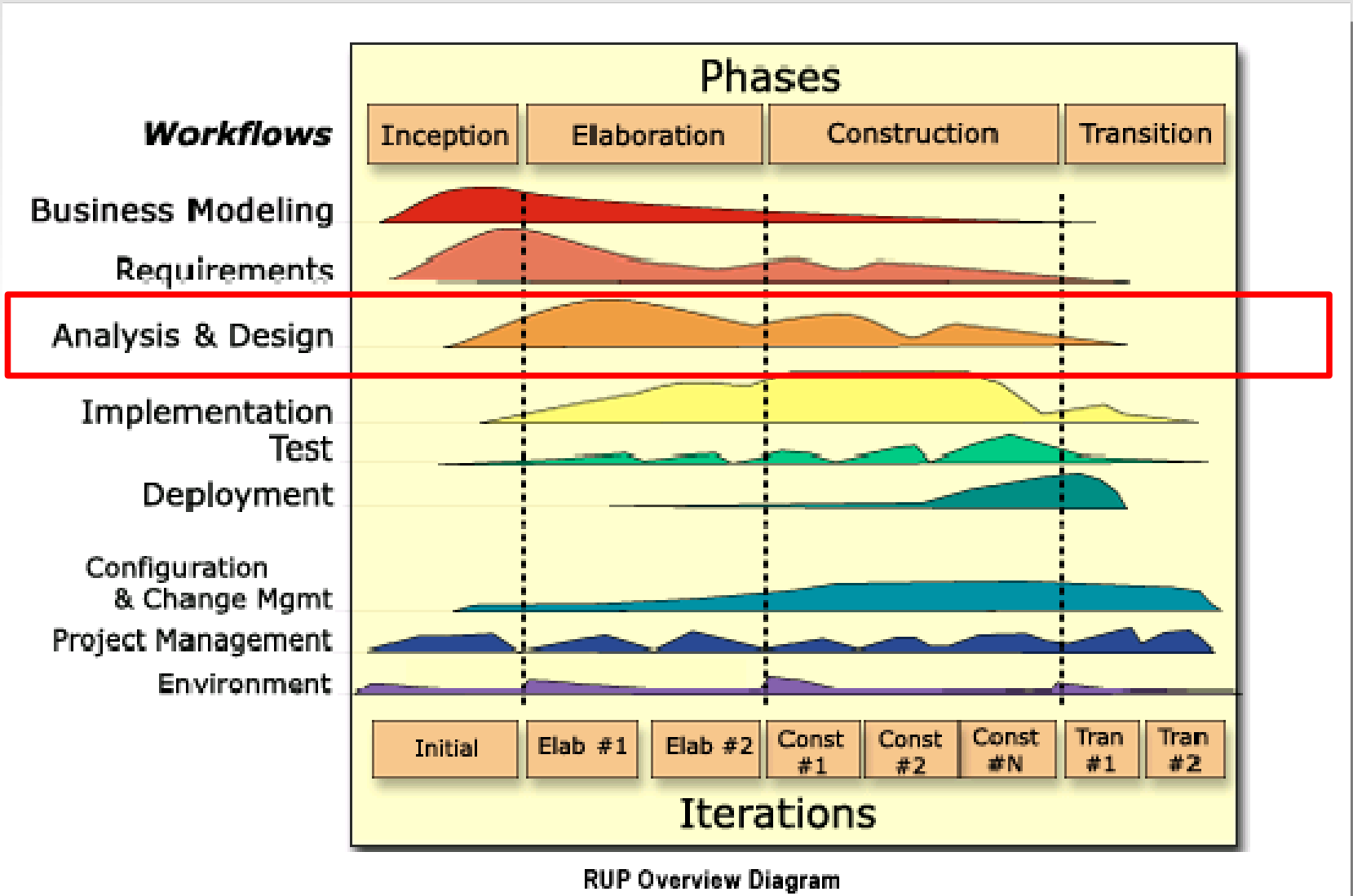
Analýza

analytické modely, objekty, třídy

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

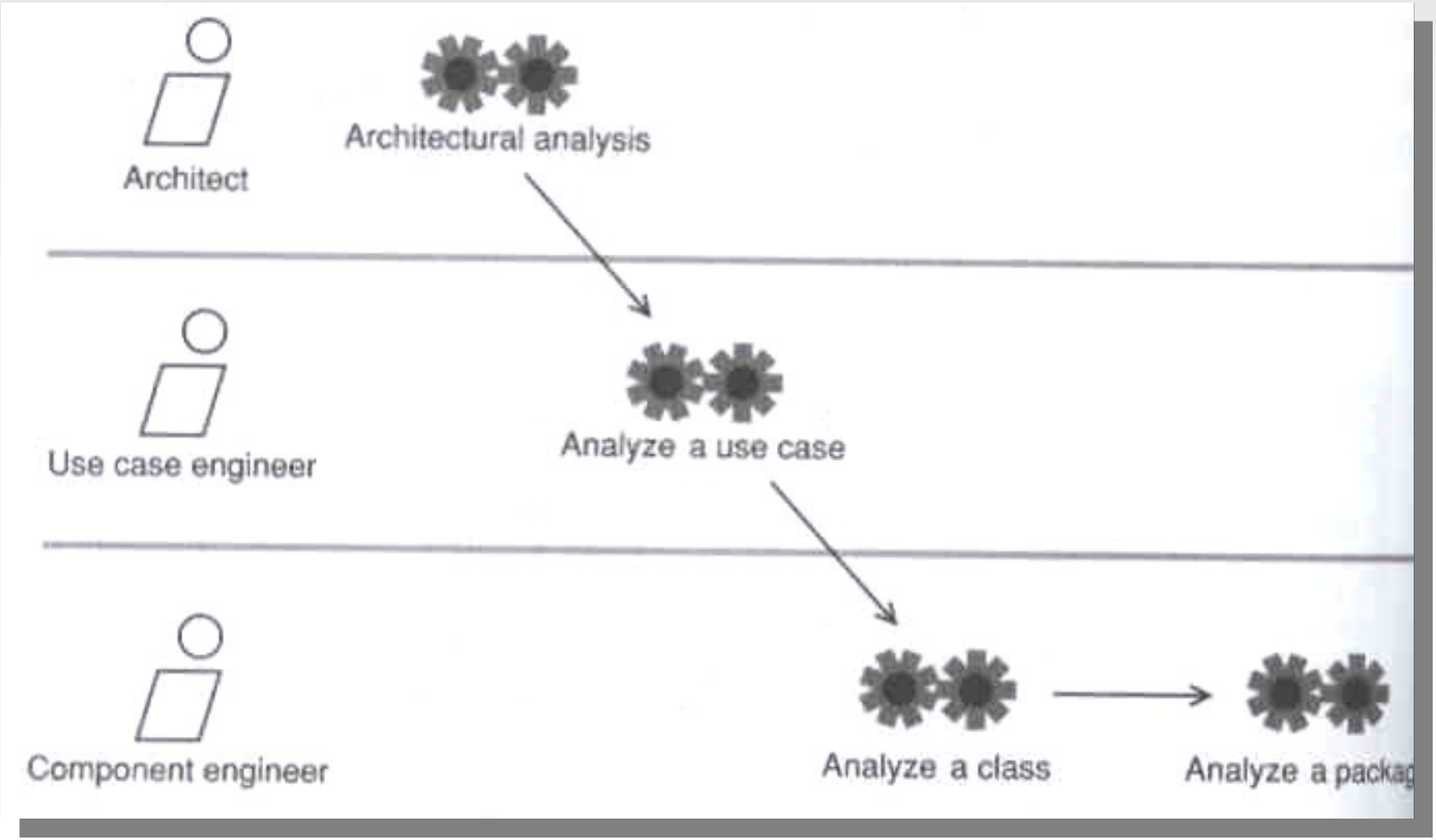
Analysis workflow

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



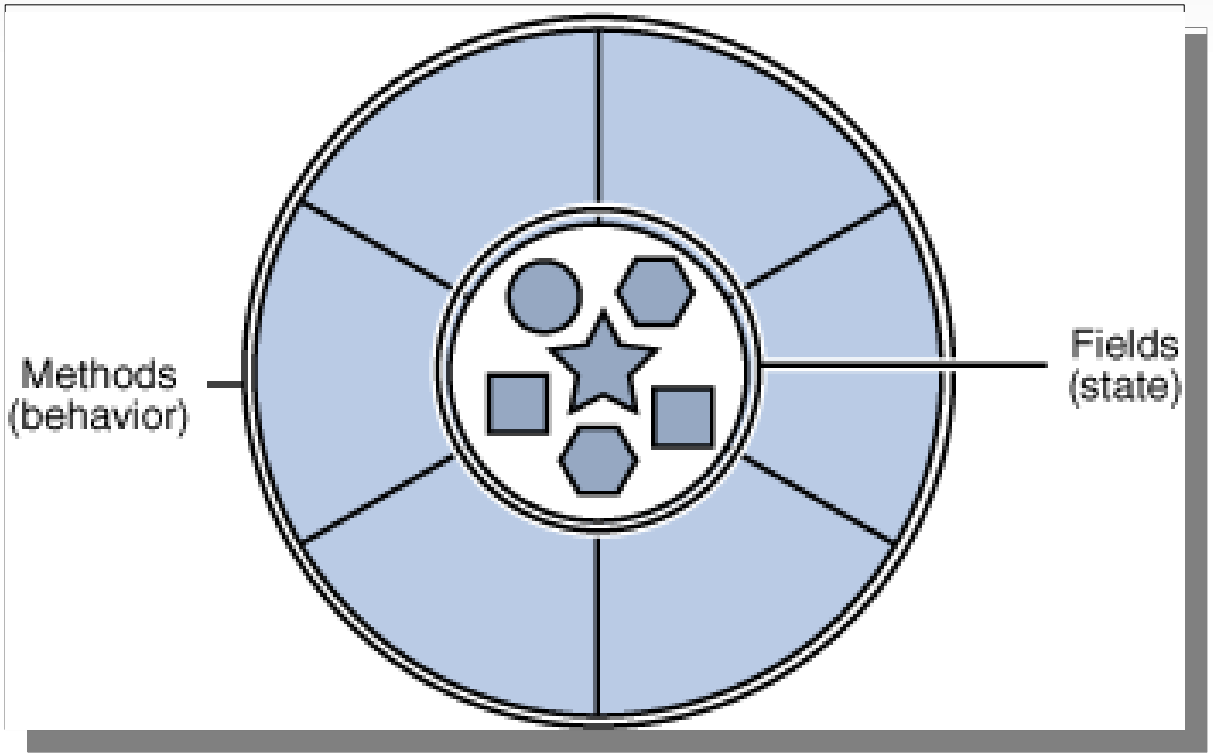
Analysis workflow detail

- Pro pochopení aktivit je třeba nejdříve pochopit objekty a třídy, 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).

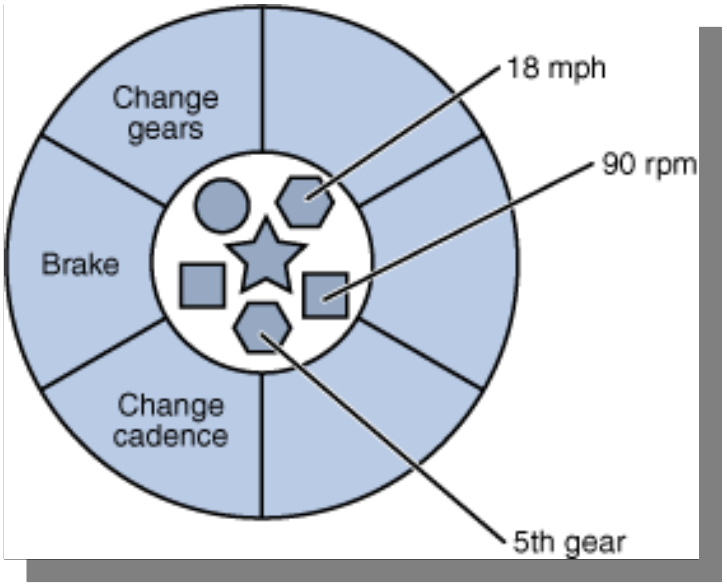
Objekt: Stav

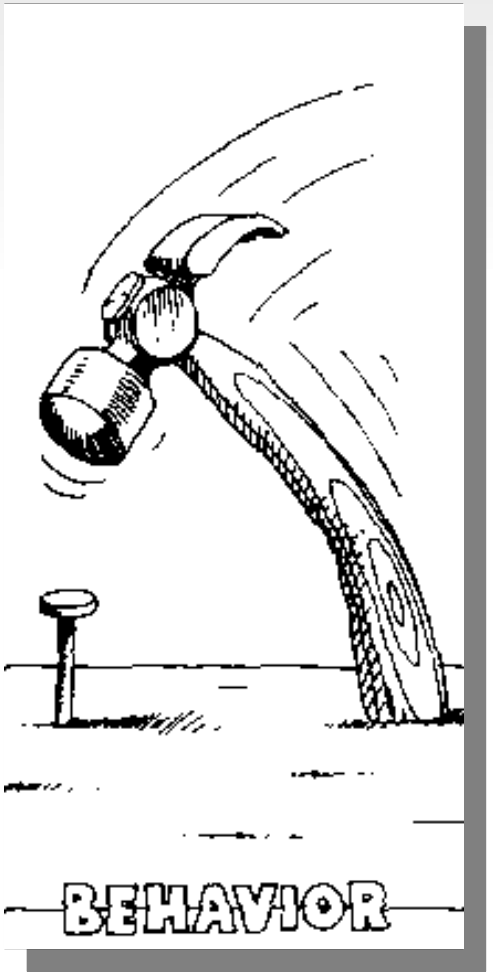
- Stav je určen ***hodnotami atributů*** a ***vazbami/vztahy na ostatní objekty*** v daném časovém okamžiku
- Př. pro tiskárnu:

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

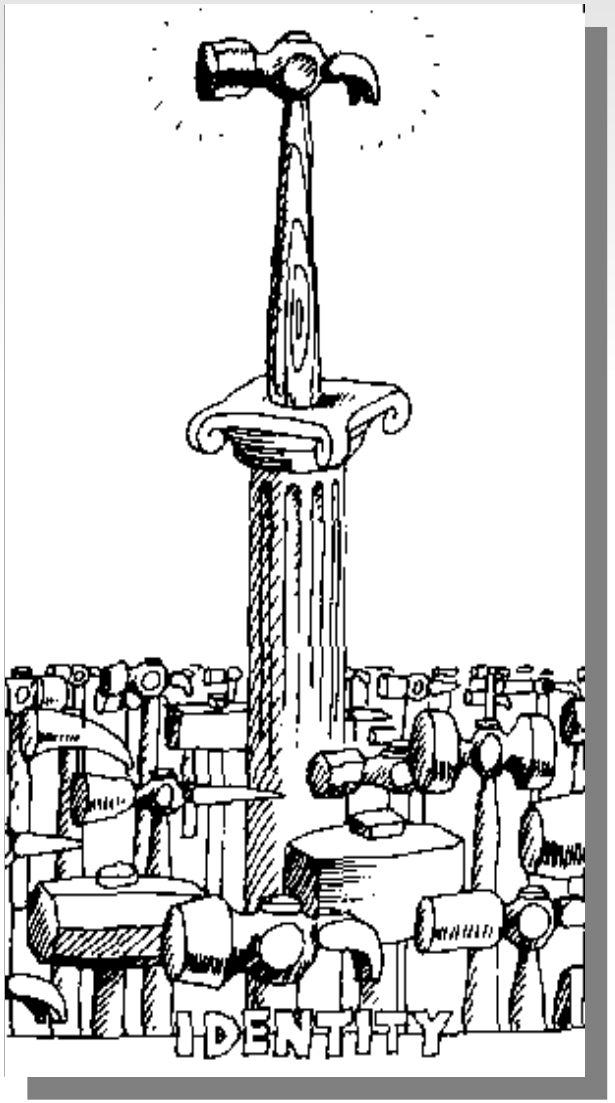
- Chování vyjadřuje, jak objekt koná a reaguje
- Chování je specifikováno operacemi. Každá operace tedy definuje část chování objektu.
- Chování se může měnit v závislosti na stavu
  - Př: chování tiskárny při stavu „DošlaNáplň“
- Vyvolání operace může vést ke změně stavu
- Implementace operace se nazývá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

