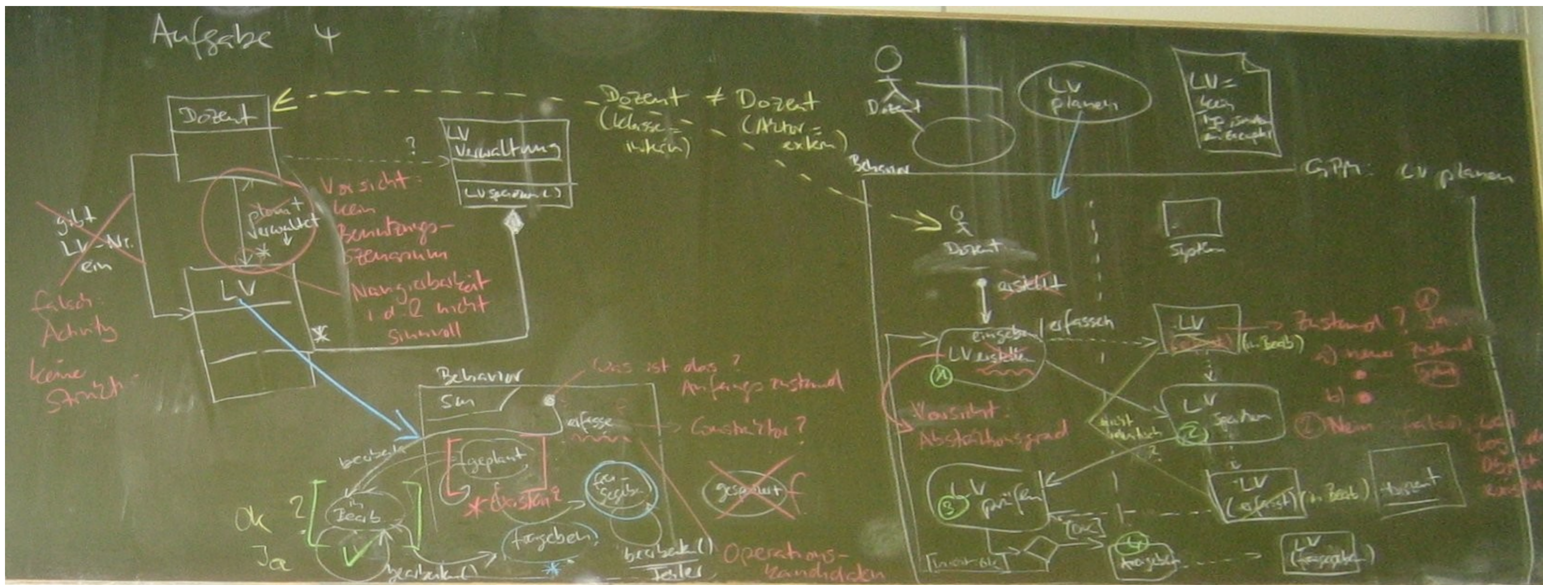

Aufgabe 4
Dozent
LV
Behavior
Tabelle
Activity
Klasse + Methode(n)
State change
LV erfassen
LV speichern
LV prüfen
LV freigeben
titel : String
public LV ( ? )

# UML (2)

## diagramy popisující statickou strukturu systému

**© 2008 Radek Ošlejšek**

**FI MU Brno**

oslejsek@fi.muni.cz
http://www.fi.muni.cz/~oslejsek/PA103

# **Diagram tříd**
# *(Class Diagram)*

# Typ vs. instance

- **Diagram tříd** *(Class Diagram, CD)* je grafický pohled na statickou strukturu
  - modeluje třídy (typy) a jejich vztahy
- **Diagram objektů** je instancí diagramu tříd
  - ukazuje snímek systému v určitém časovém bodě
- **Typ – instance:**
  - třída – objekt
  - asociace – spoj
  - parametr – hodnota
  - operace - volání

# Třídy

*Window*

**potlačené detaily**

| *Window* |
| --- |
| size: Area<br>visibility: Boolean |
| *display ()*<br>*hide ()* |

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

| ***Window***<br>{abstract,<br>author=Joe,<br>status=tested} |
| --- |
| +size:Area = (100,100)<br>#visibility:Boolean = invisible<br>+default-size:Rectangle<br>#maximum-size:Rectangle<br>-xptr: XWindow*<br>+name:String {frozen}<br>-colors[3]: Color |
| *+display ()*<br>*+hide ()*<br>*+create ()*<br>-attachXWindow(xwin:Xwindow*) |

**detaily implementační úrovně**

# Třídy (II)

## Struktura atributů:

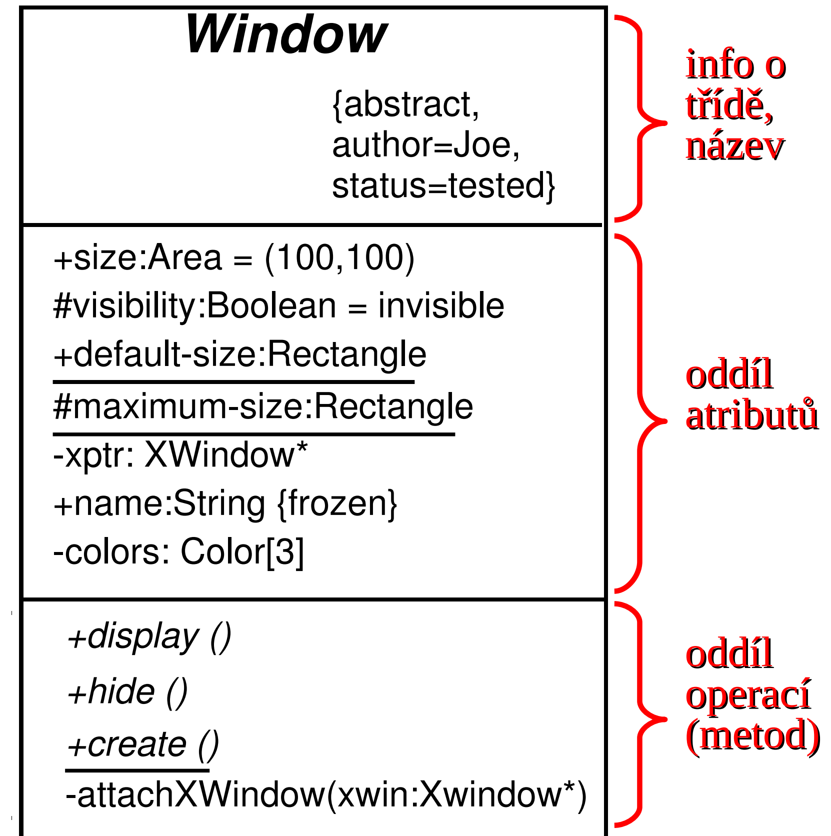
viditelnost název: typ [násobnost] = počátečníHodnota

