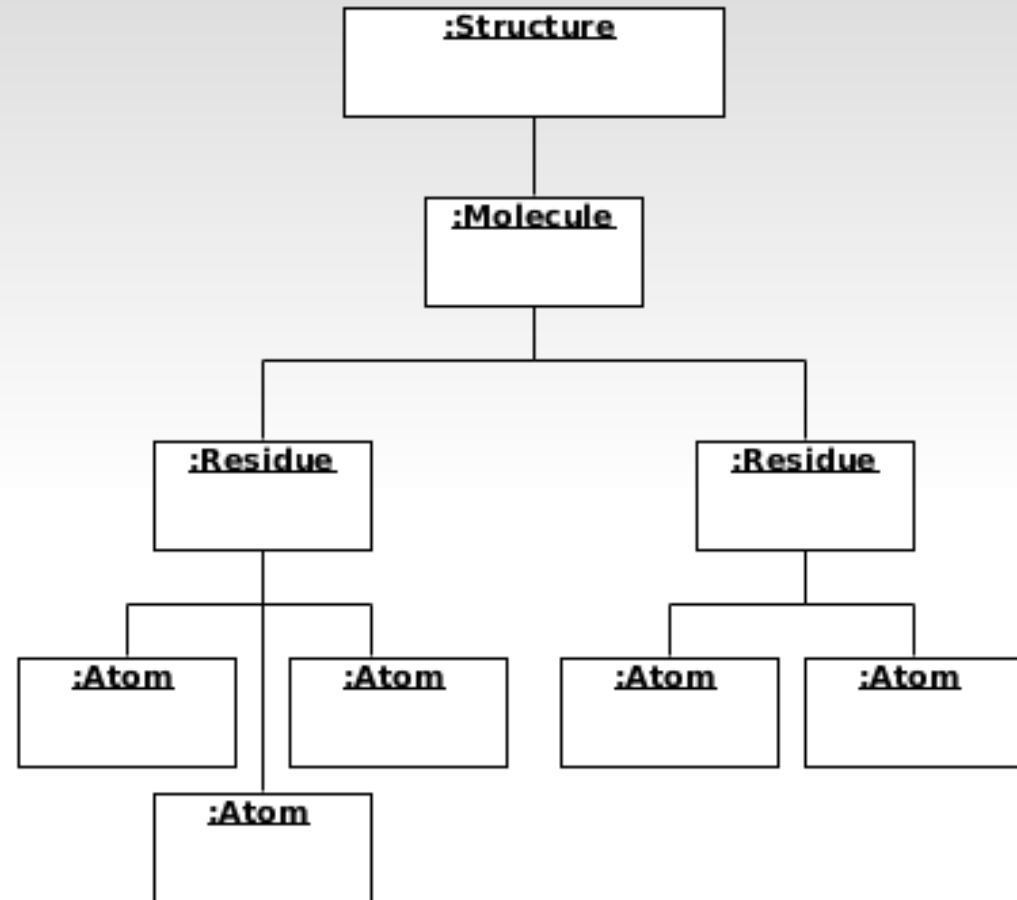
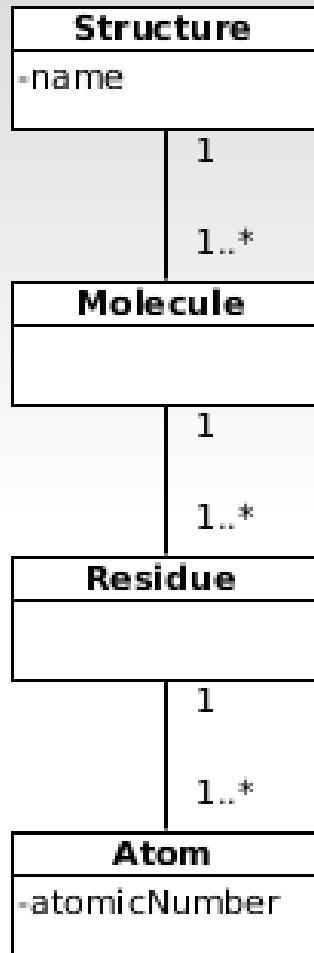