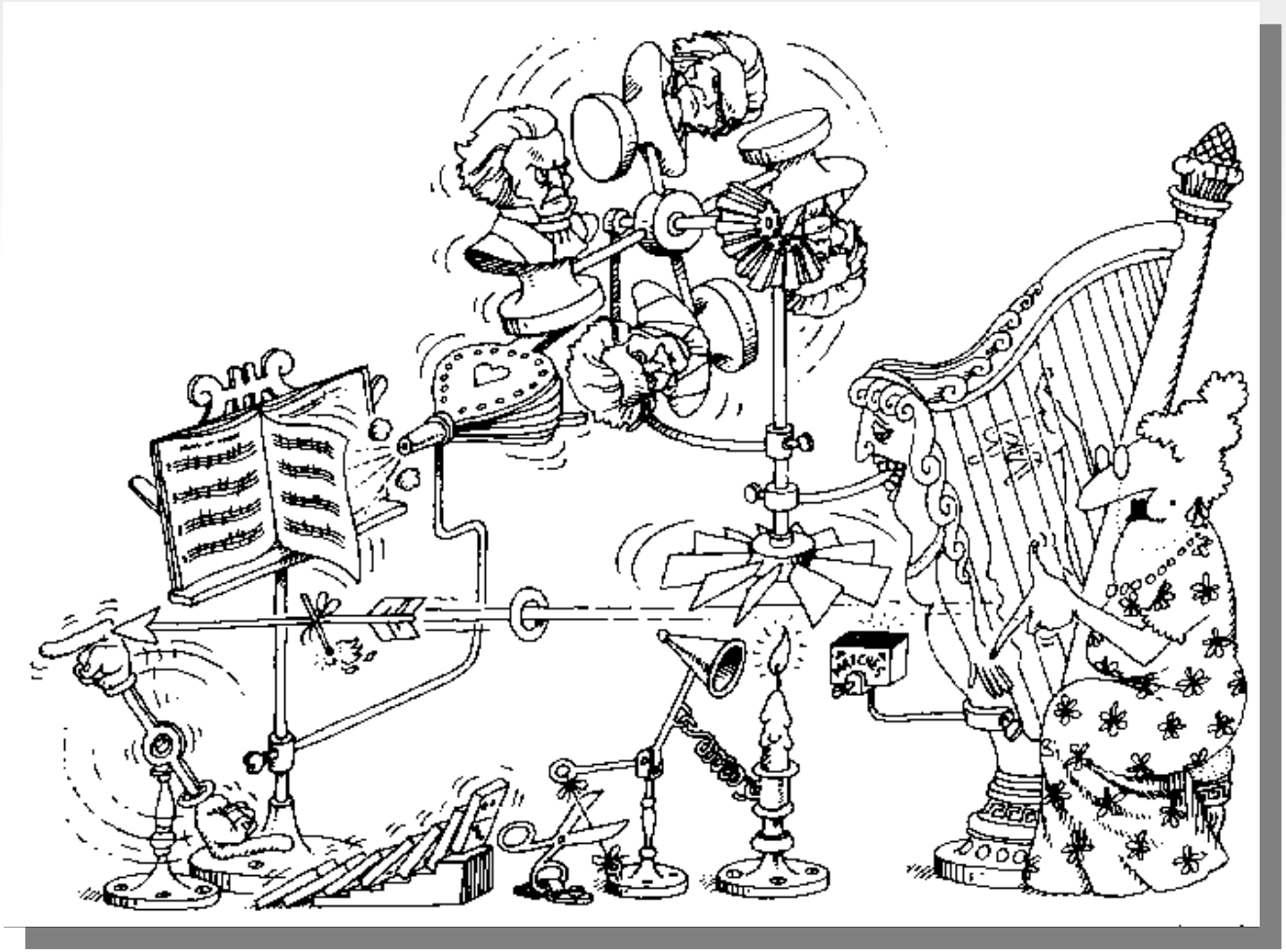


Itentita – příklad

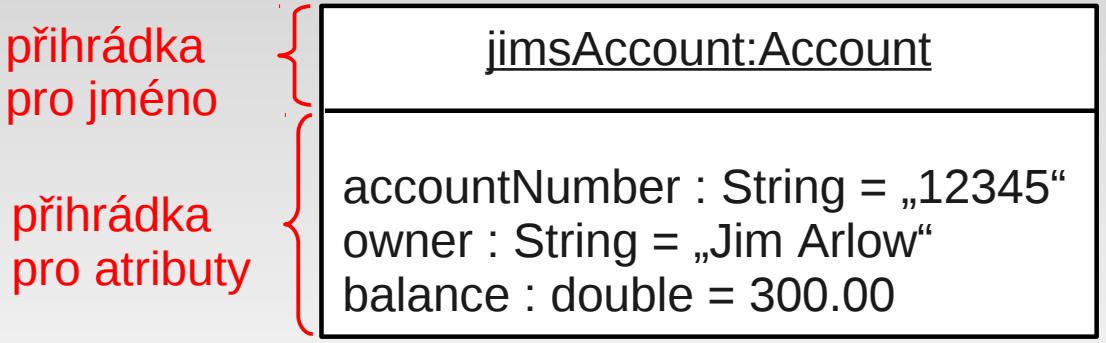
- Židle nabízené internetovým obchodem. Jak mohou být identifikovány?
- Adresa v paměti (odpovídá reálnému světu):
  - Dvě naprosto stejné reálně židle stojící vedle sebe jsou dva různé objekty (každá židle zabírá svůj prostor v daném čase)
  - Dvě naprosto stejné „virtuální“ židle jsou stejné, pokud 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

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.



Objekty – notace UML



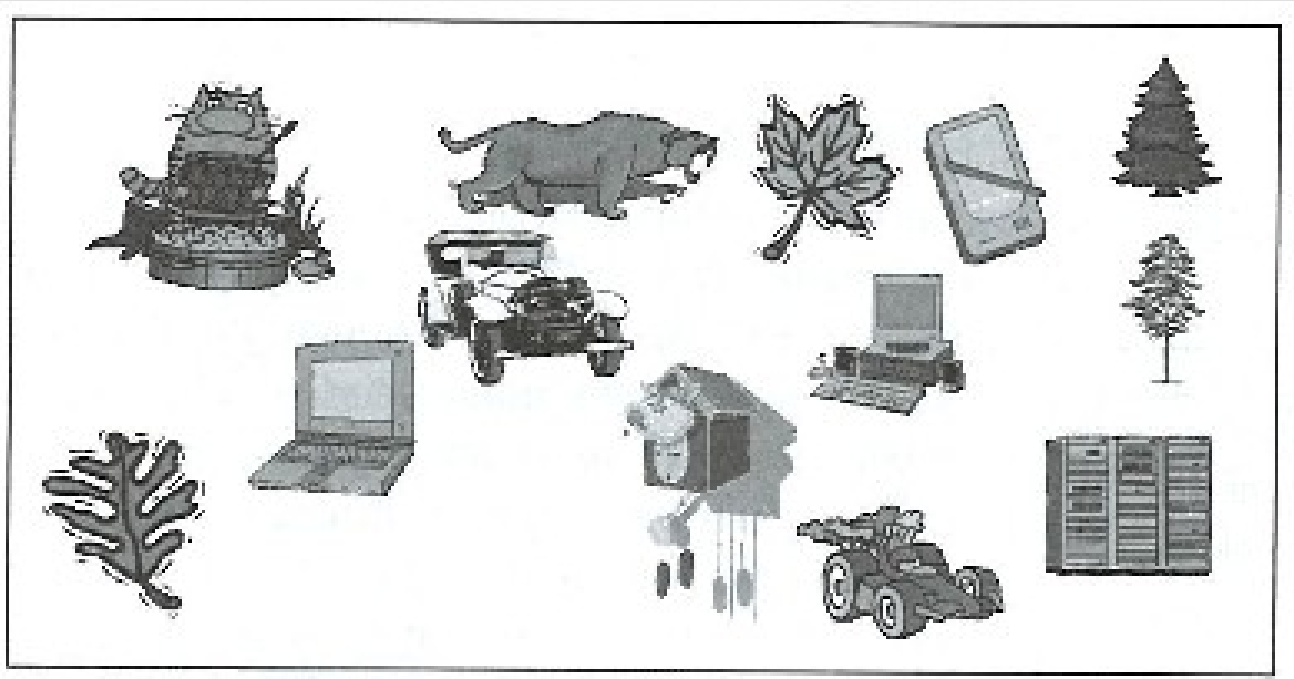
- *Syntaxe názvu: jméno objektu : jméno třídy [jména stavů]*
  - :Account – anonymní objekt dané třídy
  - jimsAccount – konkrétní objekt bez specifikované třídy, užitečné především v počátečních fázích analýzy
  - jimsAccount:Account – konkrétní instance třídy; nutné pokud je v diagramu více instancí 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.

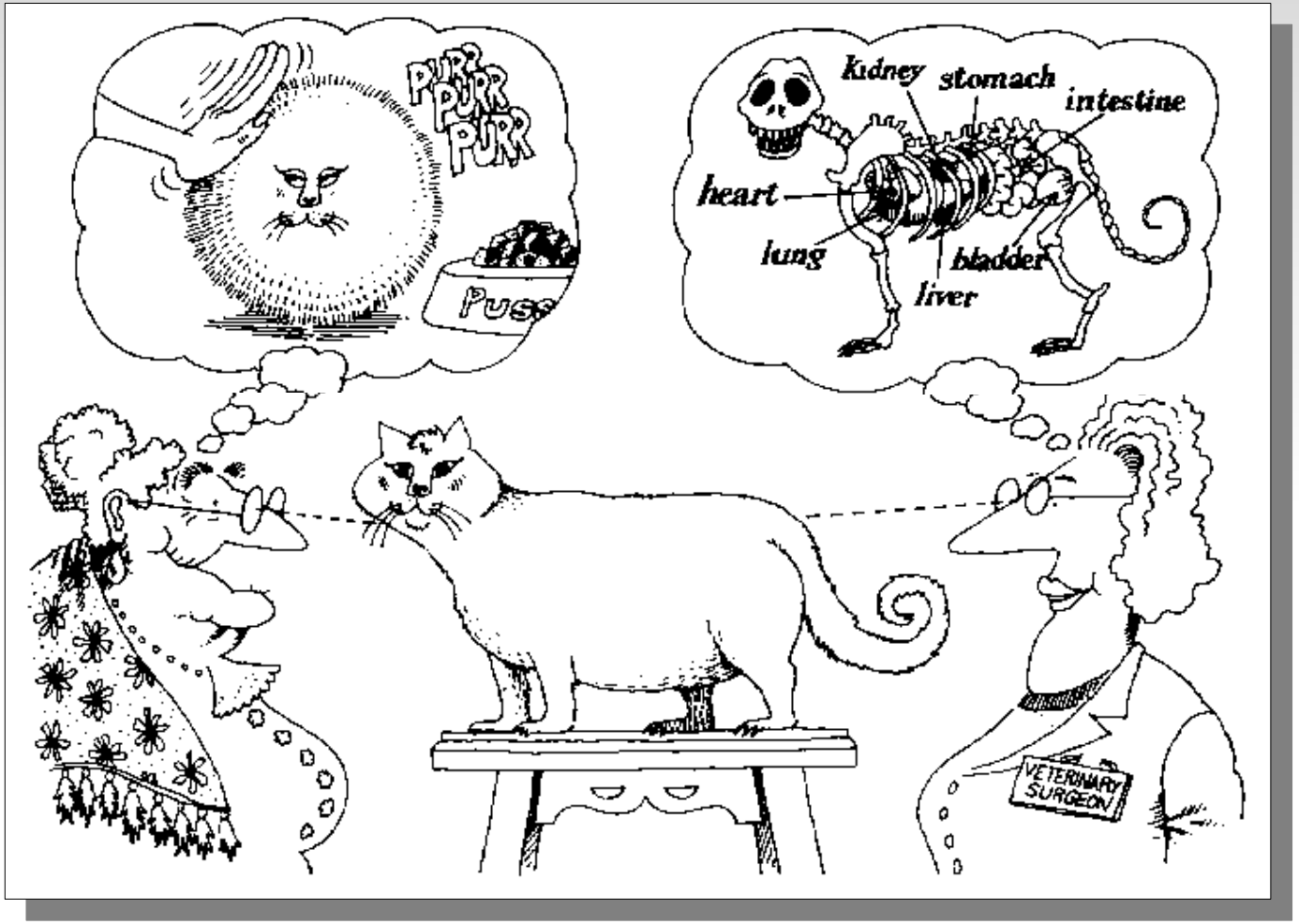
Abstrakce (klasifikace objektů a tříd)

- Kolik tříd je schováno je na obrázku?



- třída koček
- třída tlustých koček, které mají rády jídlo
- třída stromů
- třída listů
- atd., atd.

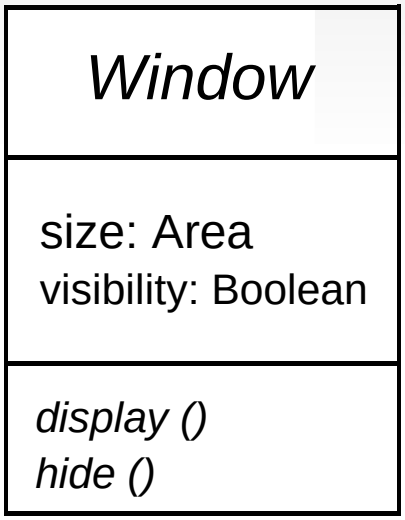
Abstrakce (klasifikace objektů a tříd)



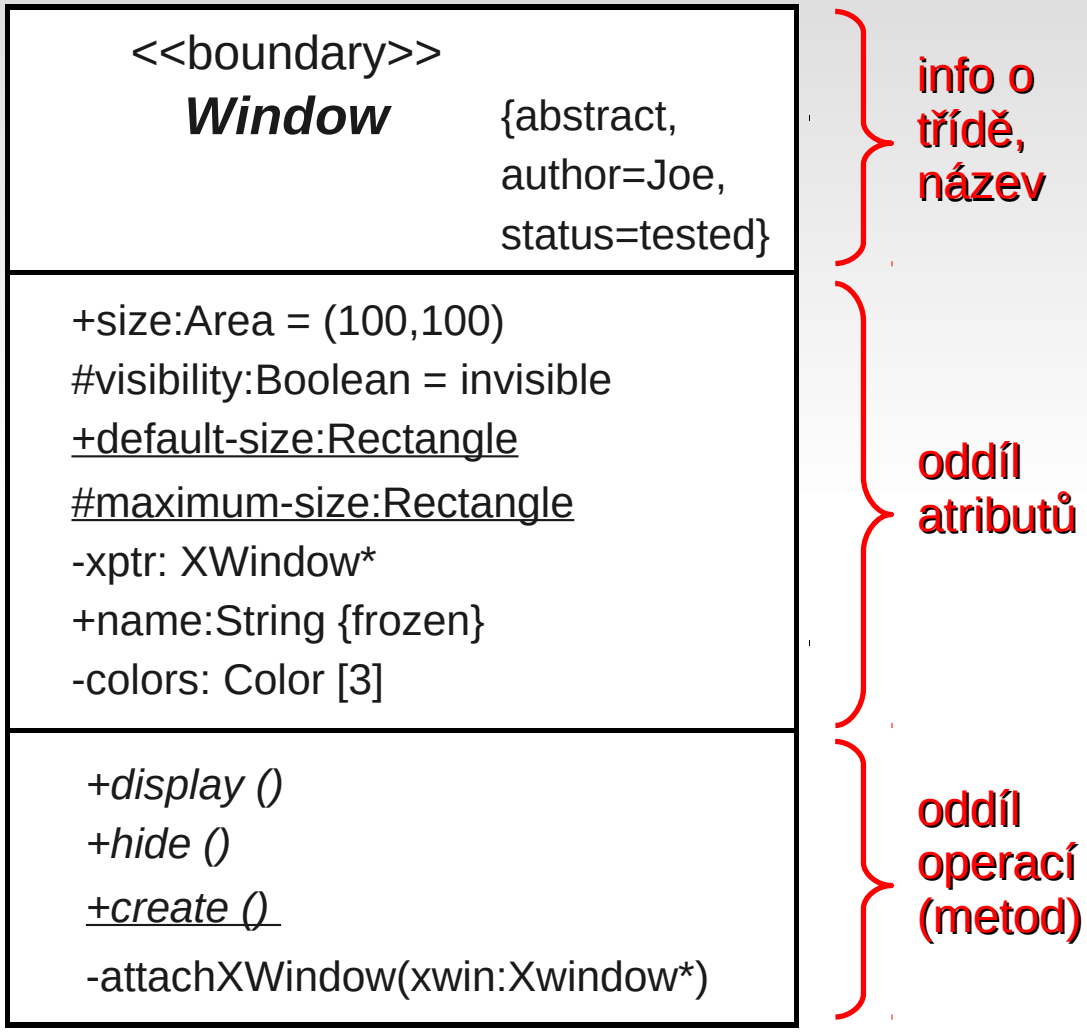
- Nalezení vhodného klasifikačního schématu je jeden z klíčových úkolů OO analýzy