nepovinné povinné nepovinné

## Struktura operací:

viditelnost název(názevArg: typArg, ...): návratovýTyp

signatura operace




## Abstraktní třída:

{abstract}

## Násobnost:

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

## Viditelnost:

+ public
- private
# protected

## Nezměnitelné:

{frozen}

# 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**

generalizace

Strom

druh

specializace

Smrk

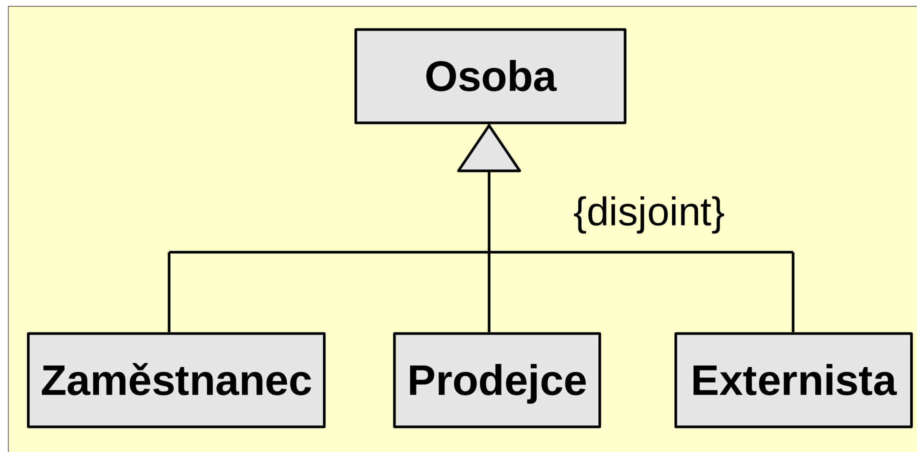
Dub

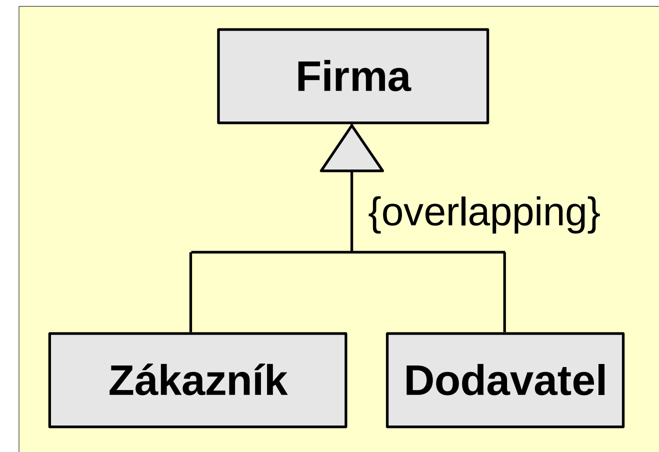
Bříza

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

# Generalizace - ozdůbky

- Ozdůbky:
  - » {overlapping} nebo {disjoint}
  - » {incomplete} nebo {complete}