Diagram objektů

Případová studie: Základní doménový model



Doménový model

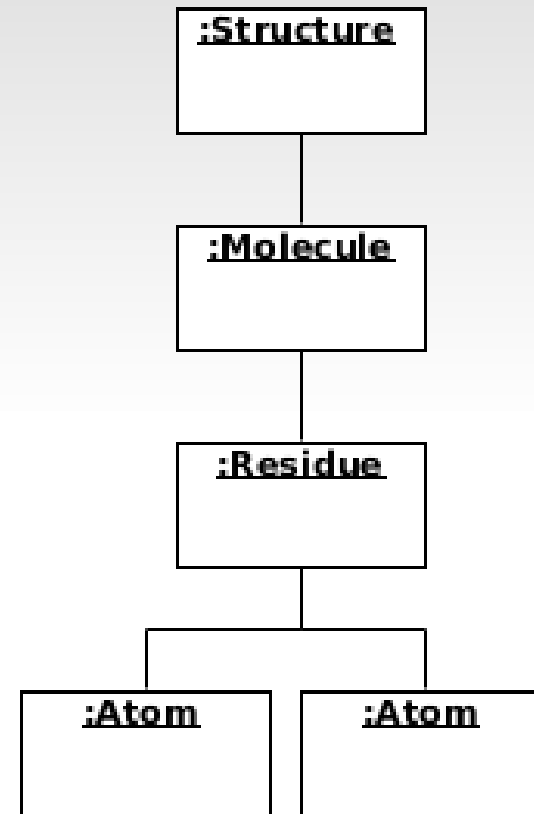
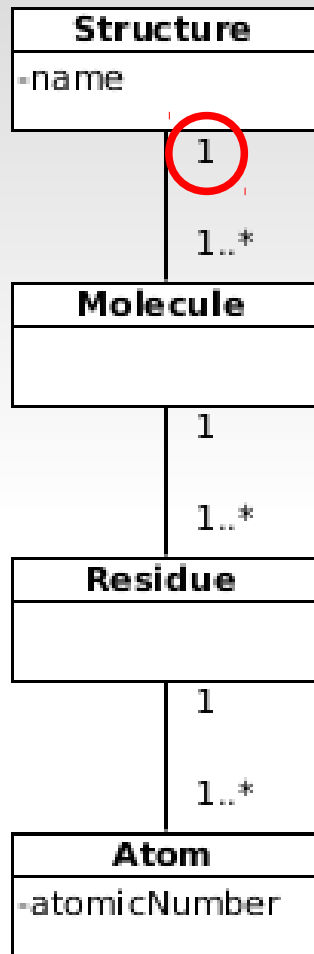