Třídy – notace UML

potlačené detaily



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



detaily implementační úrovně

Třídy – notace UML (II)

Jméno třídy:
tzv. UpperCamelCase, jednotné číslo, nezkracovat

Struktura atributů:
viditelnost název: typ [násobnost] = počátečníHodnota

nepovinné povinné nepovinné

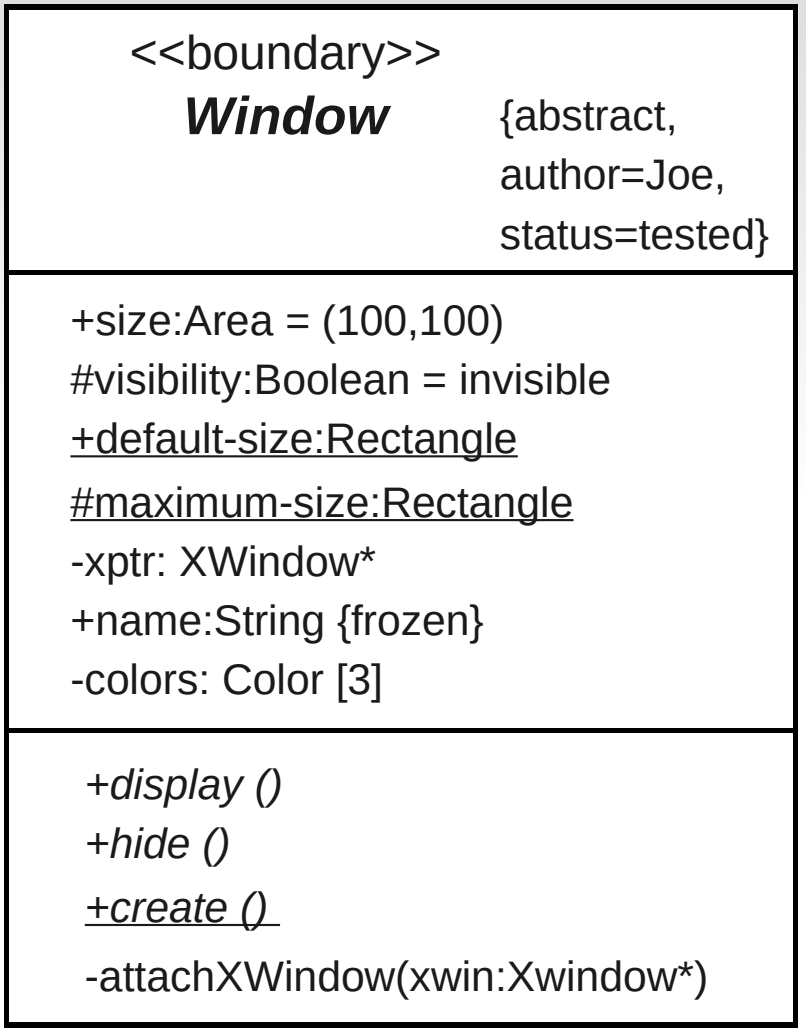
Viditelnost:
+ public
- private
protected
~ package

Počáteční hodnota:
vyjadřuje důležité omezení
v analýze řídký jev

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

Typ:

Integer	celé číslo
UnlimitedNatural	celé číslo >= 0, nekonečno = *
Boolean	*true* nebo *false*
String	řetězec v uvozovkách
Real	reálné číslo
Třída	odkaz na jinou třídu



Třídy – notace UML (III)

Struktura operací:

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

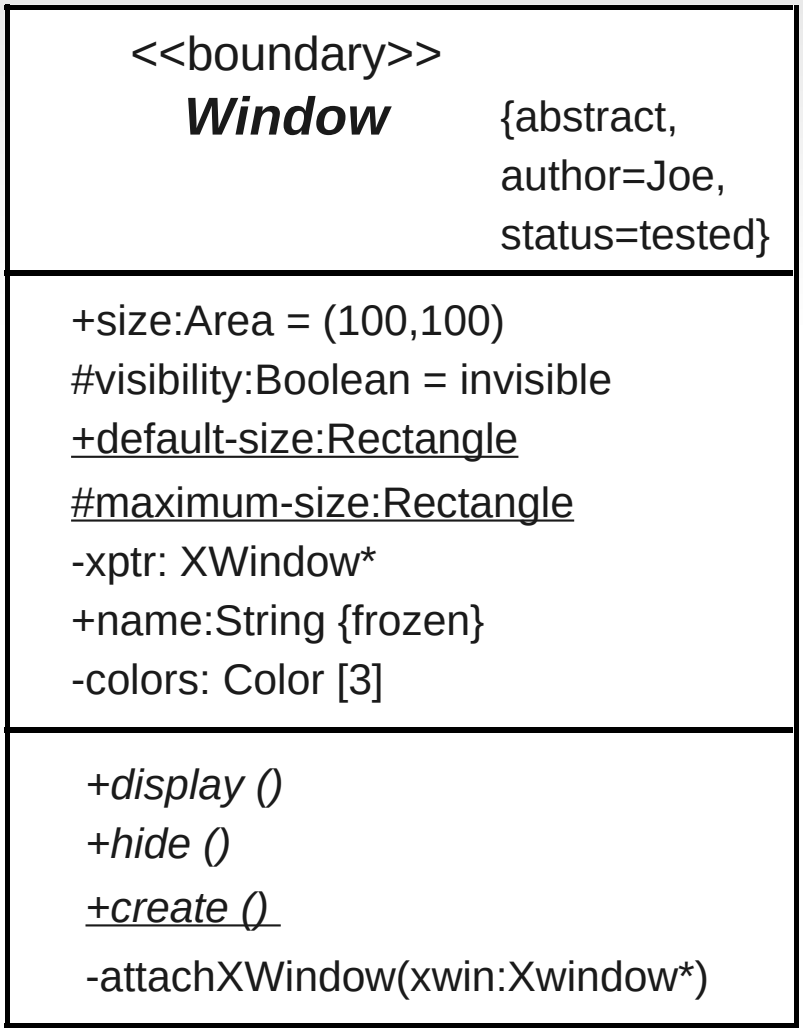
signatura operace

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)



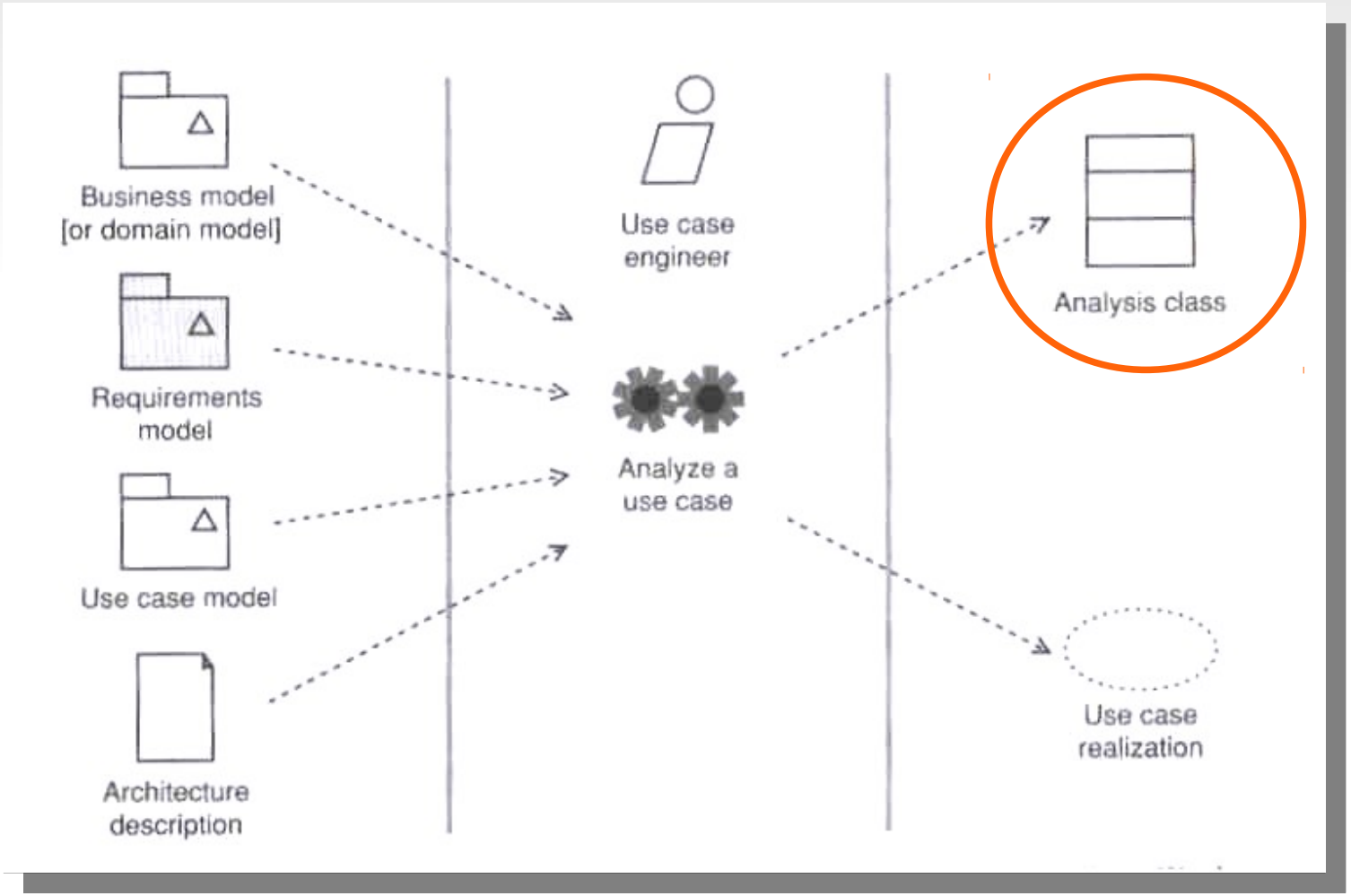
Rozsah platnosti atributů a operací

- 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++

Nalezení analytických tříd

UP aktivita: Analýza případů užití

- *Architecture description* – důležité aspekty softwarové architektury v podobě UML modelů, popisného textu apod. Architekturami se budeme zabývat později

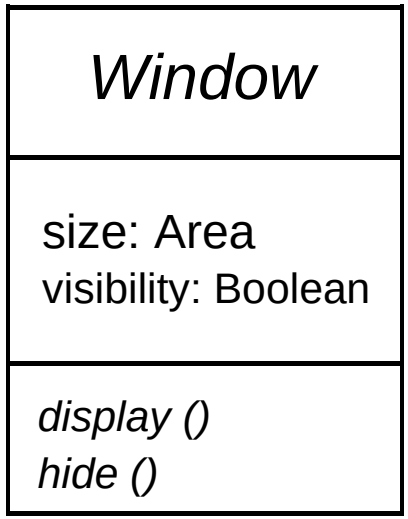


Pravidla pro vytváření analytického modelu

- 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)

- Využití stereotypů RUPu
  - zaměříme se na tři rozdílné typy analytických tříd
    - <<boundary>>
      - třída, která zprostředkovává interakci mezi systémem a aktéry
      - každá komunikace aktéra se systémem se děje prostřednictvím nějakého objektu v systému. Tento objekt je instancí “hraniční” třídy
    - <<control>>
      - třída, která zapouzdřuje chování uvnitř případů užití
      - je třeba přemýšlet, jak docílit chování popisované případem užití
    - <<entity>>
      - třída, která modeluje persistentní informace
      - jsou pasivní (často je get/set operace), spravované kontrolními třídami
      - ekvivalent datového modelu
- Další zdroje tříd
  - fyzické objekty (lidé, pracoviště, ...), dokumenty (formuláře, objednávky, ...), známá rozhraní (terminály, periferní zařízení), ...

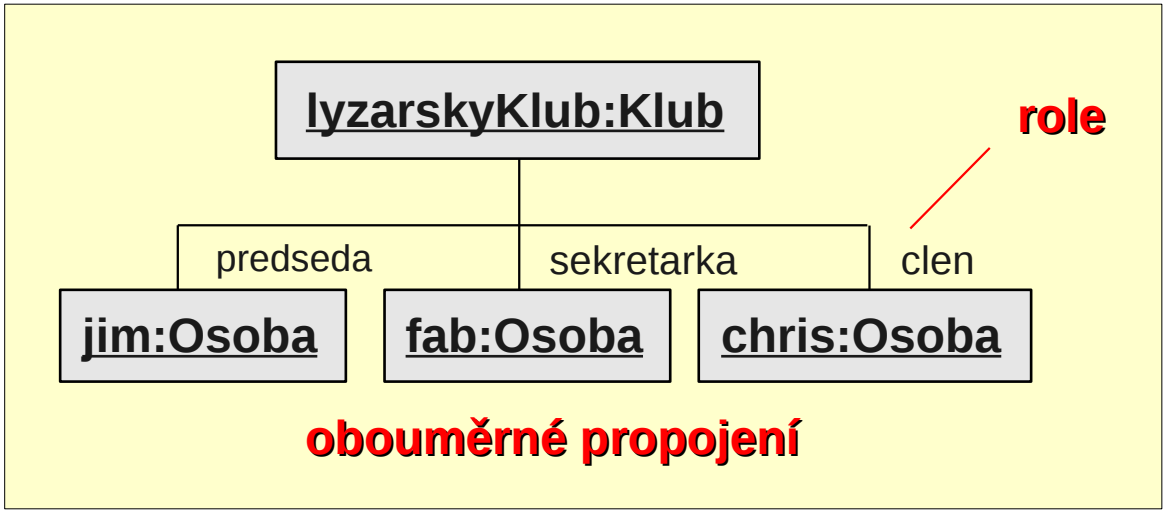
Vztahy (propojení) mezi objekty a třídami

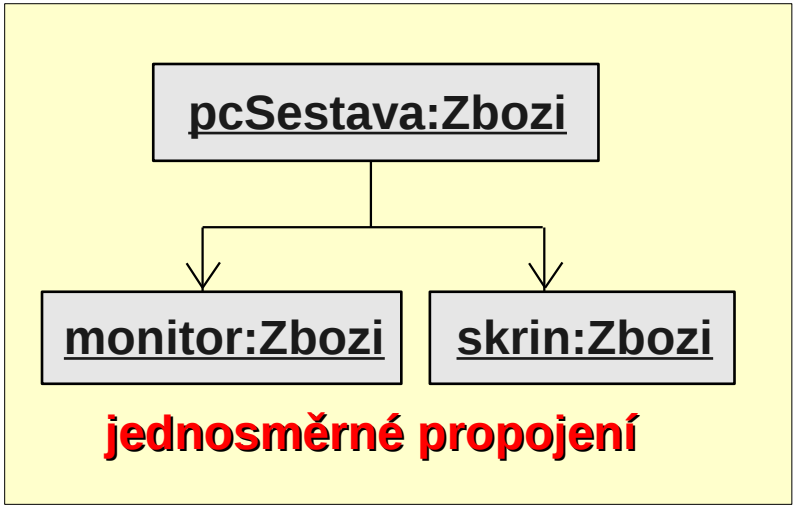
Propojení (link)