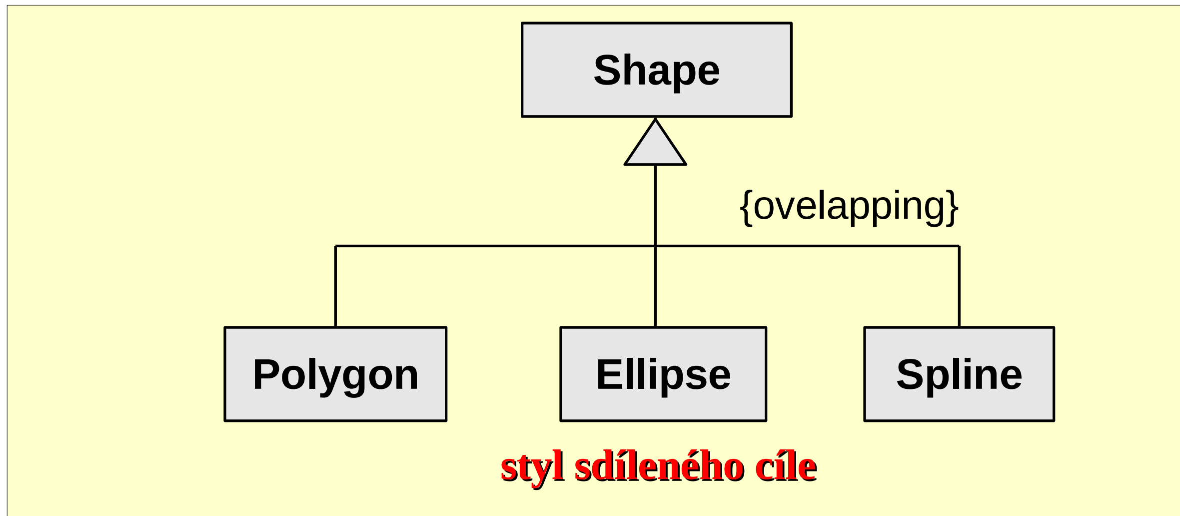

**výlučná specializace**

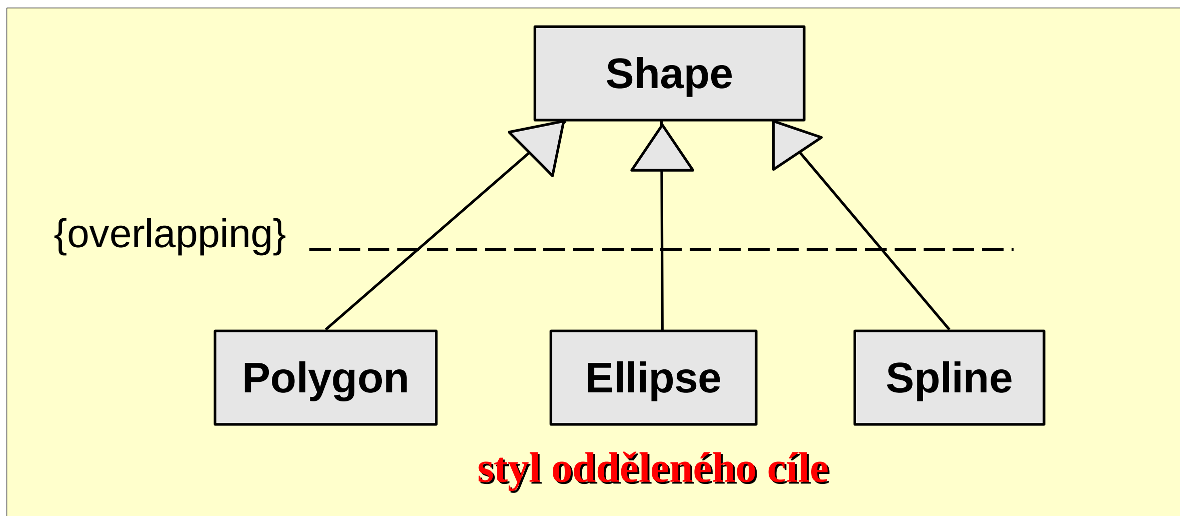
**nevýlučná specializace**

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

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

# Generalizace – styl zápisu



# Příklad „zneužití“ dědičnosti (I)

Vehicle

WindPoweredVehicle

MotorPoweredVehicle

WatherVehicle

LandVehicle

# Příklad „zneužití“ dědičnosti (II)

Vehicle

power

power

venue

venue

WindPoweredVehicle

MotorPoweredVehicle

WatherVehicle

LandVehicle

# Příklad „zneužití“ dědičnosti (III)

Vehicle

power

power

venue

venue

{overlapping}

WindPoweredVehicle

WatherVehicle

LandVehicle

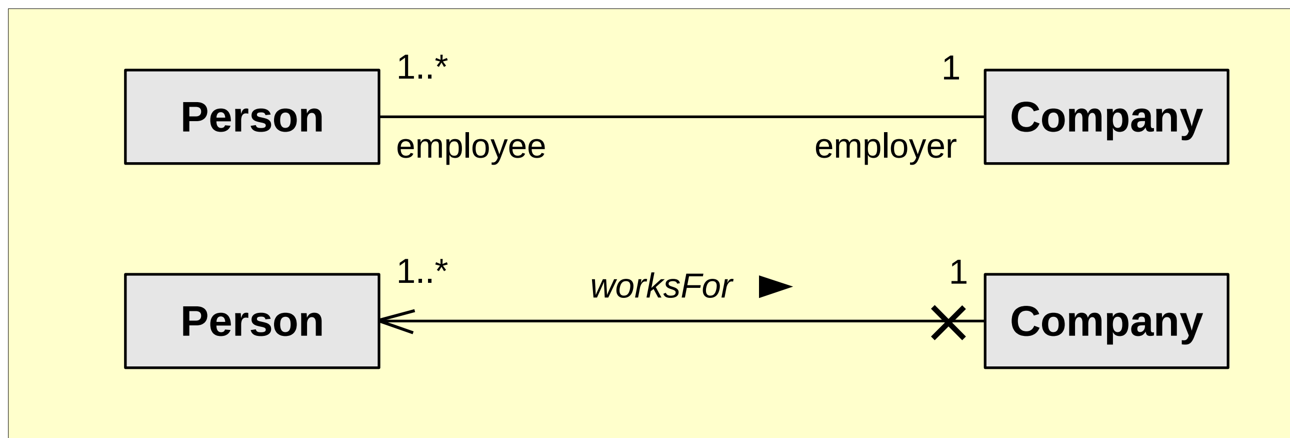
MotorPoweredVehicle

Sailboat

Truck

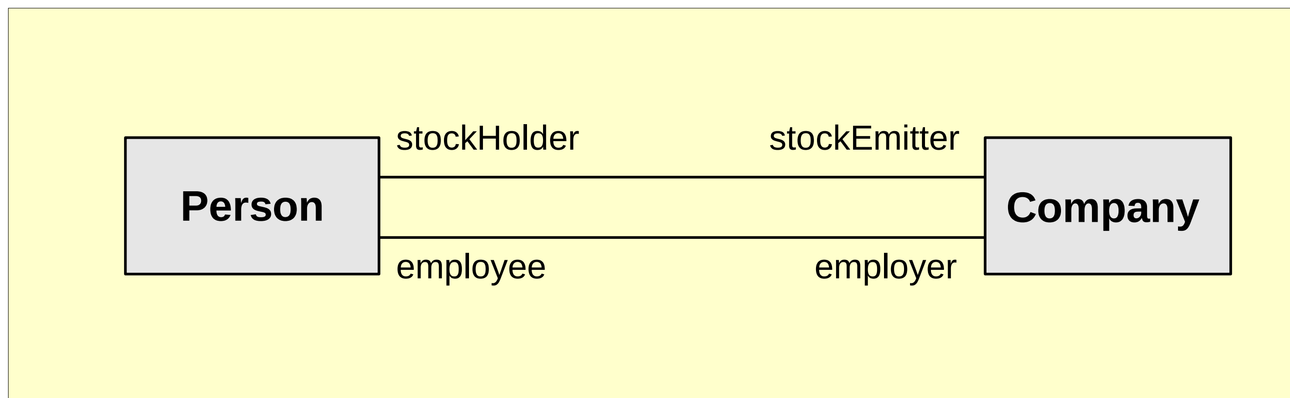
# Asociace mezi třídami

- Zobrazují se jako plná čára mezi dvěma třídami
- Měly by být pojmenovány, šipka u jména implikuje 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 statické asociace