Diagram objektů

Případová studie: Základní doménový model



Doménový model

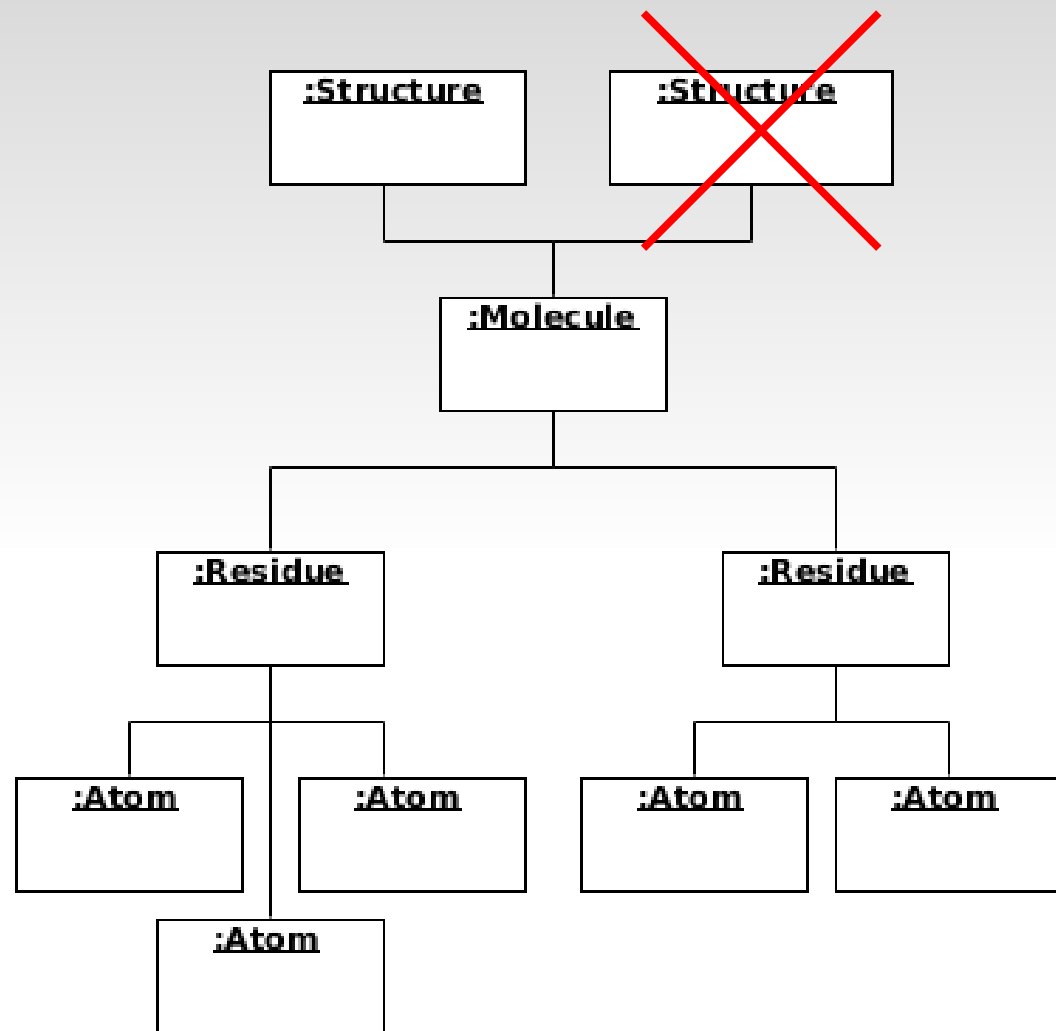
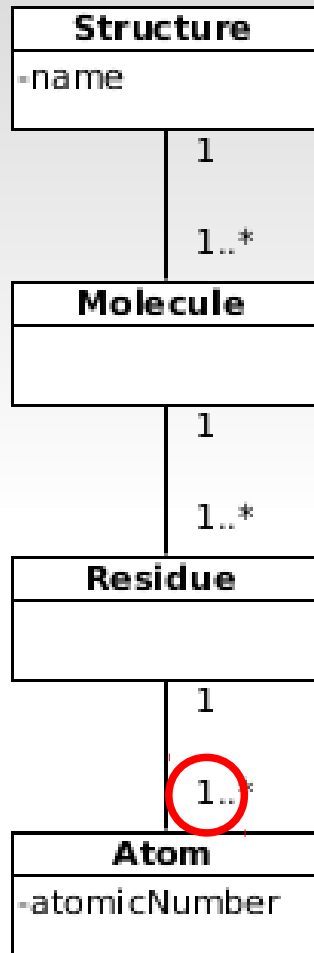
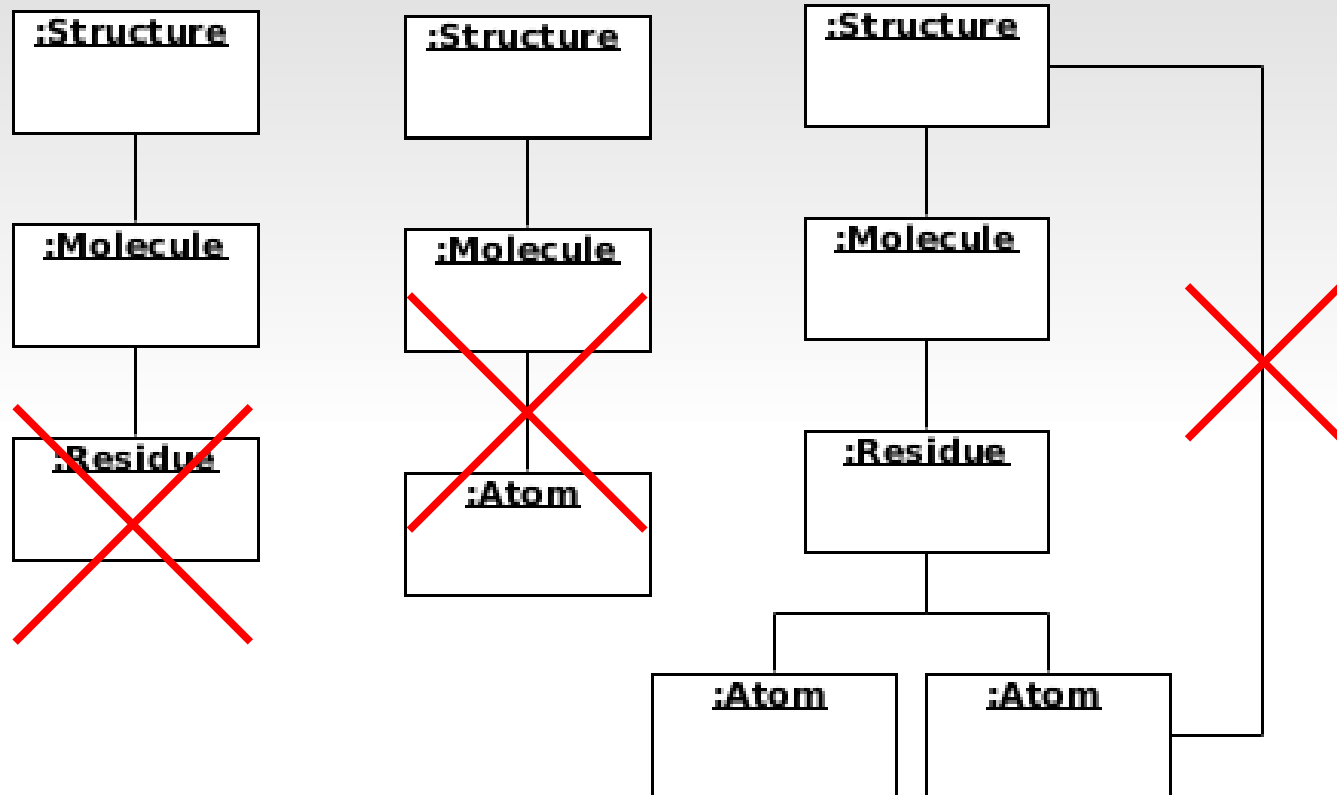