- Pokud chce jeden objekt komunikovat s jiným objektem, musí nejdříve dojít k jejich „spárování“.
- Propojení (link) představuje dlouhodobé „spárování“ objektů.
  - Ke komunikaci není vždy nutné dlouhodobé „spárování“. Pokud např. druhý objekt dostaneme jako parametr metody, jedná se o dočasné krátkodobé „spárování“, které nepředstavuje propojení (link).
- Různé typy realizace propojení v různých jazycích, např.
  - odkaz na objekt (reference) – Java, C++
  - ukazatele na objekt (pointer) – C++
  - vložení jednoho objektu do druhého – Java, C++

Diagram objektů

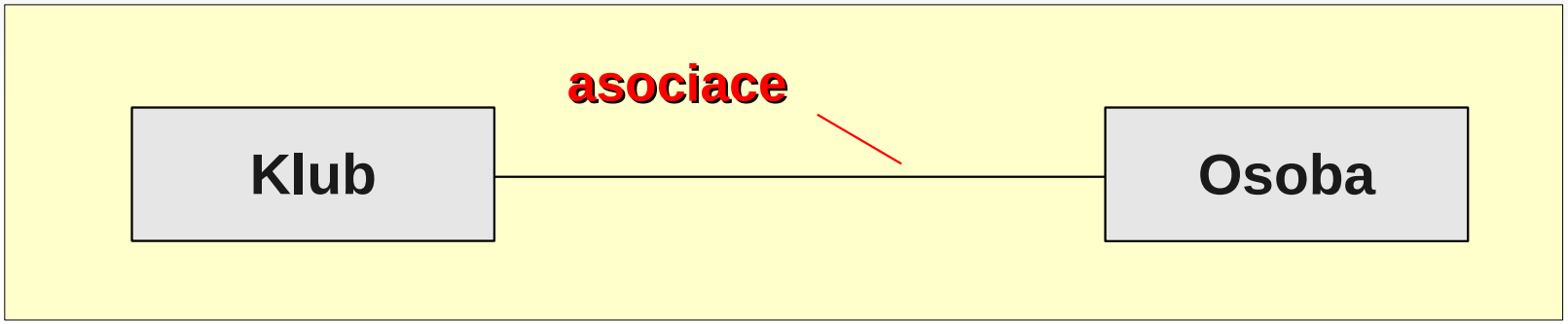
- Snímek v časovém bodě, který ukazuje podmnožinu objektů, jejich stavů, hodnot atributů a propojení.
- Vhodné zejména pro komunikaci se zákazníky (konkrétní příklad hierarchie, časový snímek systému, …)
- Šipka upřesňuje směr řízení – viz směr řízení u asociací

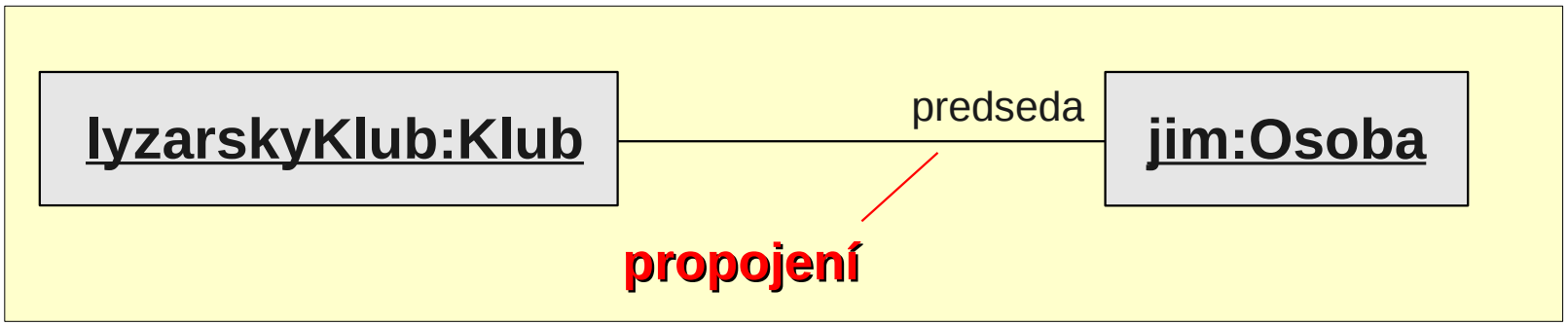




Asociace vs. propojení

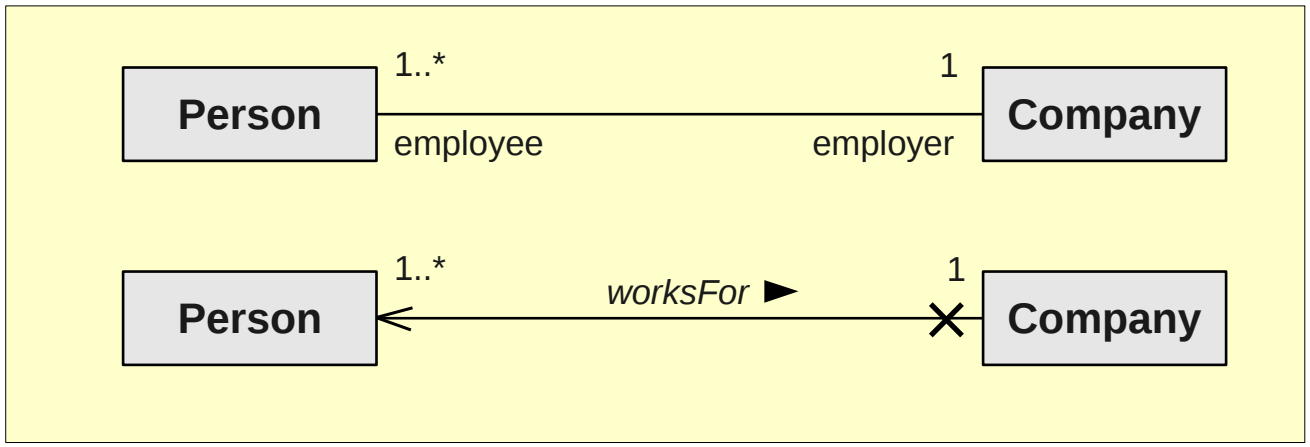
- Asociace je vztah mezi třídami (obdoba propojení u objektů)
- Pokud existuje asociace mezi třídami, lze vytvářet propojení mezi instancemi tříd na diagramu tříd
  - propojení mezi objekty nelze vytvořit (namodelovat), pokud neexistuje asociace na příslušném diagramu tříd





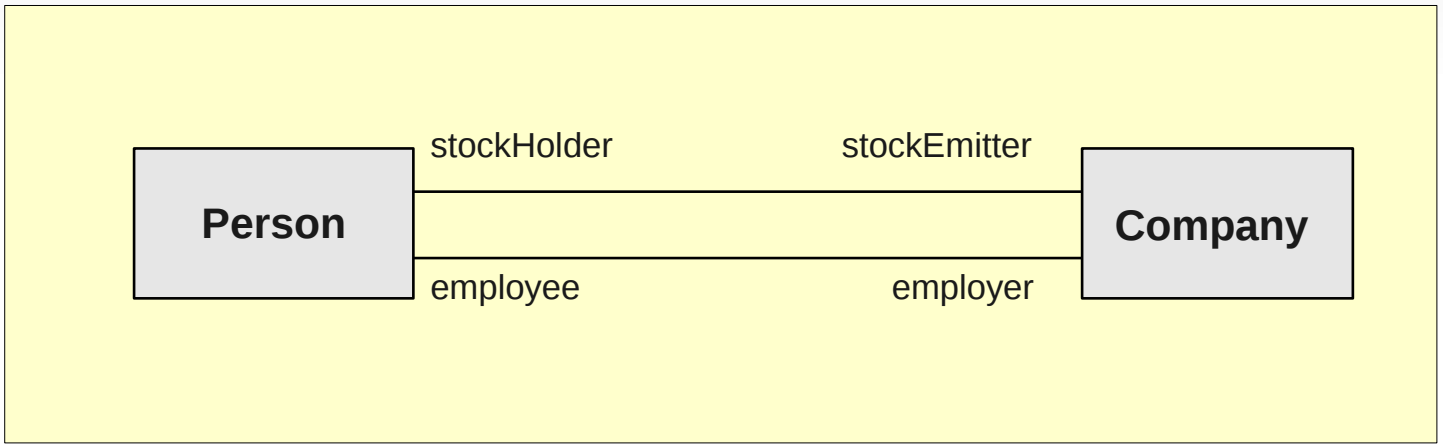
Asociace mezi třídami

- 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



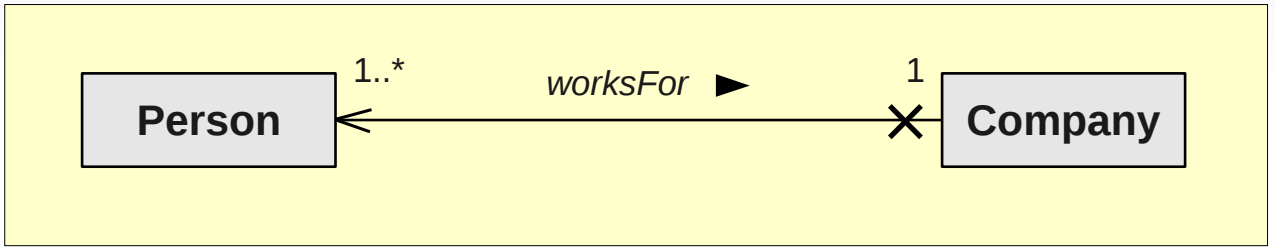
Násobné asociace

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



Směr asociací (směr řízení)

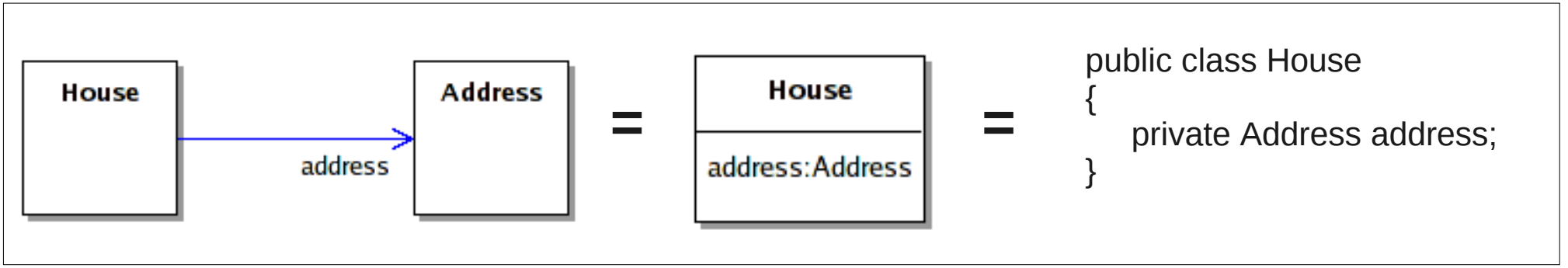
- I když je asociace jednosměrná, většinou je možné ji “procházet i v zakázaném směru” za cenu vyšší výpočetní náročnosti
- Př: Osoba neví nic o firmách, ve kterých je zaměstnána. Přesto můžeme firmy zaměstnávající osobu najít tak, že procházíme všechny firmy a hledáme ty, které zaměstnávají danou osobu.



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

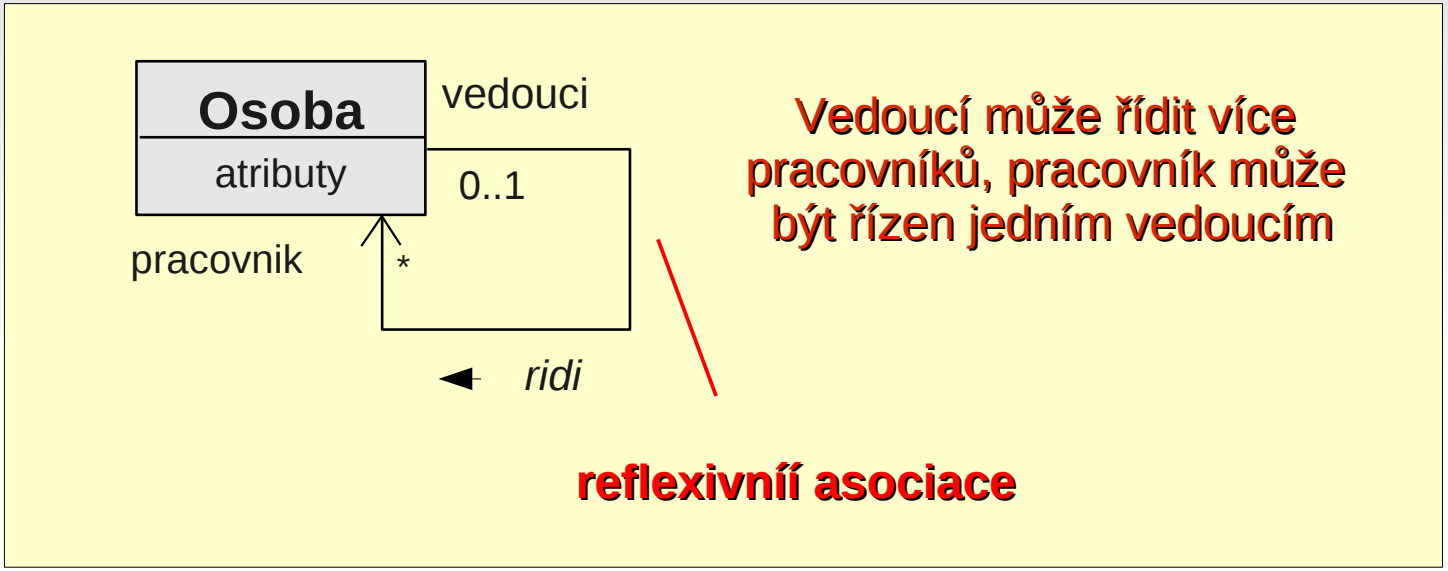
Asociace a atributy

- 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

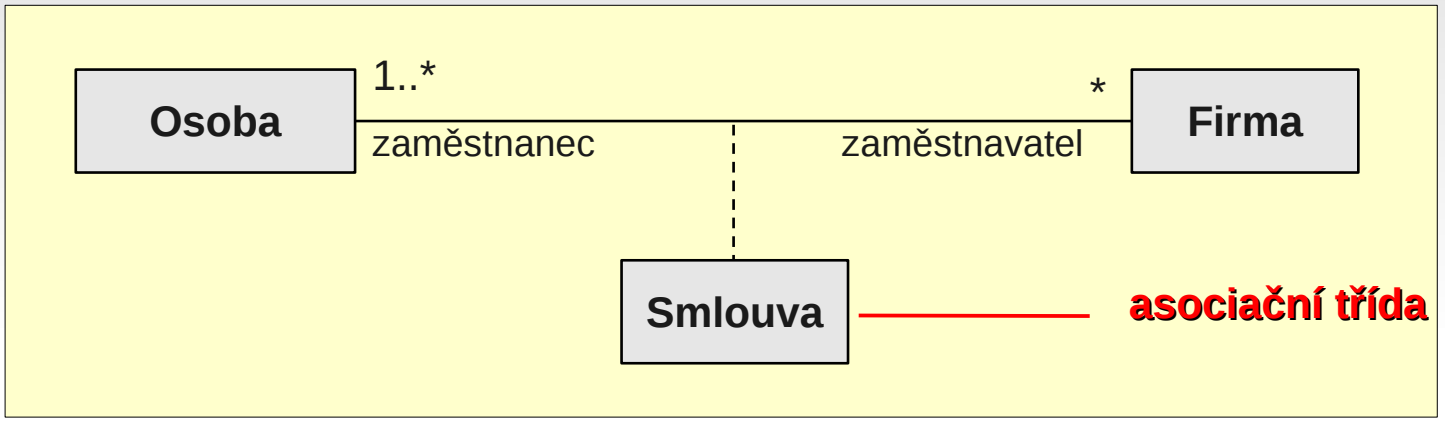
- Angl.: *Reflexive association*
- Měly by mít vždy role, jinak vzniká chaos