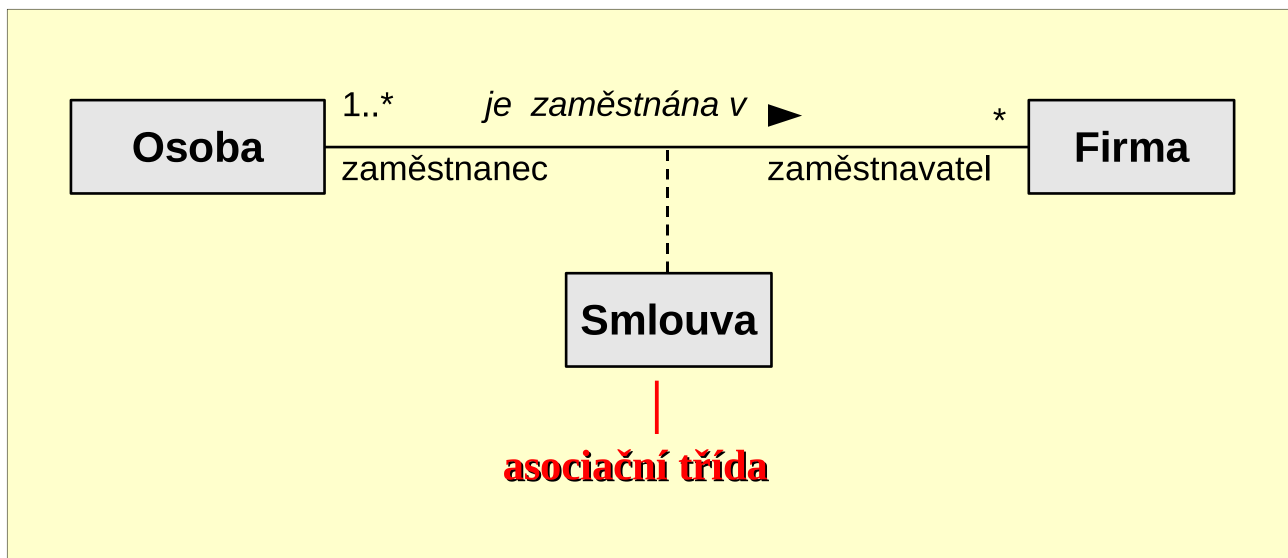
# Násobné asociace

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



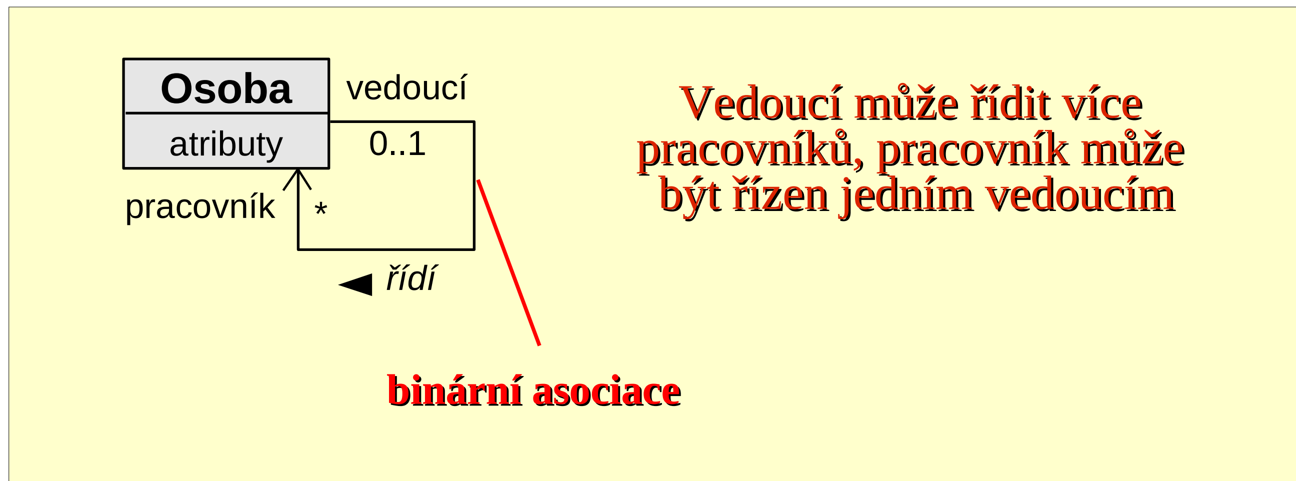
# Asociační třídy

- Angl: *Association class*
- Asociační třída vzniká jako produkt vztahu
- Uchovává dodatečné informace o vzniklém propojení
- Používá se často během analýzy, v návrhu se dekomponuje



# Reflexivní asociace

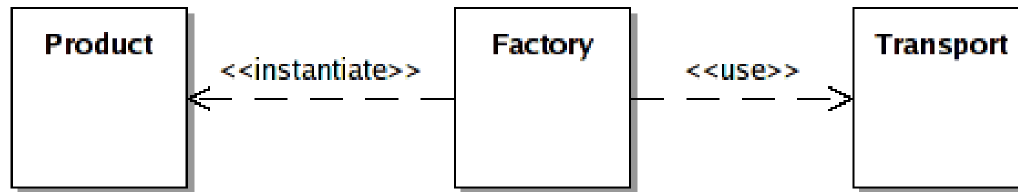
- Angl.: *Reflexive association*
- Měly by mít vždy role, jinak vzniká chaos




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

# 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.




**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>>

# Pojišťovna: Analytický model

Demo

Visual Paradigm for UML Standard Edition(Masaryk University)