Diagram objektů

Případová studie: Základní doménový model

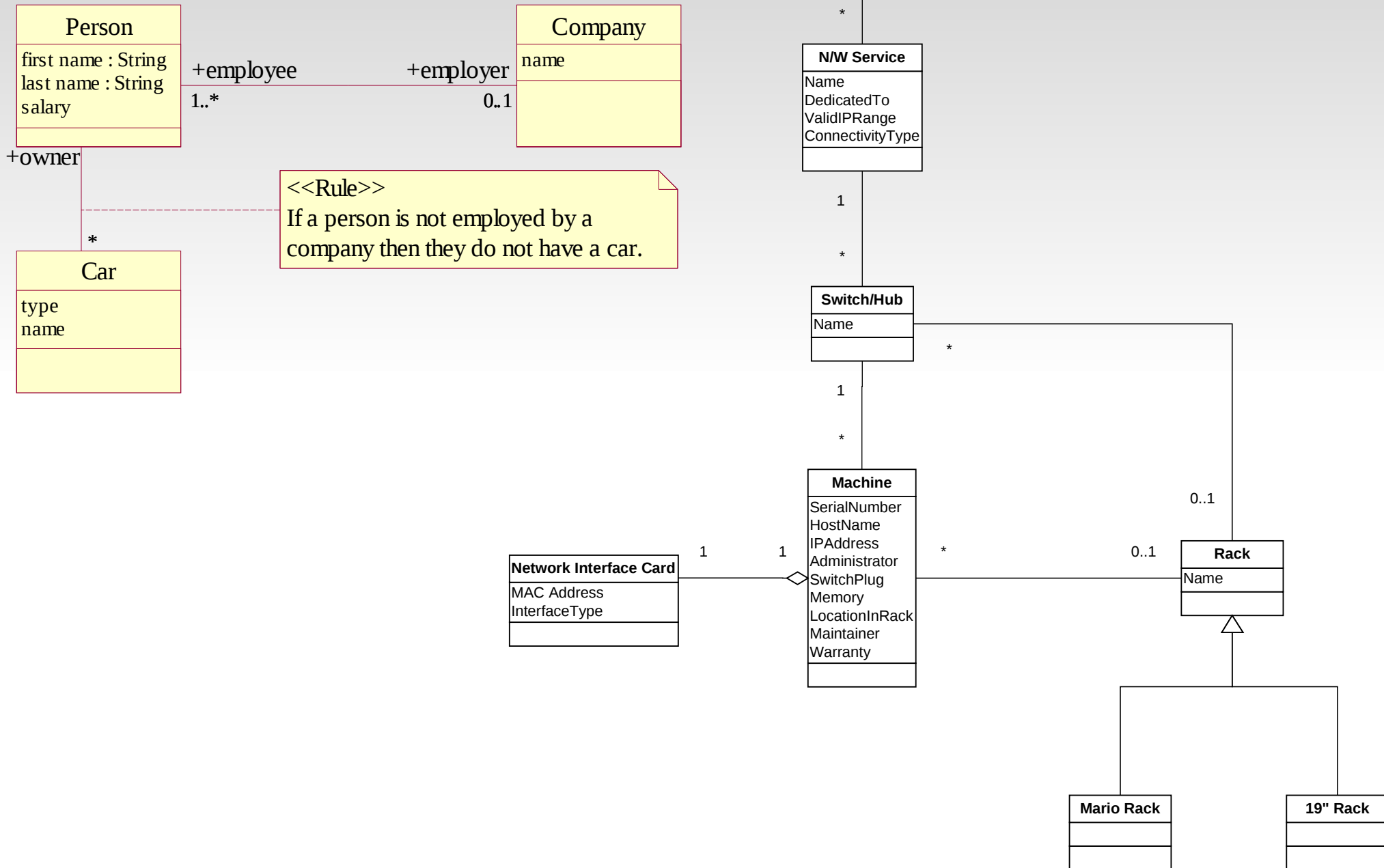


Doménový model



Diagramy objektů

Další příklady

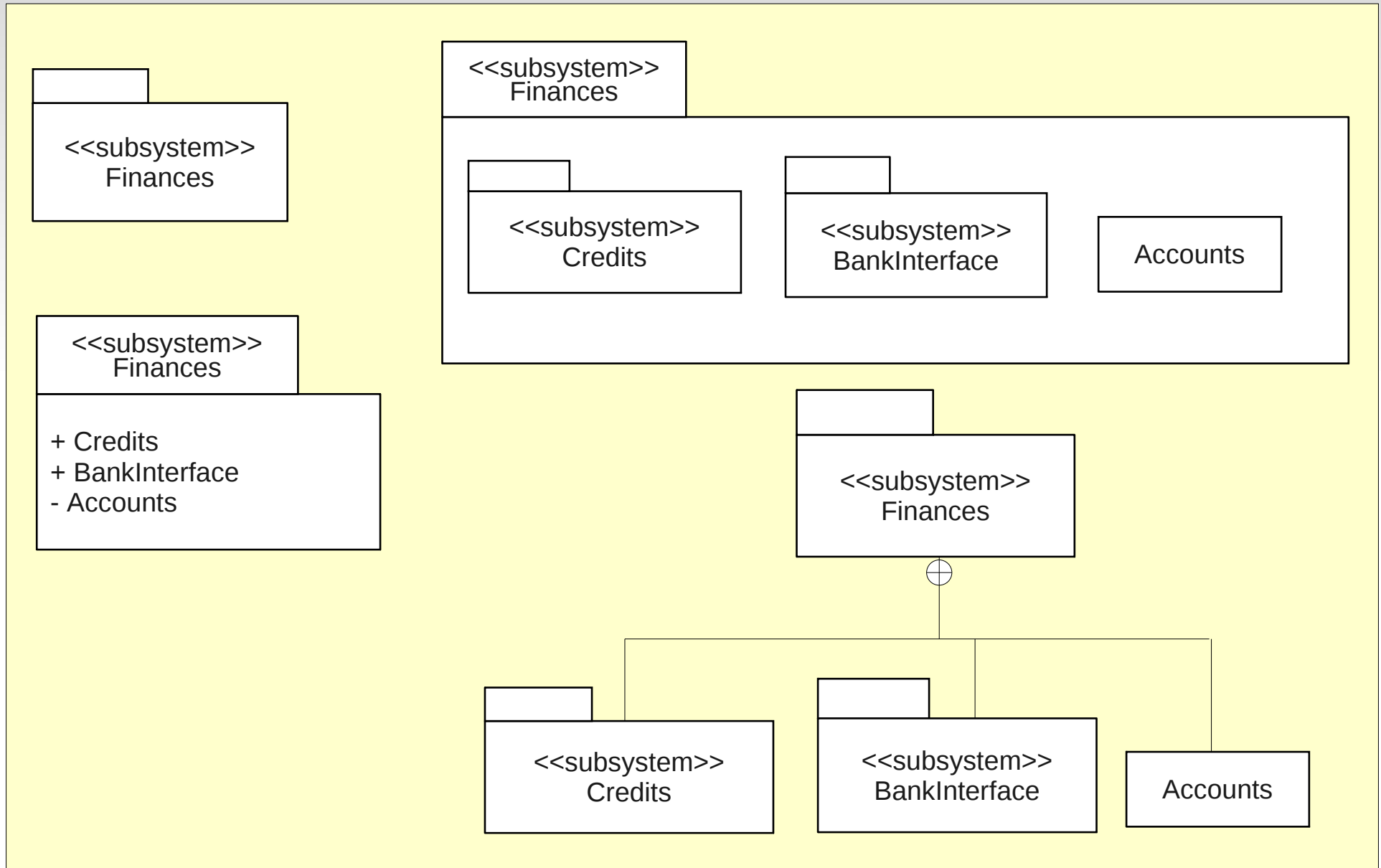


Analytické balíky

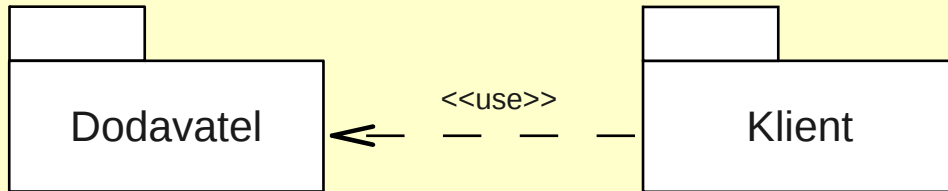
Balíky

- Angl: *Packages*
- Umožňují seskupit třídy a další modelové prvky (compile-time grouping)
 - do systémů (stereotyp `<<system>>`)
 - obsahuje všechny třídy (celý modelovaný systém)
 - do subsystémů (stereotyp `<<subsystem>>`)
 - do soustavy (stereotyp `<<framework>>`)
- Modelované samostatně v diagramu balíků nebo v rámci diagramu tříd
 - diagram balíků nemá třídy, soustředí se opravdu jen na balíky
- Základní vazby: generalizace/realizace, „containment“ a **závislost**