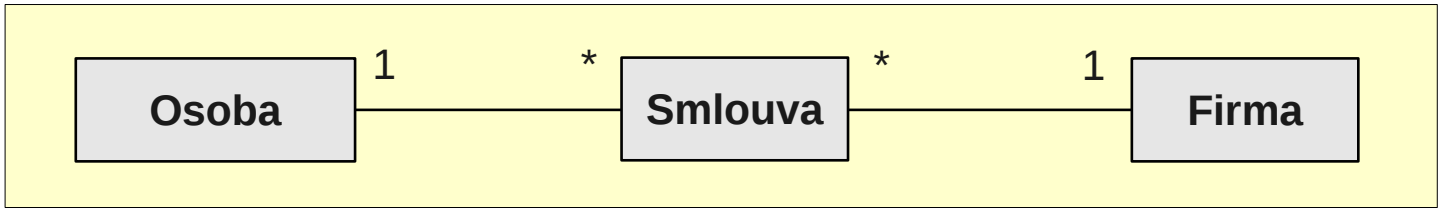
- Další “ozdoby” asociací:
  - » uspořádání: {ordered}
  - » měnitelnost: {addOnly}
  - » viditelnost: +, #, -
  - » navigace: směrové šipky
  - » omezení: {union}, {subset}, {unique}

Asociační třídy

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

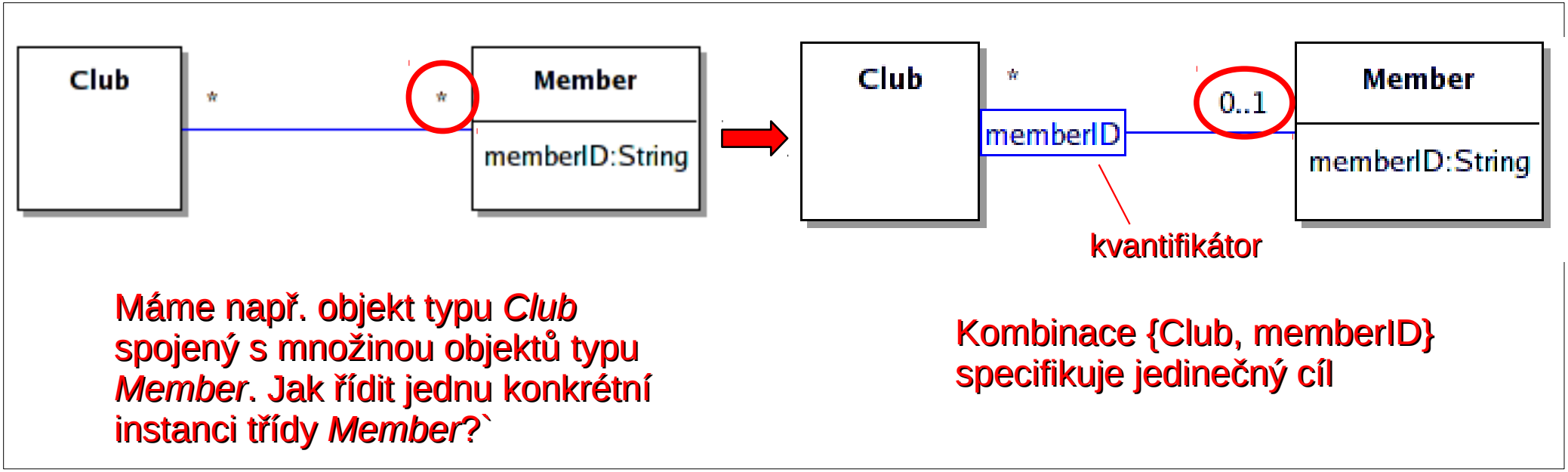


- Jedna osoba může mít s jednou firmou pouze jedinou smlouvu
  - může existovat pouze jediné propojení mezi objekty třídy Osoba a Firma v daném čase
- Pokud může být smluv více, tak:



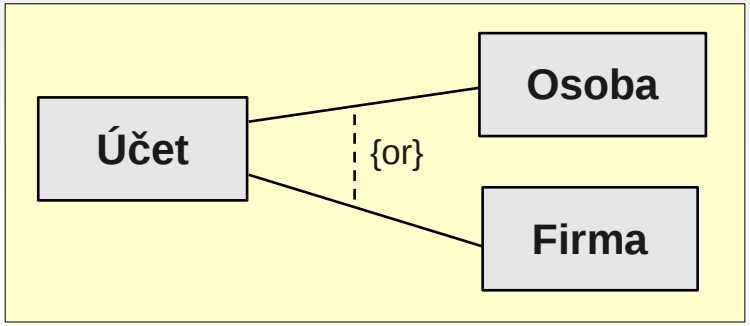
Asociace s kvantifikátorem

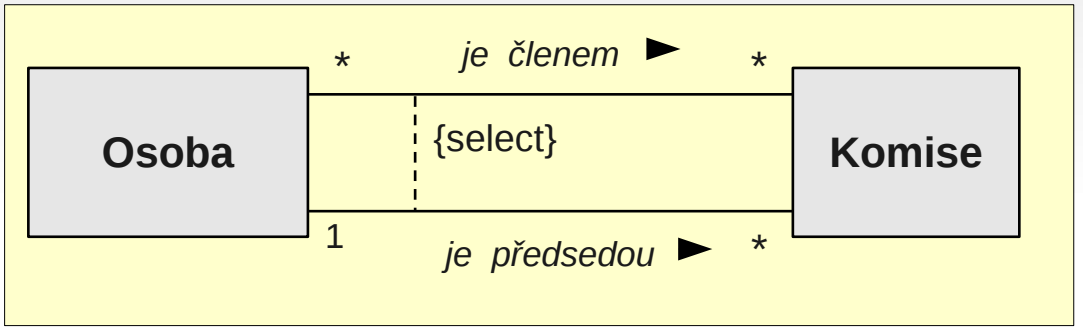
- Angl: *Quantified association*
- Redukce asociace typu M:N na typ N:1
  - slouží k výběru jednoho člena cílové množiny
  - představuje “vyhledávací klíč”
  - kvantifikátor se obvykle odkazuje na atribut cílové třídy
  - kvalifikátor je součástí asociace, nikoliv třídy!

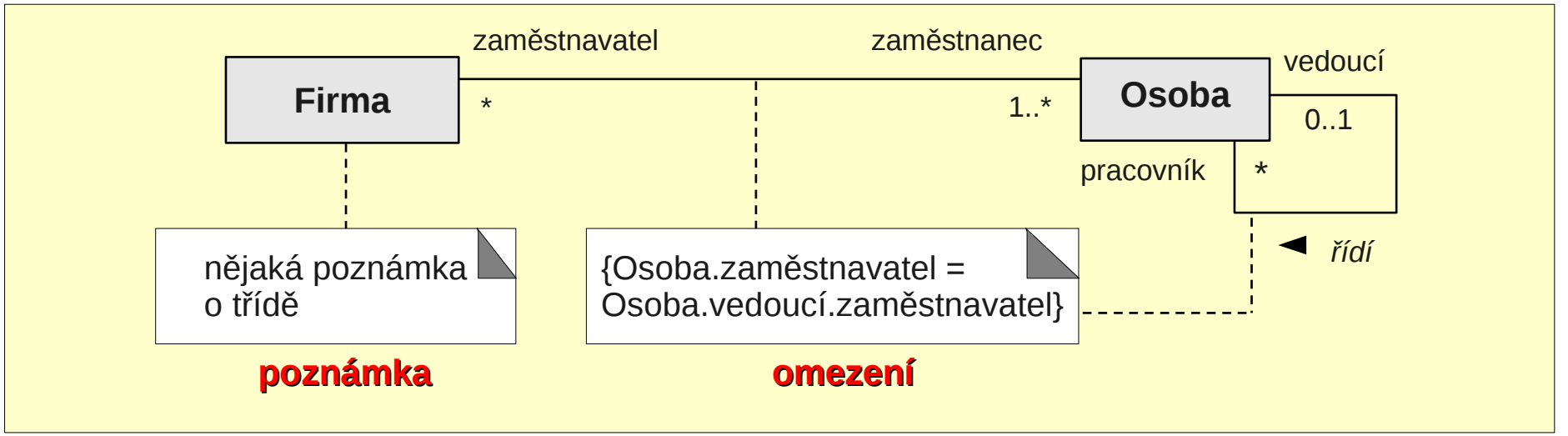


Omezující podmínky na asociacích

- 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 {}

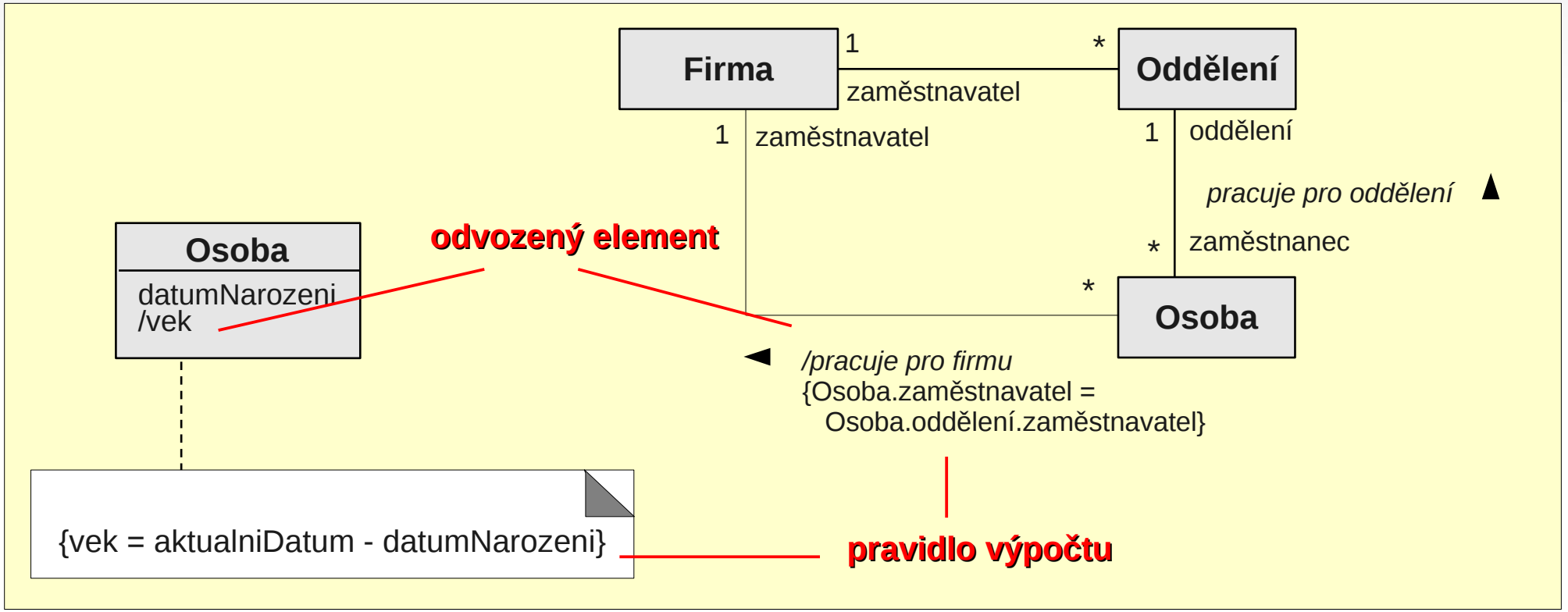






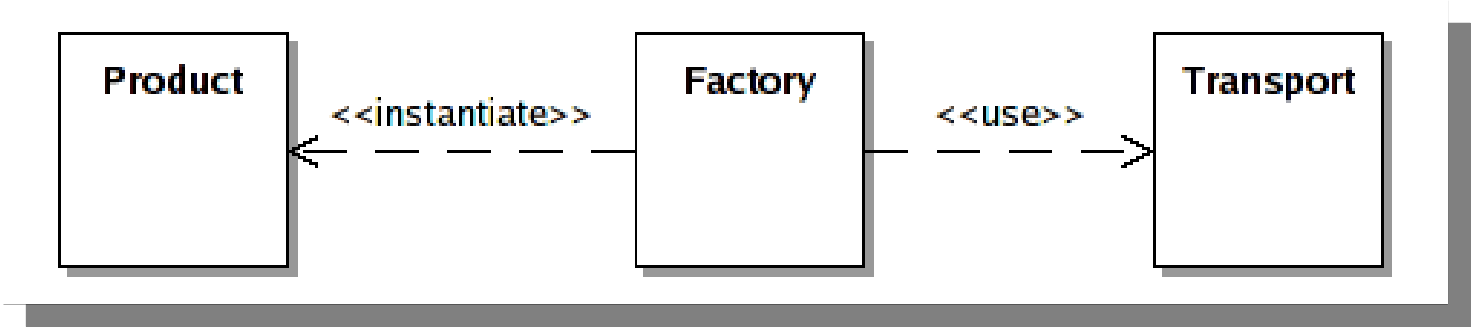
Odvozené atributy a asociace

- 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.

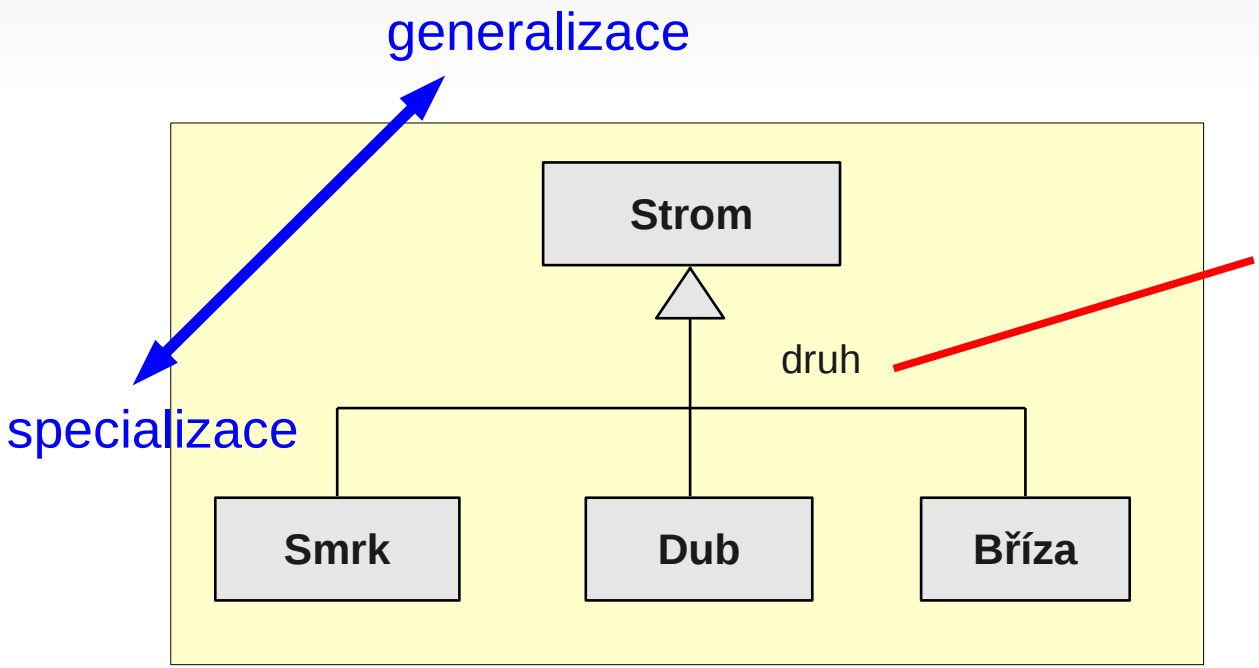


Některé předdefinované stereotypy:

- 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

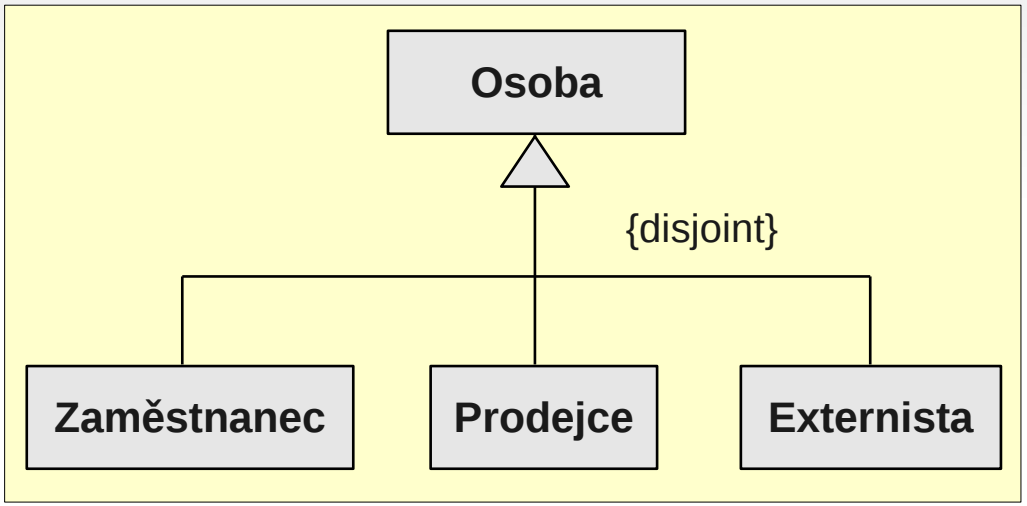
- Generalizace je vztah mezi obecným a specifičtějším elementem (třídy, případy užití, rozhraní), který:
  - je **plně konzistentní** s obecným elementem
  - a přidává **dodatečnou informaci**



diskriminínátor
- nepovinný
- určuje, podle které charakteristiky nadtřídy je prováděna abstrakce
- každá podtřída má jedinečnou hodnotu diskriminátoru

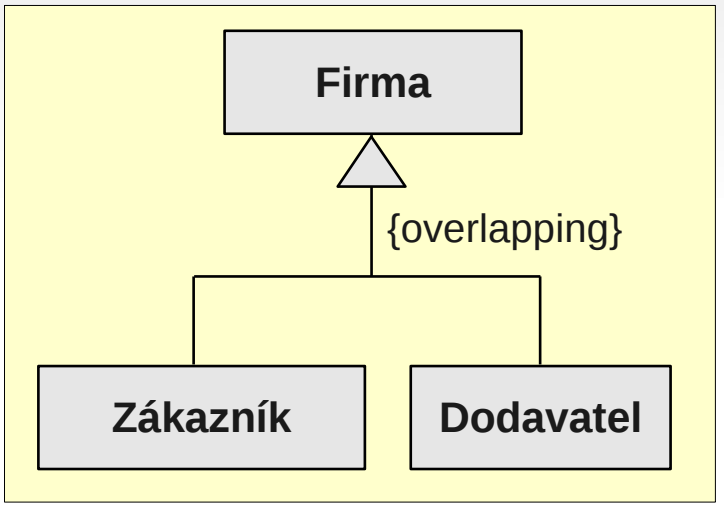
Generalizace - ozdůbky

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



výlučná specializace

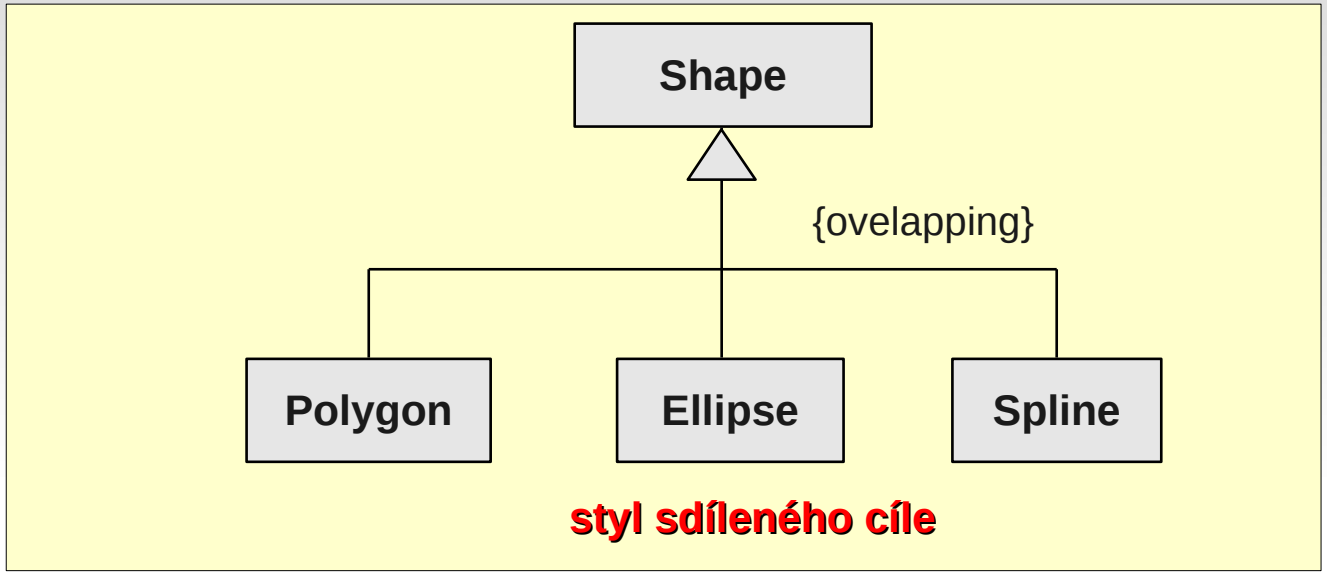
Osoba je:
- buď zaměstnanec, nebo prodejce, nebo externista.

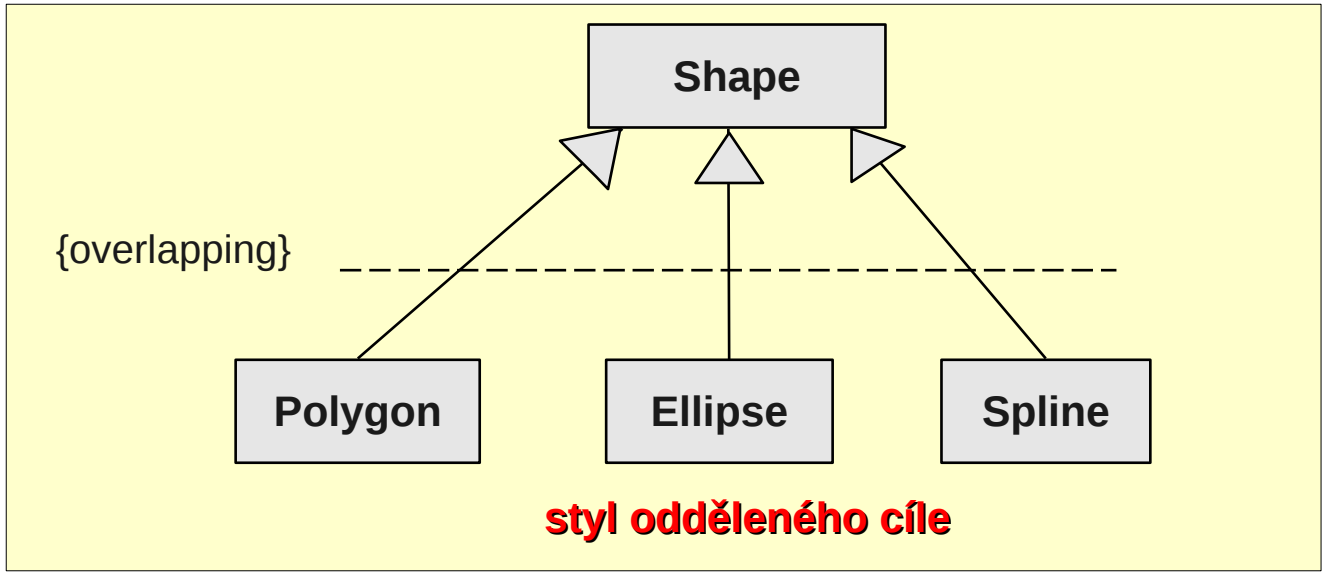


nevýlučná specializace

Firma je:
- zákazníkem, nebo dodavatelem, nebo obojím.

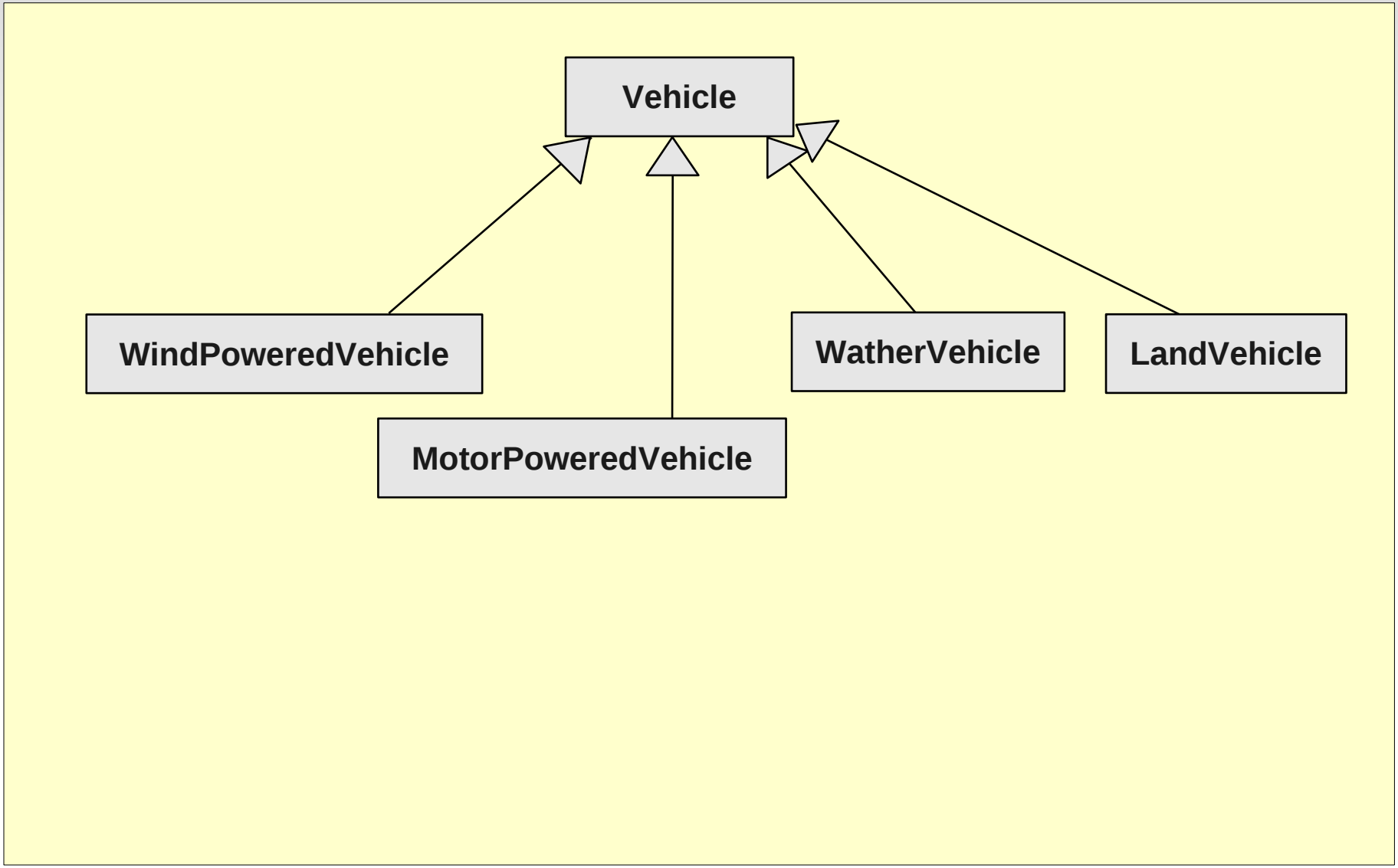
Generalizace – styl zápisu



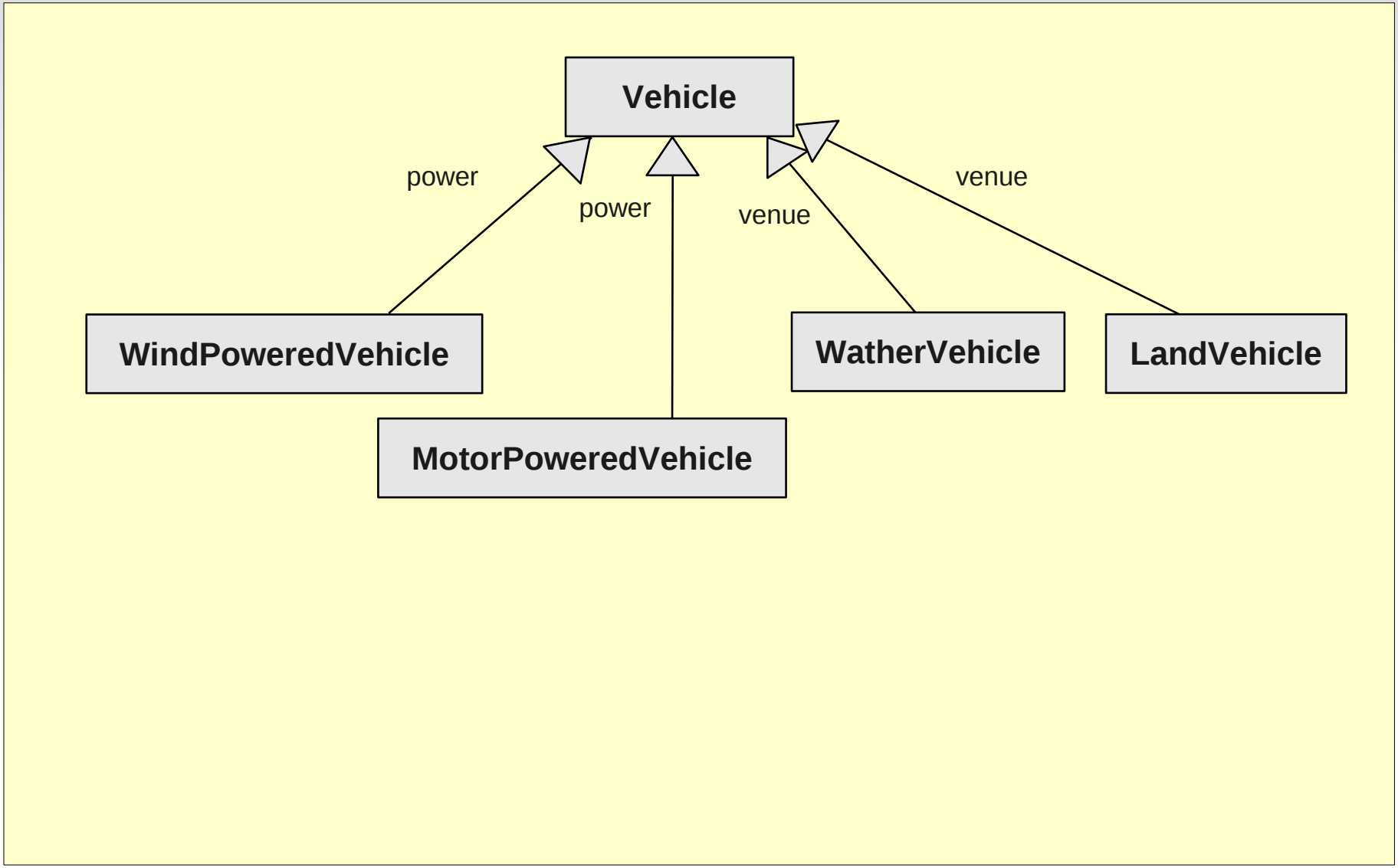


Příklad „zneužití“ dědičnosti (I)