**Person**
-name

**Assured**
applicant
applicant

ratifier
**Agreements Manager**

**Assessor**
assessor

**Agent**
▶ mediated
0..*
0..1

**InsuranceImpl**
-validFrom
-price
+isValid()
+getPrice()
+isPaid()

**PriceList**

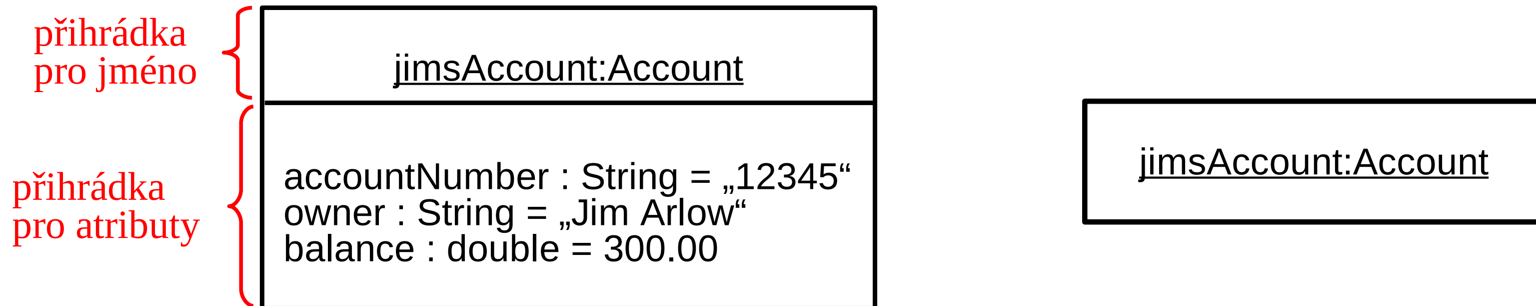
1
0..*

**Event**
+isConfirmed()
+getPriceToPayOut()

▶ mediated

# **Diagram objektů**
*(Object Diagram)*

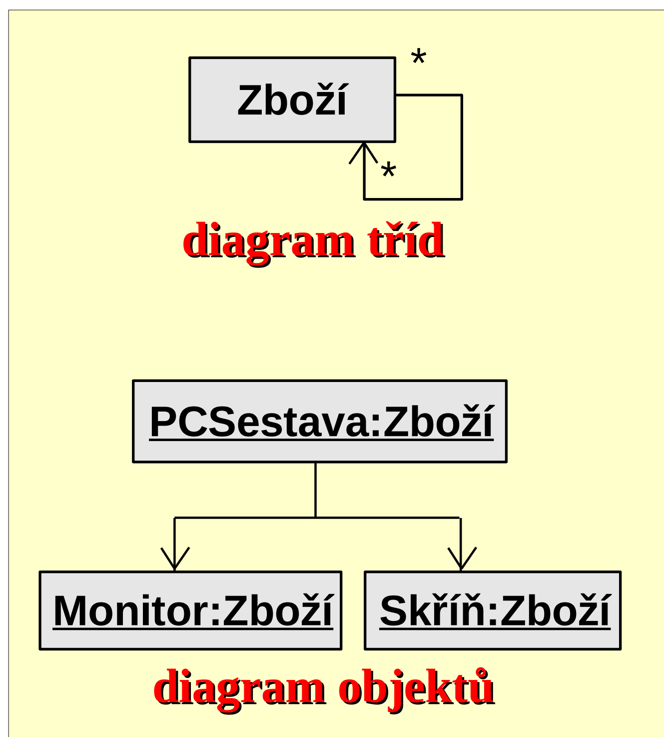
# Objekty

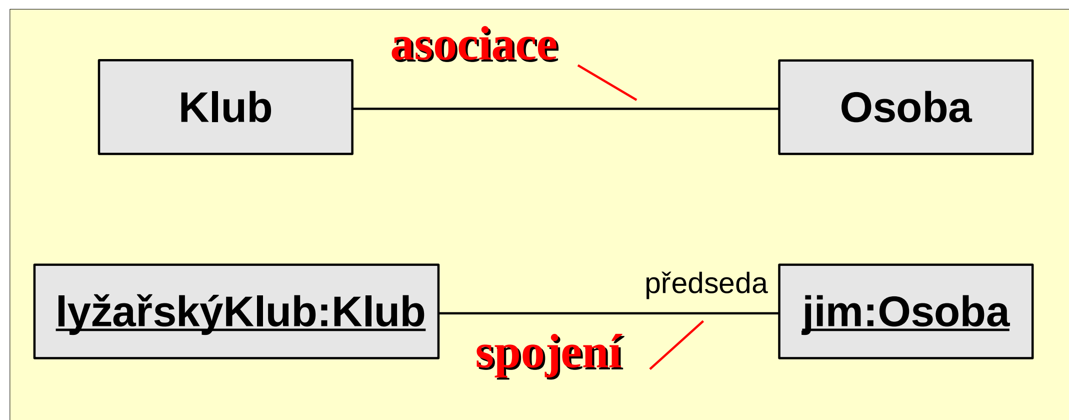


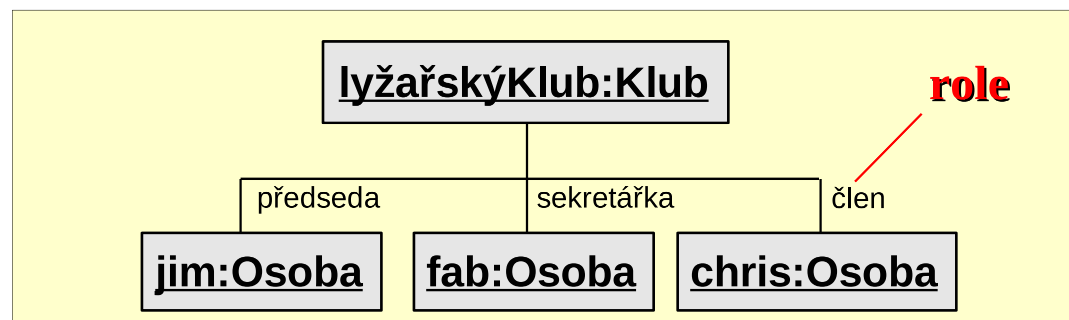

- *Syntaxe názvu: jméno objektu : jméno třídy [jména stavů]*
  - :Account – anonymní objekt dané třídy
  - jimsAccount – konkrétní objekt bez specifikované třídy, užitečné především v počátečních fázích analýzy
  - jimsAccount:Account – konkrétní instance třídy; nutné pokud je v diagramu více instancí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

# Diagram objektů

- Snímek v časovém bodě, který ukazuje podmnožinu objektů, jejich stavů, hodnot atributů a propojení.
- Místo o *asociaci* mluvíme o *spojení*
- Vhodné zejména pro komunikaci se zákazníky (konkrétní příklad hierarchie, ...)
- Syntaxe: *jméno objektu* : *jméno třídy* [*jména stavů*]