- Balík definuje jmenný prostor pro svoje elementy. Každý element lze jednoznačně identifikovat kvalifikovaným jménem (posloupnost jmen balíků a podbalíků od kořene až po daný element, např. `ProdejceAut.Servis.Zakazka`)
- Balík definuje viditelnost svých elementů (private nebo public), viz. balíky v Javě.

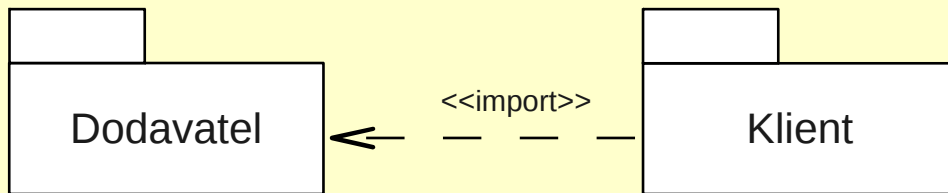
Balíky – notace UML



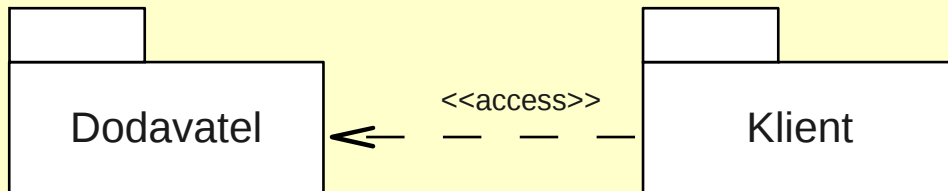
Balíky - závislosti



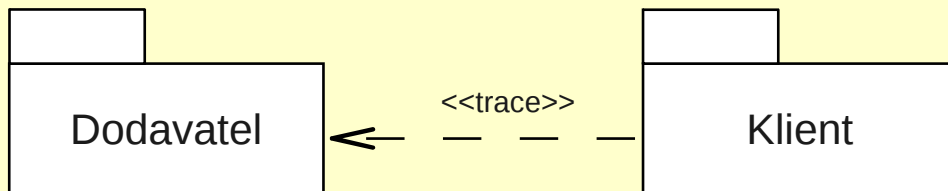
Některý element klienta závisí na některém elementu z dodavatele. Totéž co bez stereotypu. Vhodné pro analýzu, v návrhu se upřesní <<import>> nebo <<access>>



<<use>> + jmenný prostor dodavatele se sloučí se jmenným prostorem klienta, nemusí se používat plně kvalifikovaná jména.



<<use>> + jmenné prostory zůstávají oddělené.