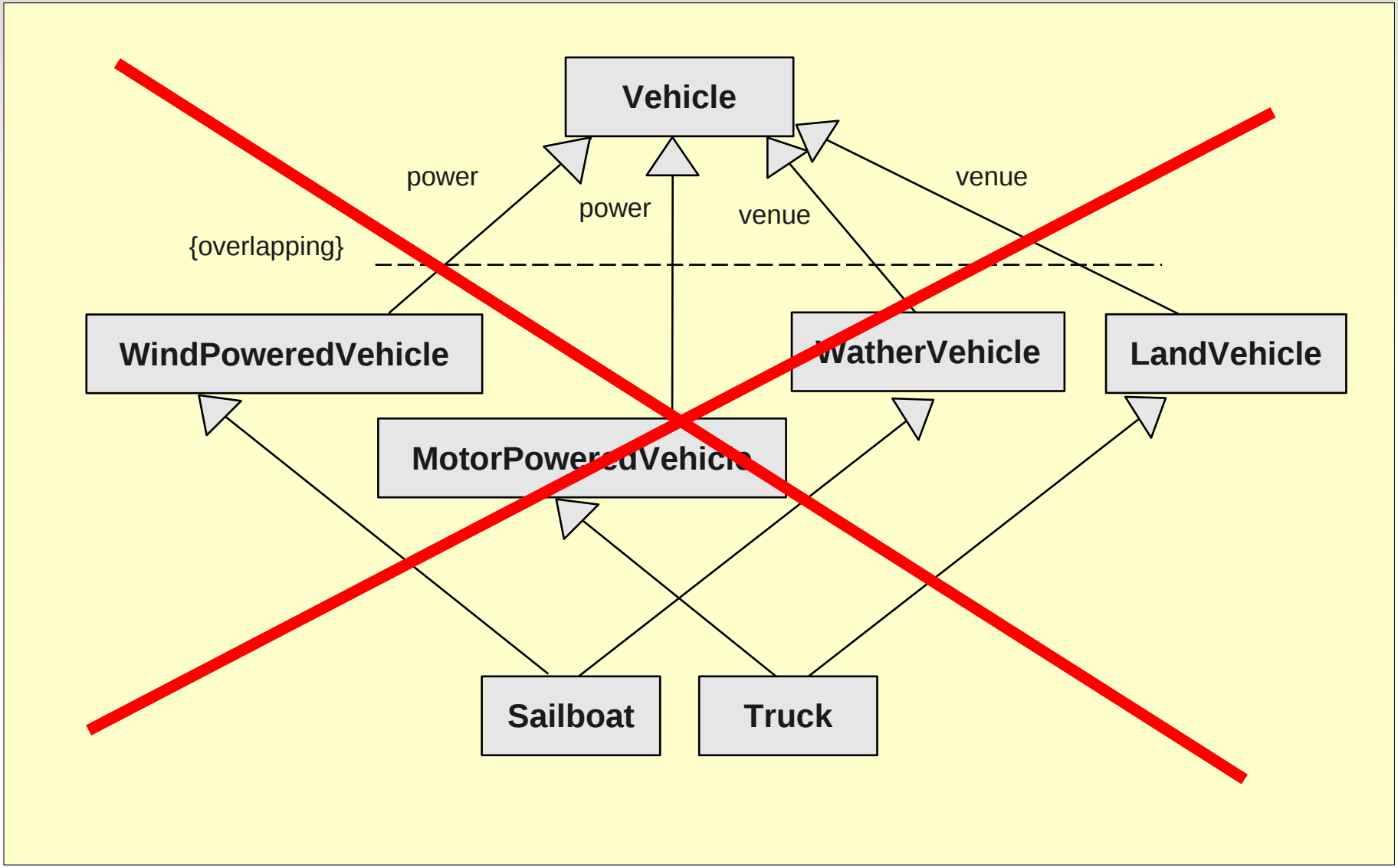
- Příklad chybné volby abstrakce



Příklad „zneužití“ dědičnosti (II)



Příklad „zneužití“ dědičnosti (III)

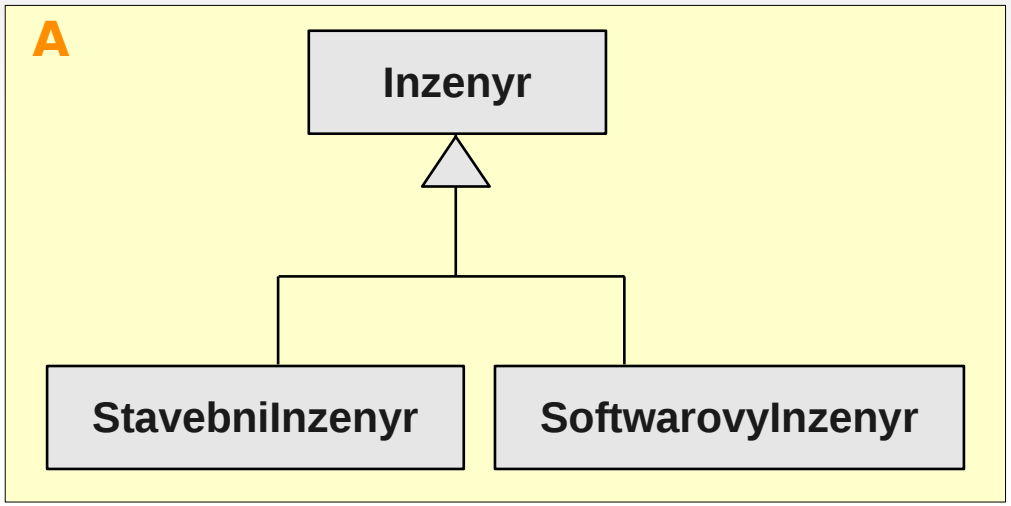


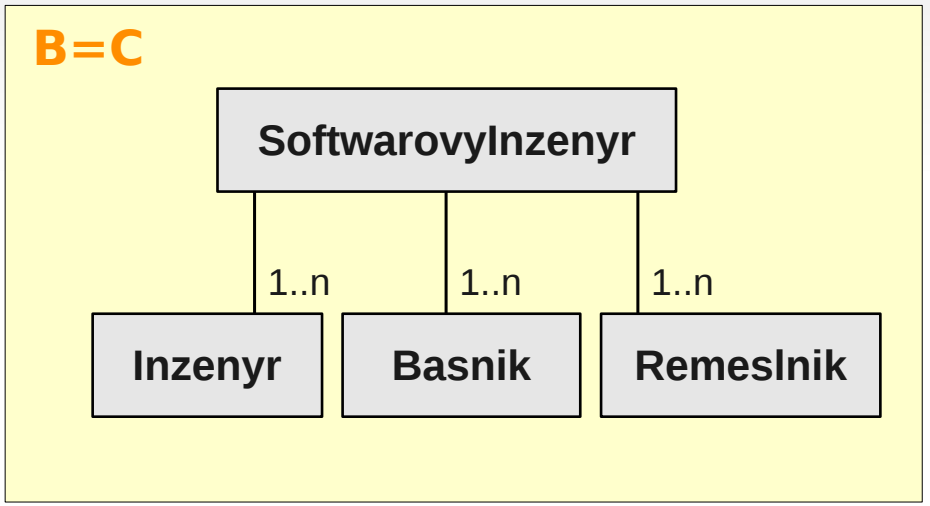
„Would you rather buy or inherit ?“

A. Každý softwarový inženýr je inženýr.

B. V každém softwarovém inženýrovi je inženýr.

C. Každý softwarový inženýr má inženýrskou komponentu.





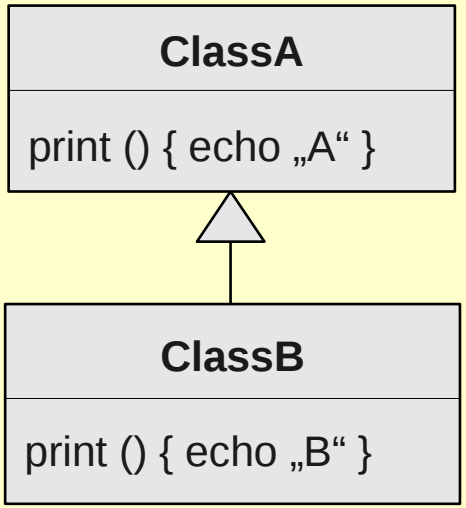
Nepoužívejte dědičnost pro popis vnímaného vztahu „je něčím“,
pokud odpovídající objektové komponenty se
mohou změnit během provozu (běhu).

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

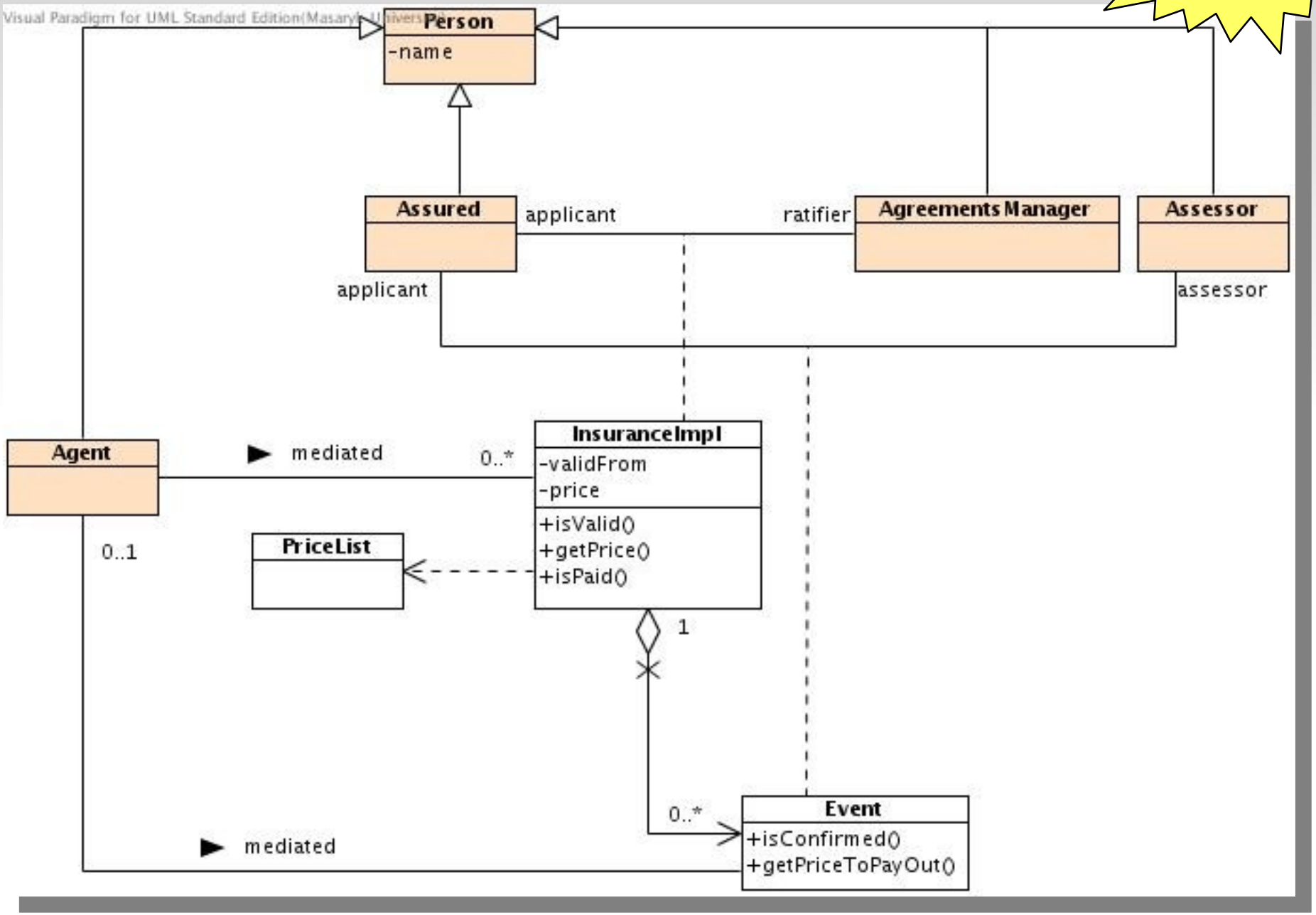
Co vypíše následující program?

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



Pojišťovna: Analytický model

Demo

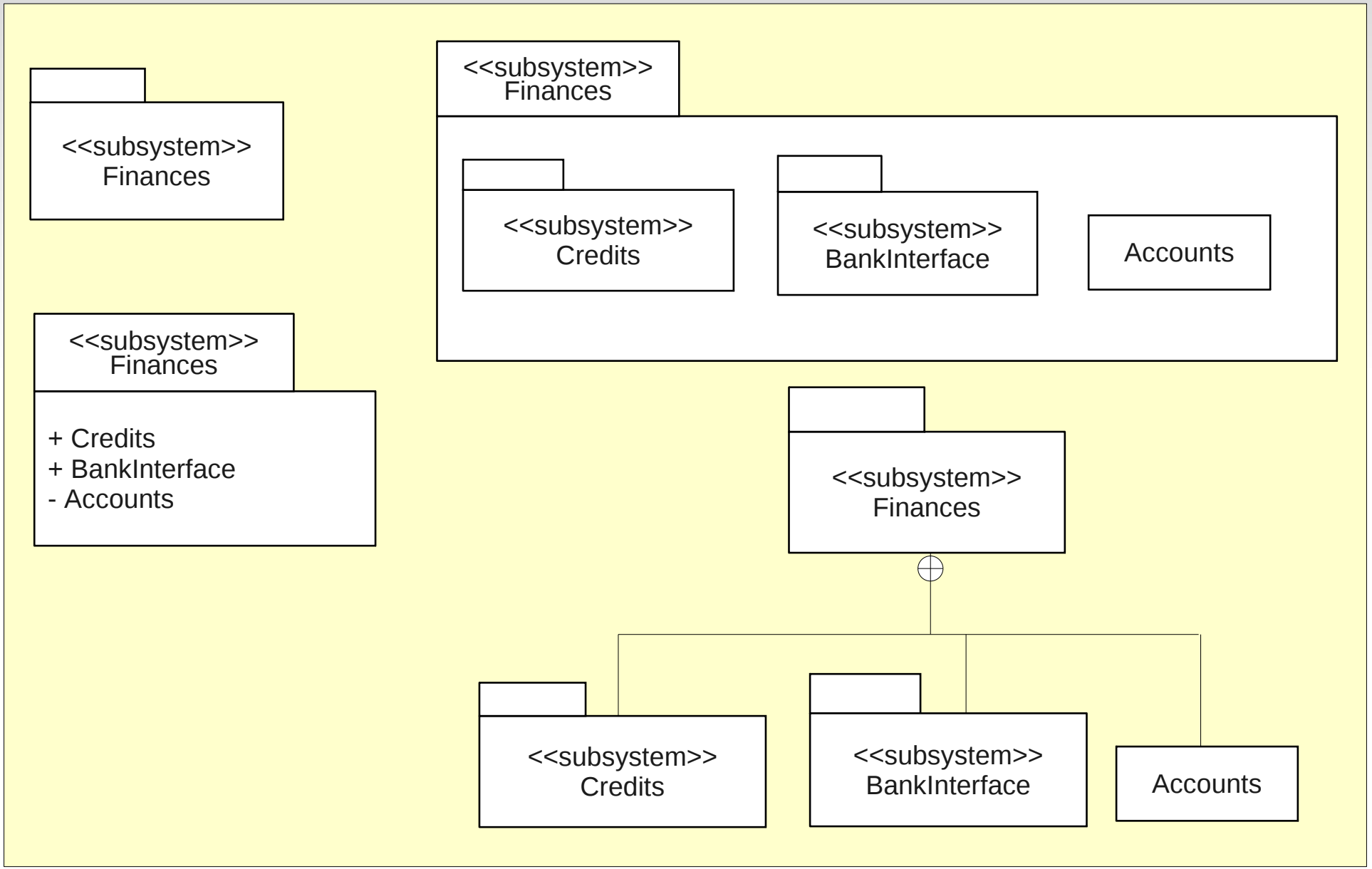


Analytické balíky

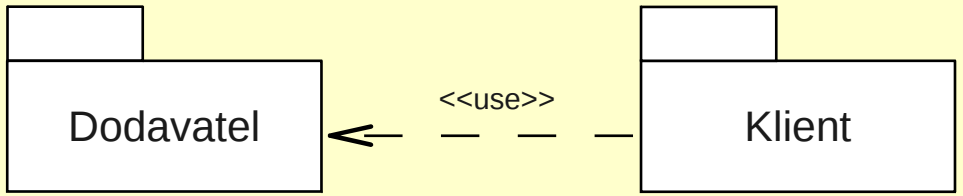
Balíky

- 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.