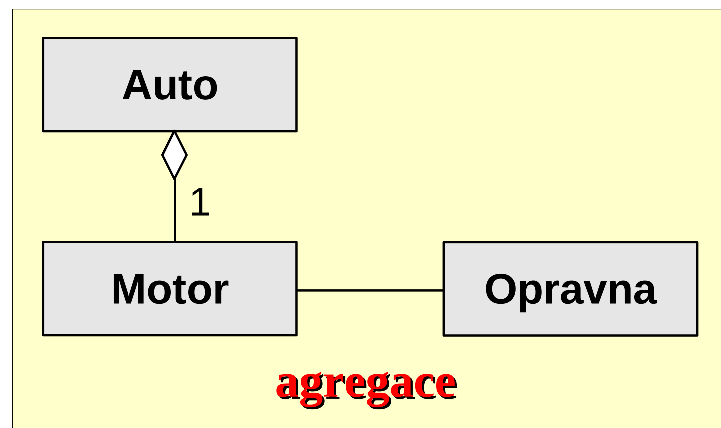



# Pokročilé modelování v diagramu tříd

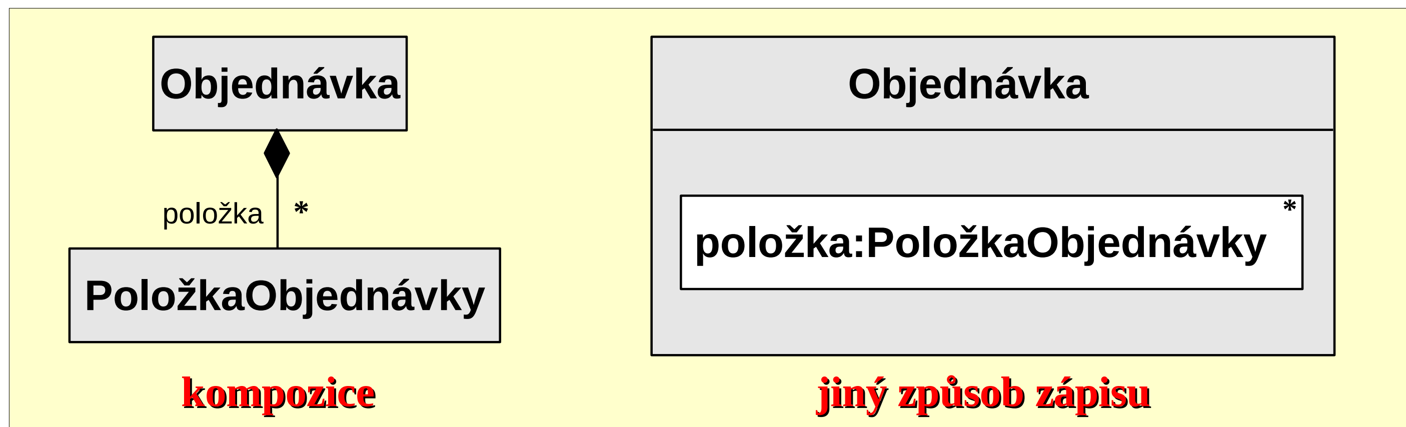
# Agregace

- Angl. *Aggregation*
- Speciální případ asociace pro **vztah celek-část**
- Symbol prázdného diamantu
- Tranzitivní
  - pokud je píst součást motoru a motor součástí auta, pak je i píst součástí auta
- Asymetrická
  - pokud je motor součástí auta, pak auto není součástí motoru
  - diagram objektů musí být acyklický
- Sémantika:
  - totéž co asociace, pouze zdůrazňujeme vztah celek-část
  - součásti mohou existovat nezávisle na celku
  - součást může být sdílena více celky



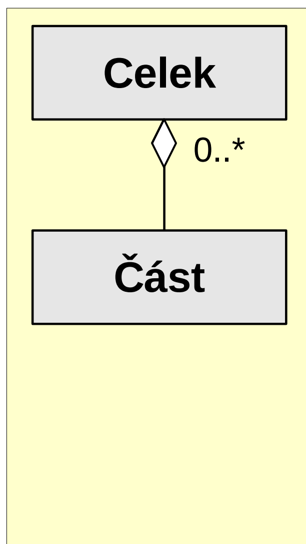
# Kompozice

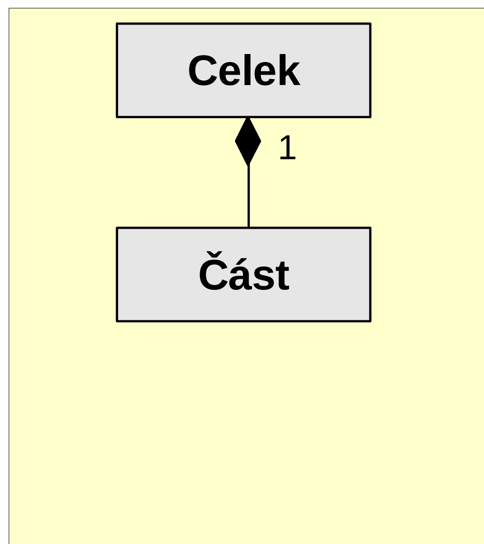
- Angl. *Composition*
- Silnější forma agregace
- Symbol plného diamantu
- Tranzitivní a asymetrická
- Sémantika:
  - na úrovní instancí (objektů) části neexistují vně celku
  - každá část patří pouze do jediného celku (části nelze sdílet)
  - je-li celek zničen, musí zničit i svoje součásti, nebo převést zodpovědnost za ně na jiný objekt

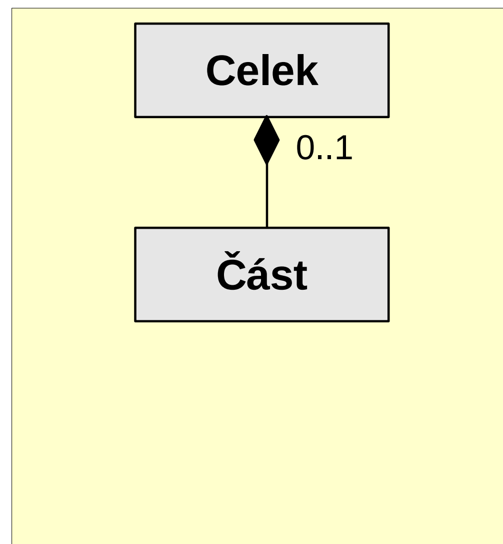


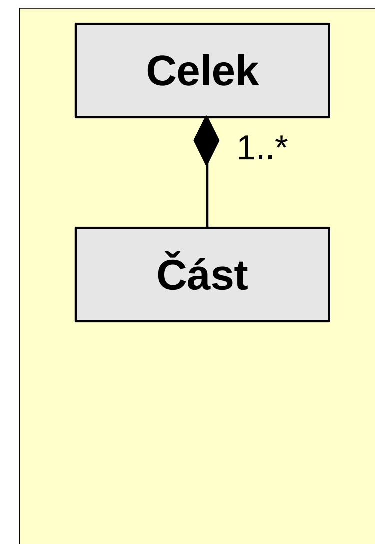
# Vlastnosti agregace a kompozice (I)

- Které modely jsou správně a které naopak špatně z důvodu vlastností agregace a kompozice?