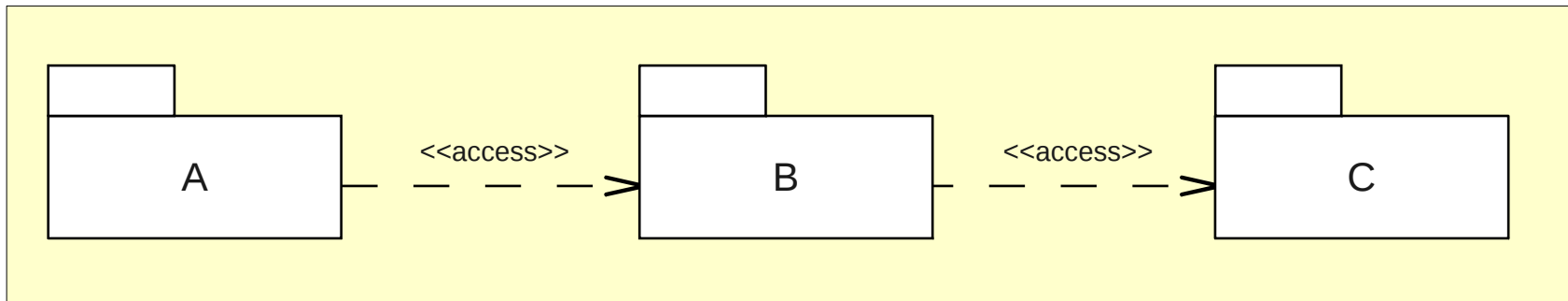


Klient je dalším vývojovým stupněm dodavatele (v průběhu analýzy se balík *Dodavatel* změnil na *Klient*)

Balíky – závislosti (II)

- `<<access>>` **není** tranzitivní:
 - elementy v balíčku A **vidí** elementy v balíčku B,
 - elementy v balíčku B **vidí** elementy v balíčku C,
 - elementy v balíčku A **nevidí** elementy v balíčku C.

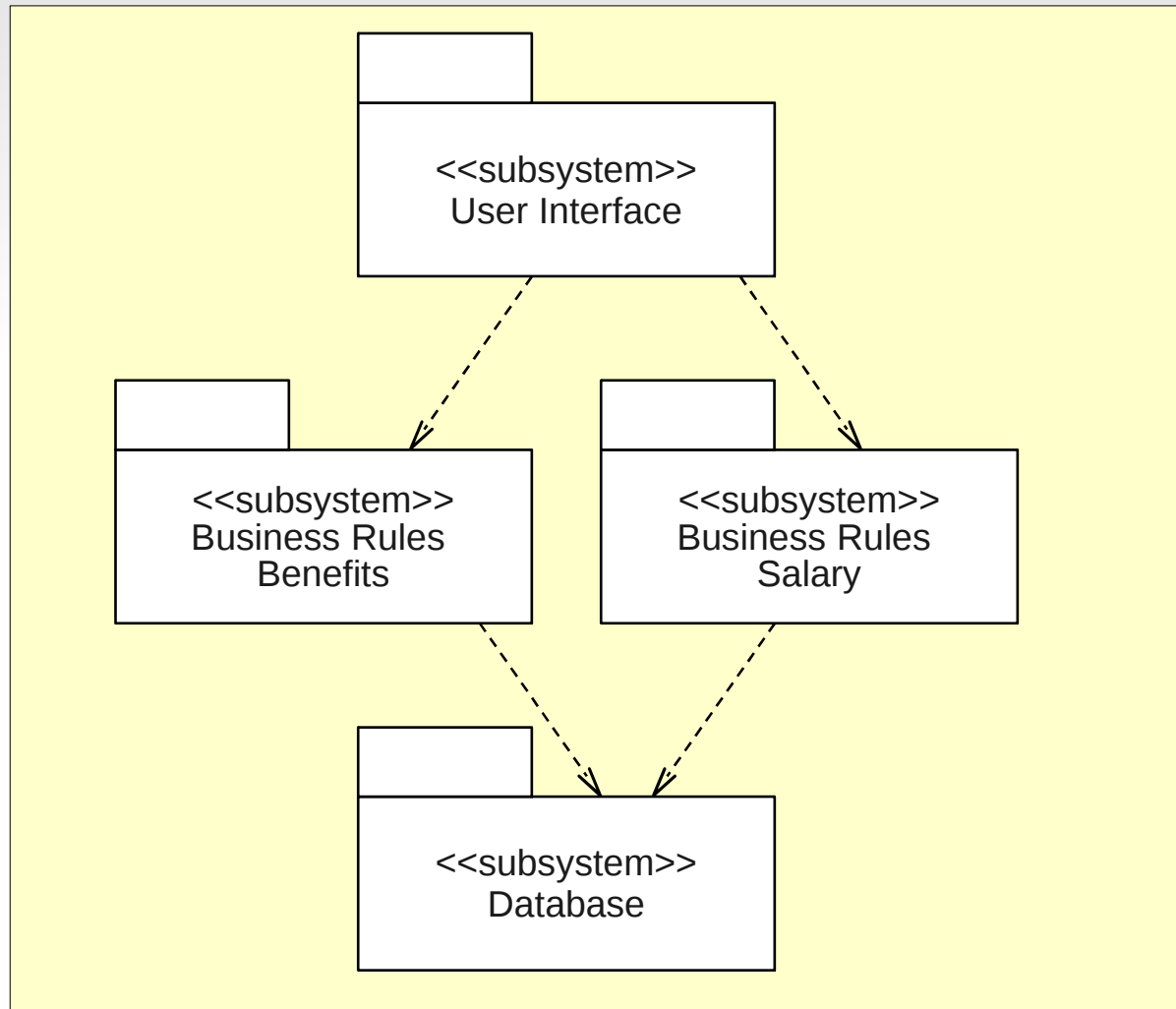
=> soudržnost modelu, jasnější zodpovědnost