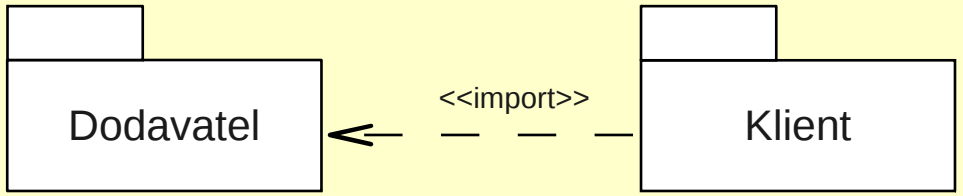
Balíky – notace UML



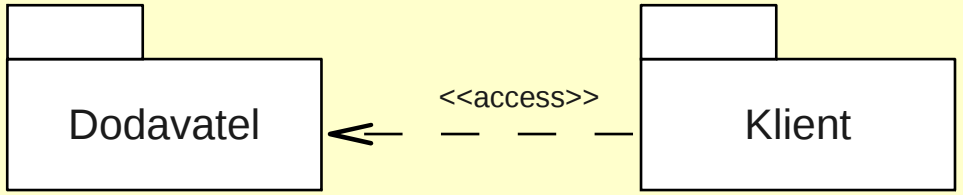
Balíky - závislosti



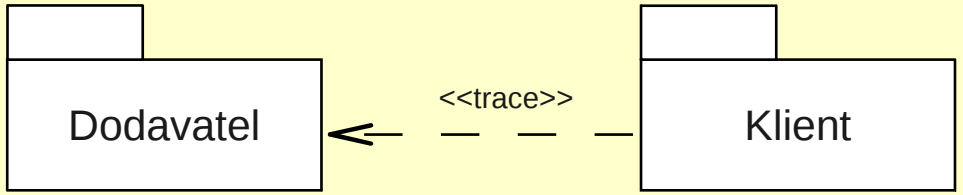
Některý element klienta závisí na některém elementu z dodavatele.
Totéž co bez stereotypu.
Vhodné pro analýzu, v návrhu se upřesní <<import>> nebo <<access>>



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



<<use>> + jmenné prostory zůstávají oddělené.

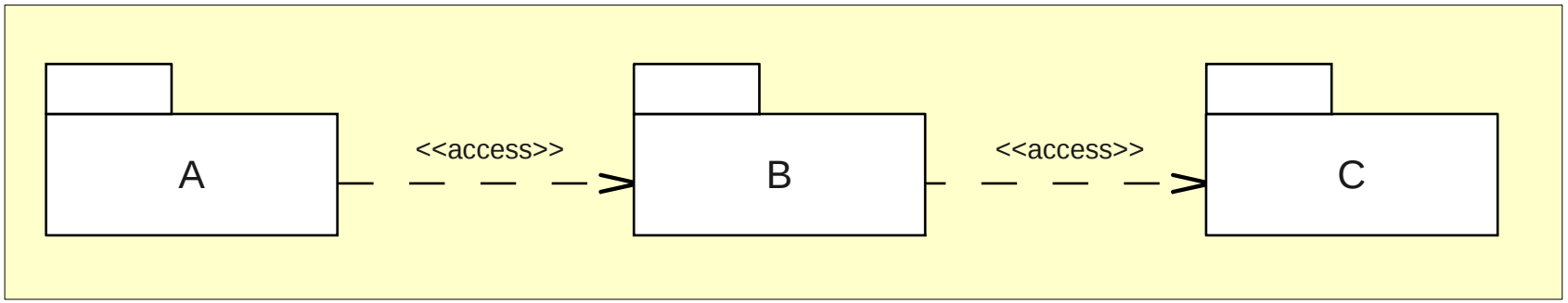


Klient je dalším vývojovým stupněm dodavatele (v průběhu analýzy se balík *Dodavatel* změnil na *Klient*)

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.

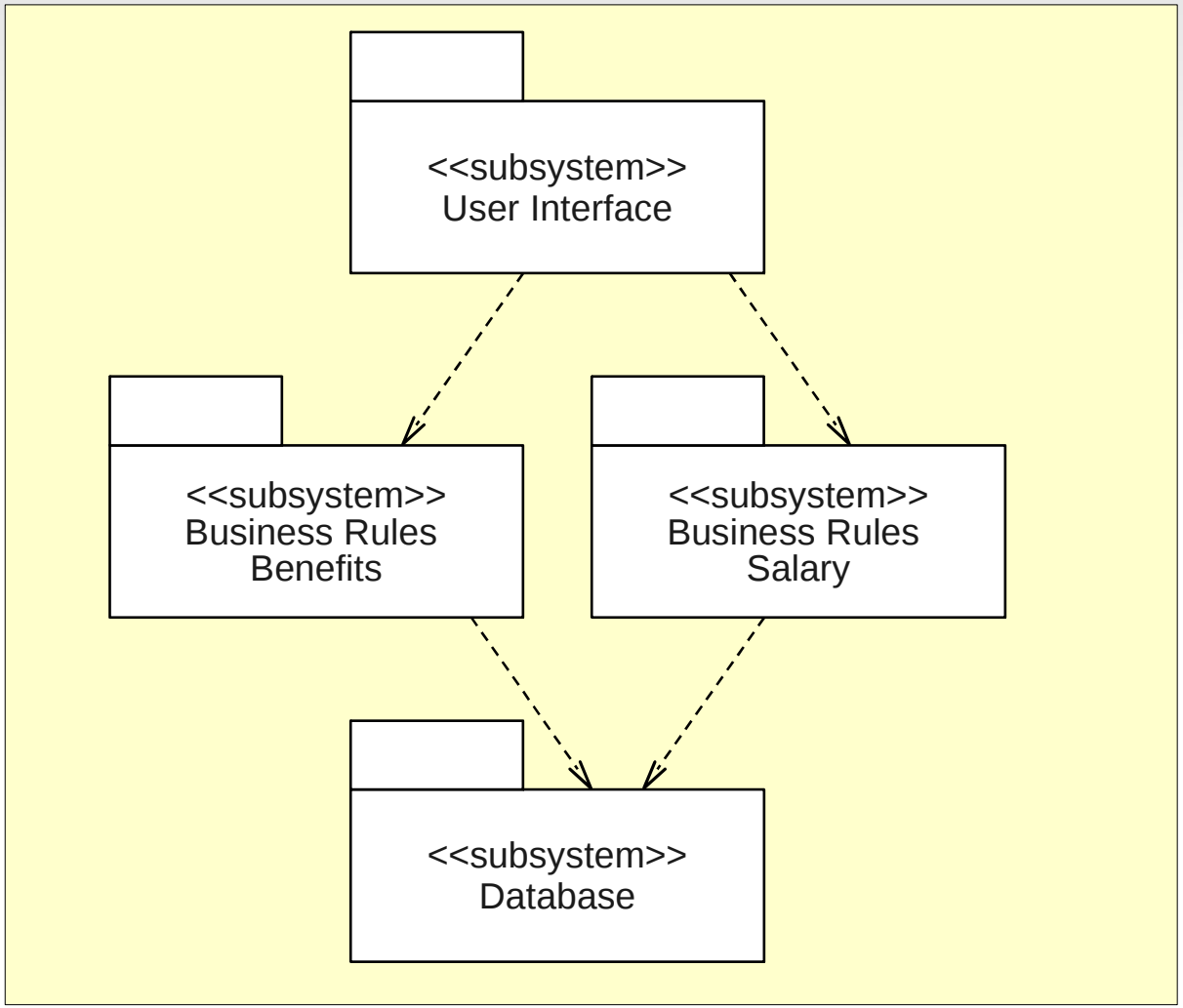
=> soudržnost modelu, jasnější zodpovědnost



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

Balíky – závislosti (III)

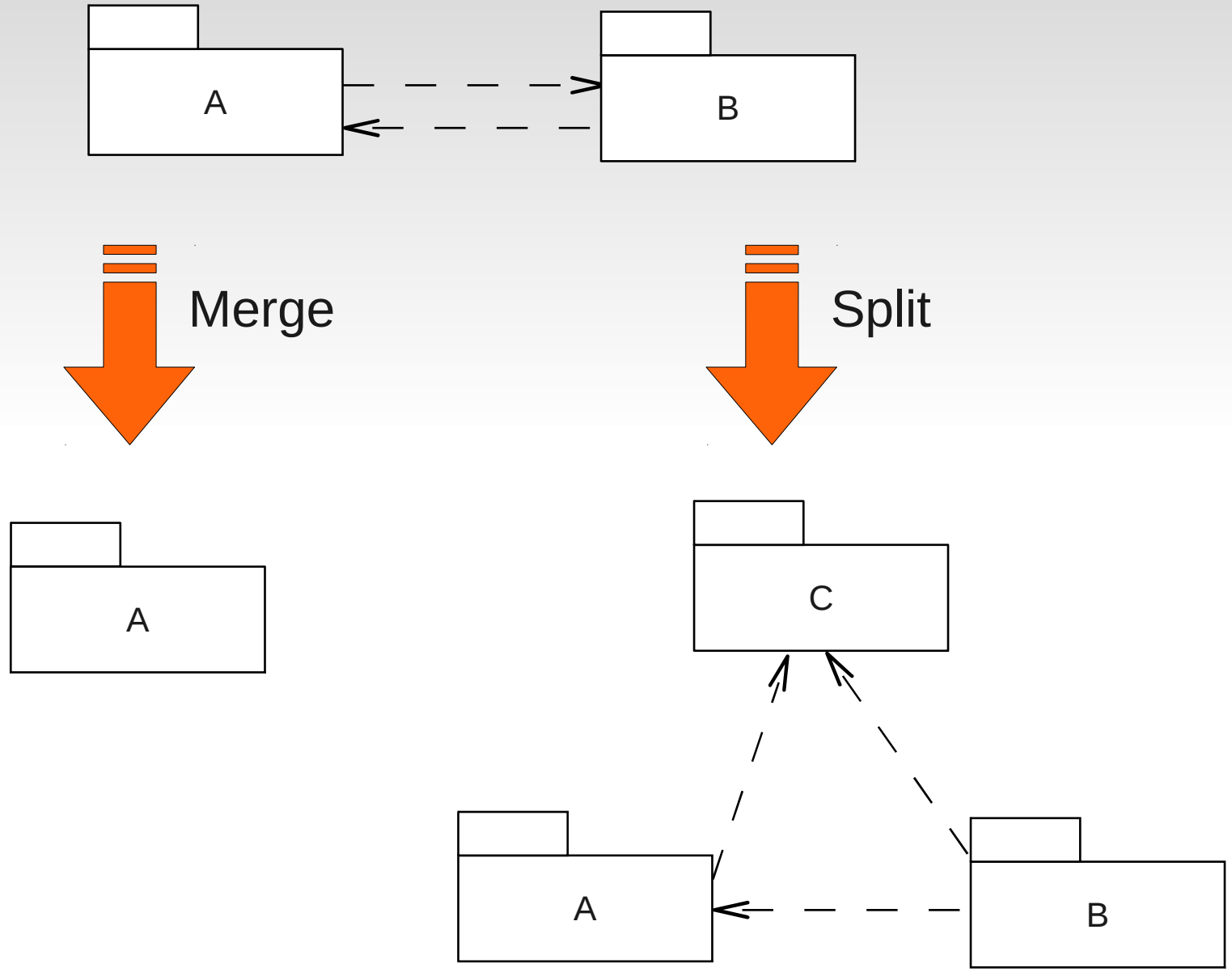
Příklad diagramu architektury ukazující subsystémy a jejich závislosti



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

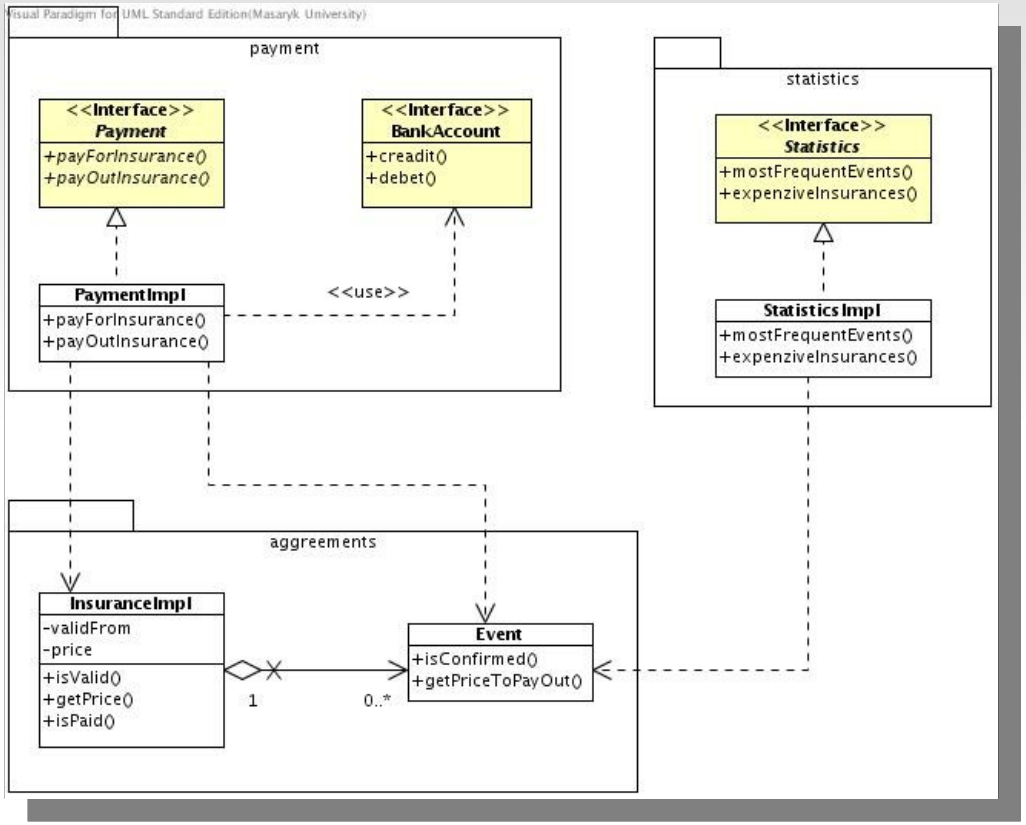
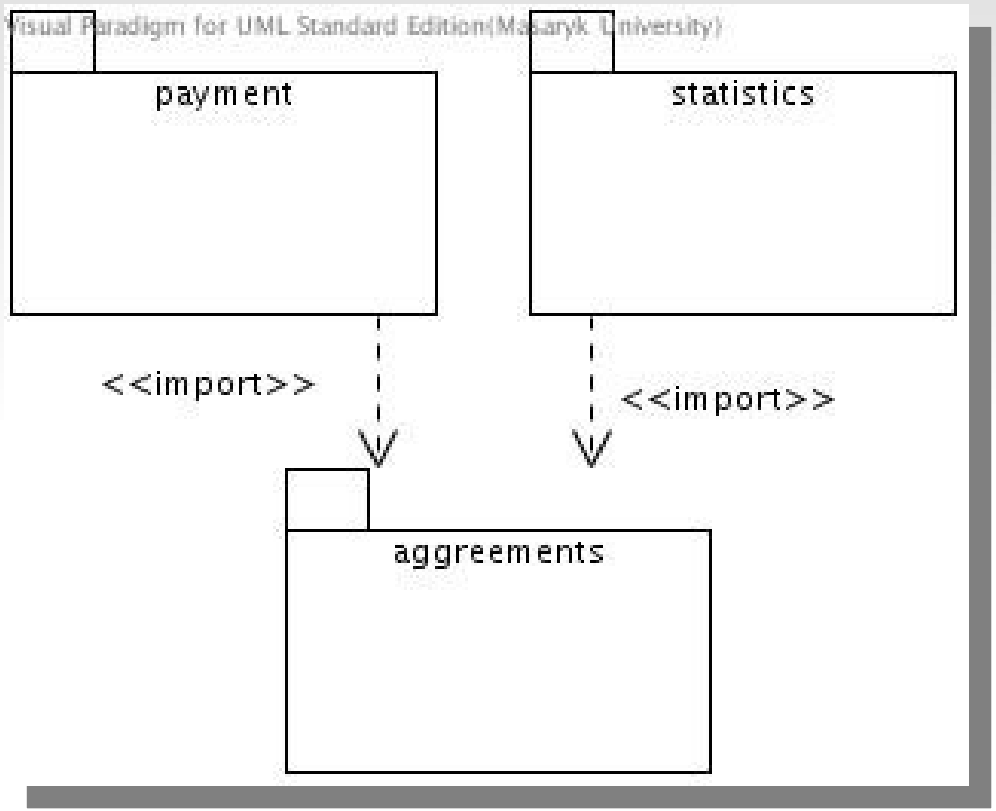


Diagram tříd včetně balíků



Přehlednější diagram balíků