- `<<import>>` **je** tranzitivní:
 - elementy v balíčku A **vidí** elementy v balíčku C,
 - častější než `<<access>>`

Balíky – závislosti (III)

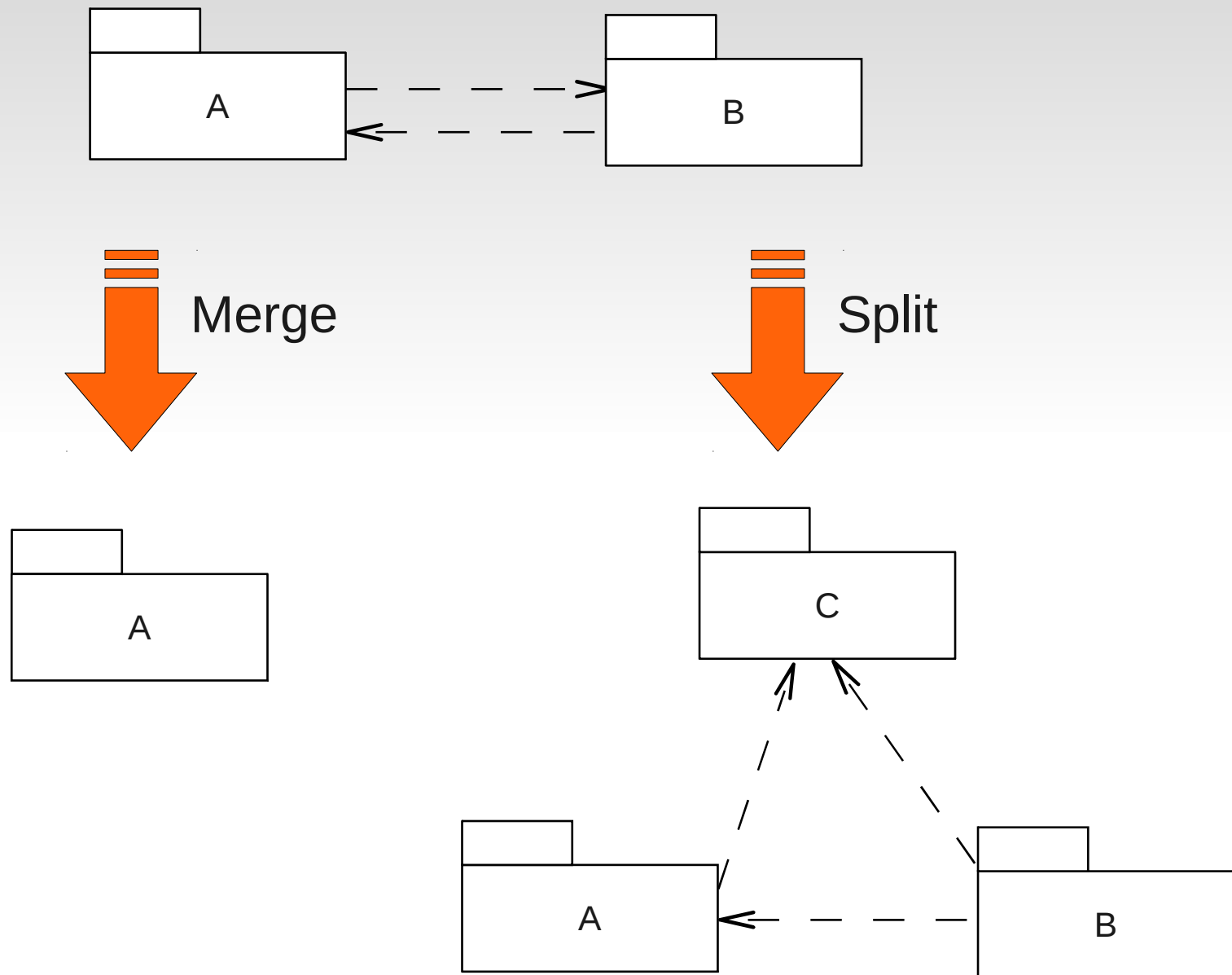
Příklad diagramu architektury ukazující subsystémy a jejich závislosti



Minimalizace propojení (coupling)

- Nalezení analytických balíků
 - Hledejte shluky tříd se silnou kohezí (silné vztahy mezi třídami)
 - Hledejte hierarchie dědičností.
 - Případy užití mohou být základem balíků
 - Snaha o kohezi systému i z pohledu business procesů
 - Ne vždy to jde
- Minimalizace propojení (coupling):
 - Minimalizujte závislosti mezi balíky
 - Maximalizujte závislosti uvnitř balíků
 - Minimalizujte počet veřejných (public) elementů
 - Maximalizujte počet soukromých (private) elementů
- Ne příliš složité (hluboké) stromy balíků
- Vyhněte se cyklické závislosti mezi balíky

Cyklická závislost balíků



Pojišťovna: Diagram balíků

Demo

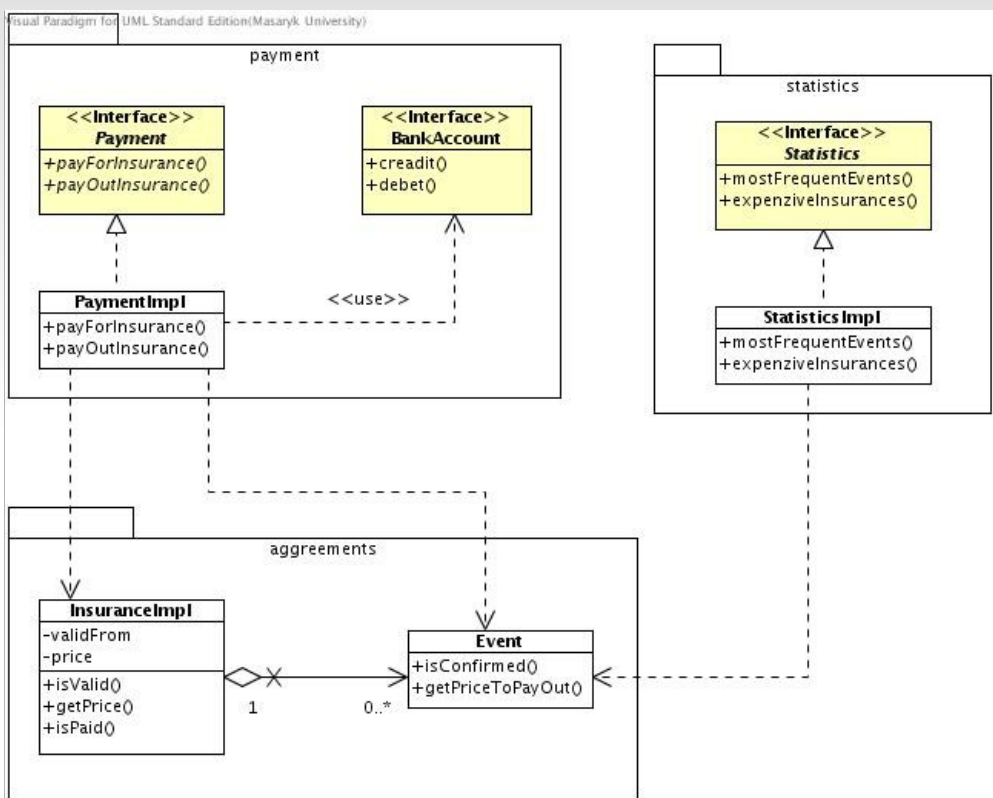
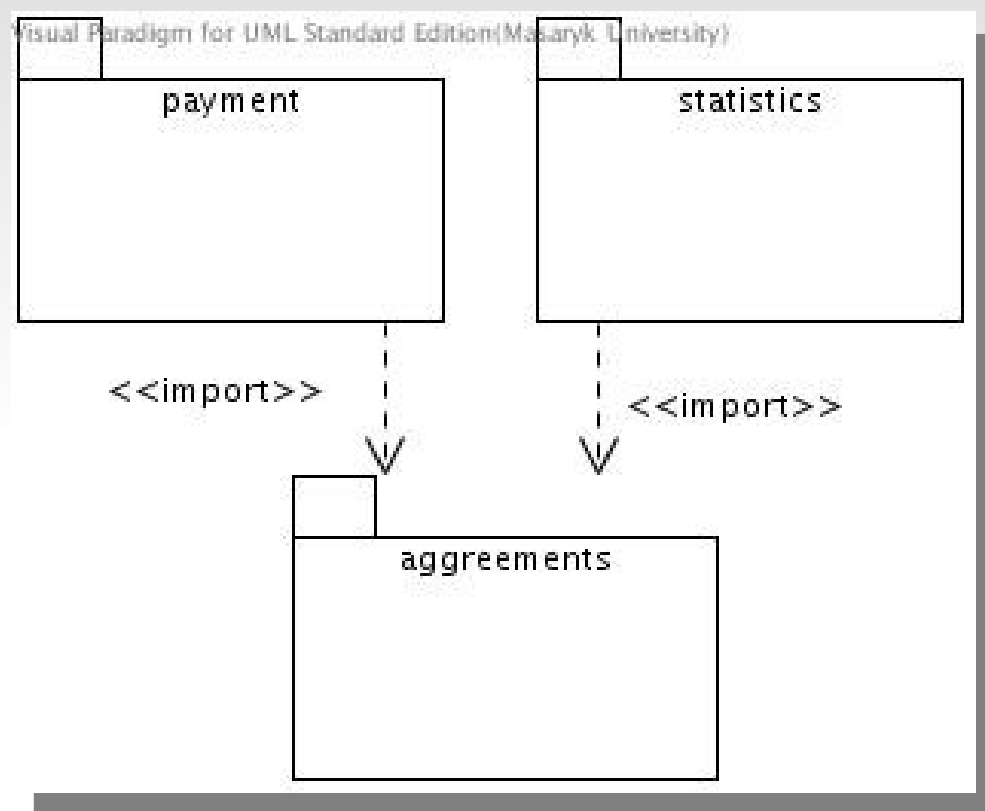


Diagram tříd včetně balíků (návrhový model)



Přehlednější diagram balíků