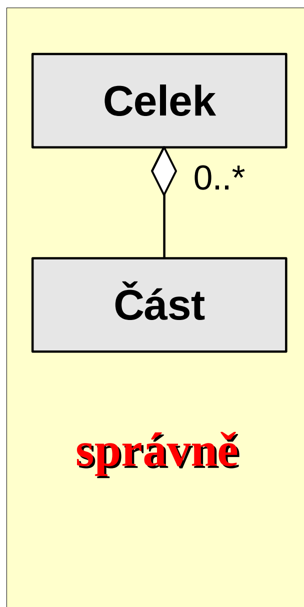



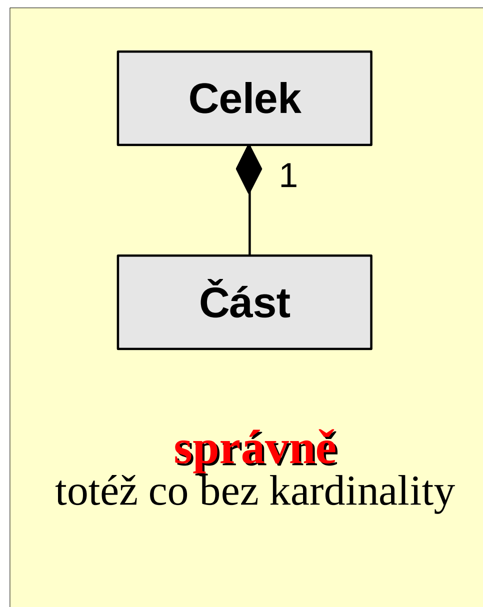



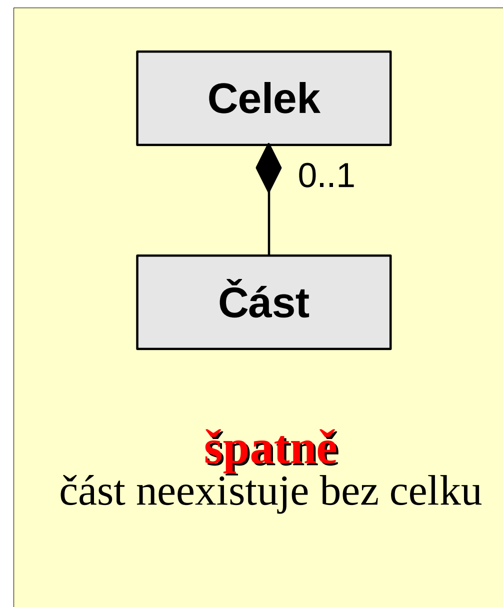

... odpověď na další obrazovce >>

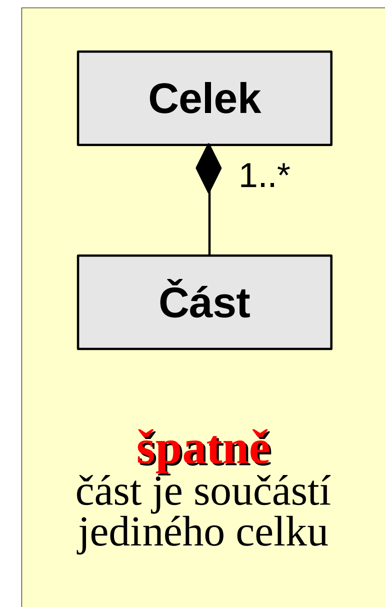
# Vlastnosti agregace a kompozice (II)

... odpovědi z předchozí stránky:



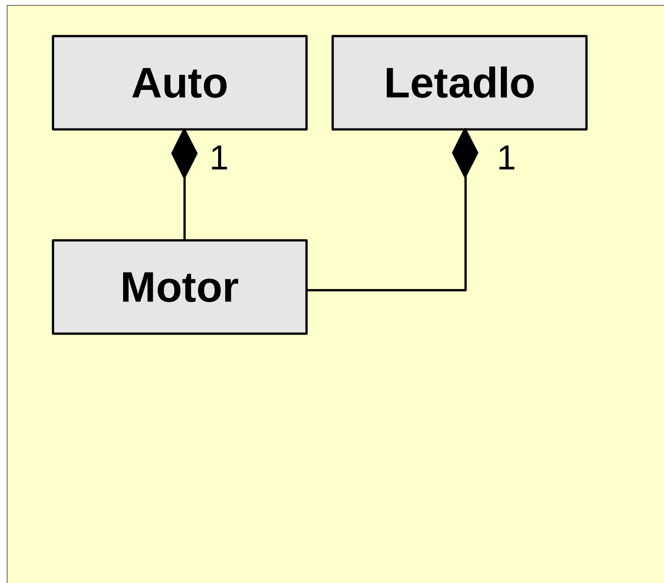



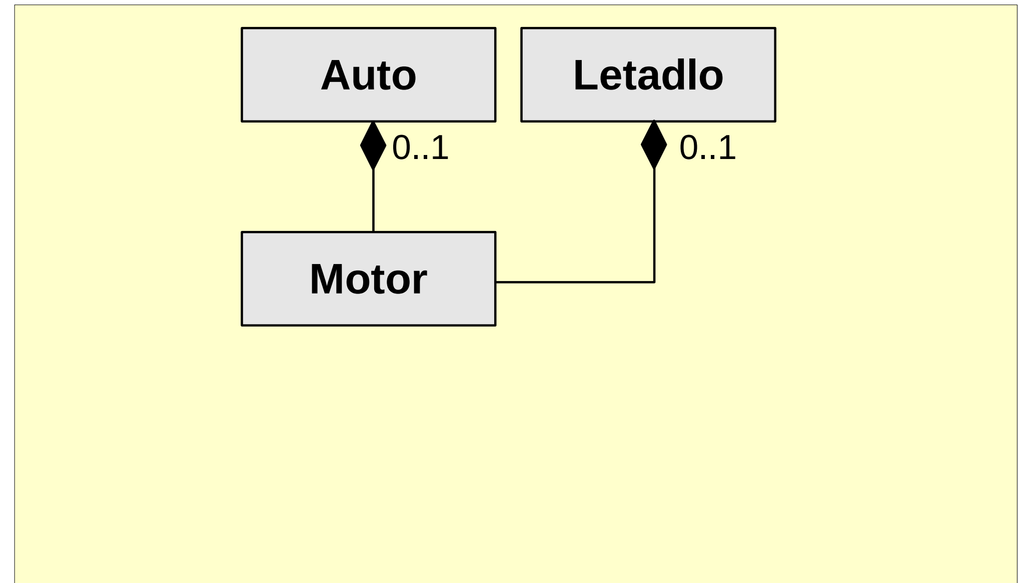



# Vlastnosti agregace a kompozice (IV)

- Které modely jsou správně a které naopak špatně z důvodu vlastností agregace a kompozice?

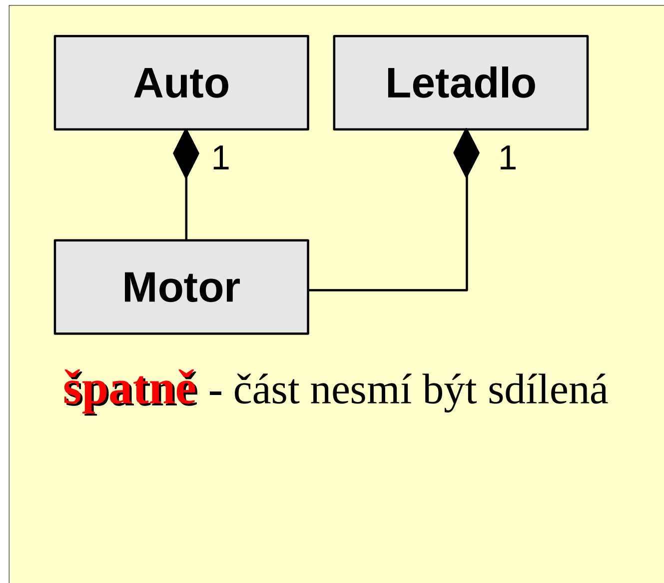


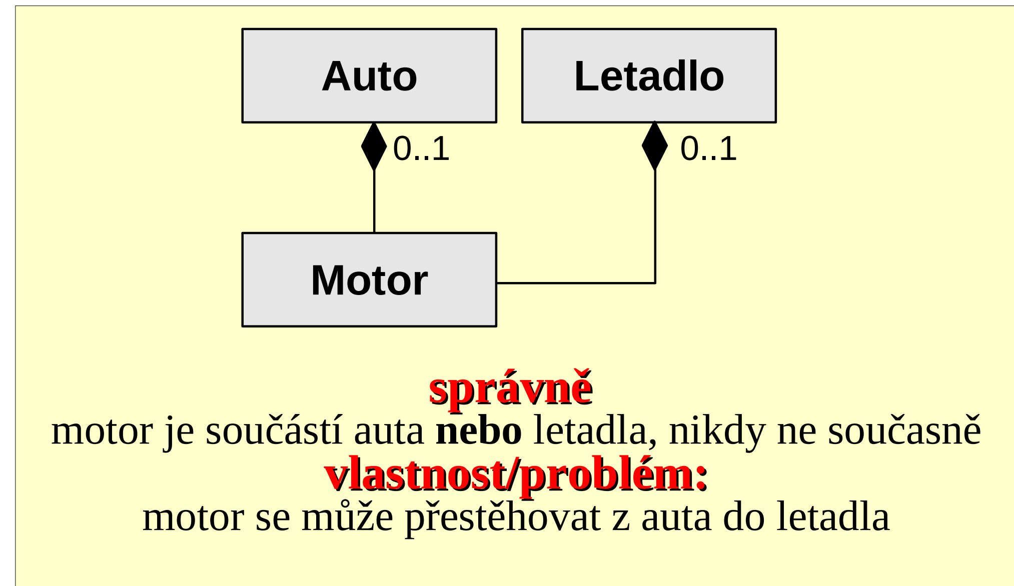



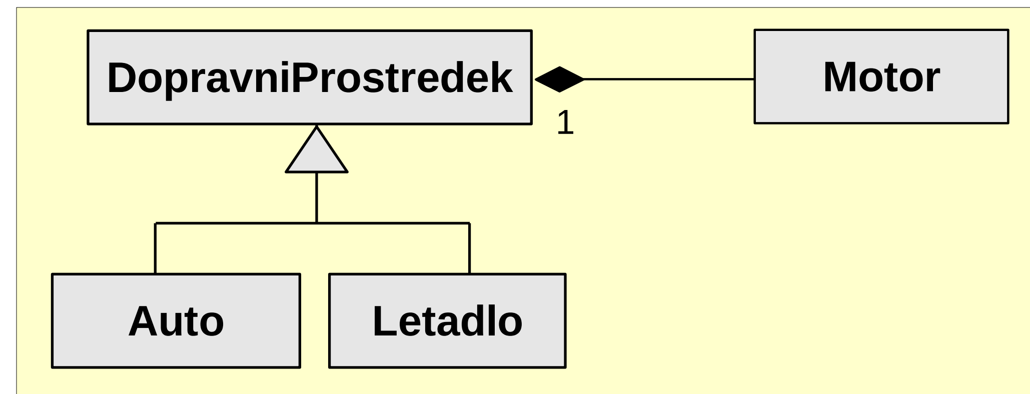

... odpověď na další obrazovce >>

# Vlastnosti agregace a kompozice (V)

... odpovědi z předchozí stránky:

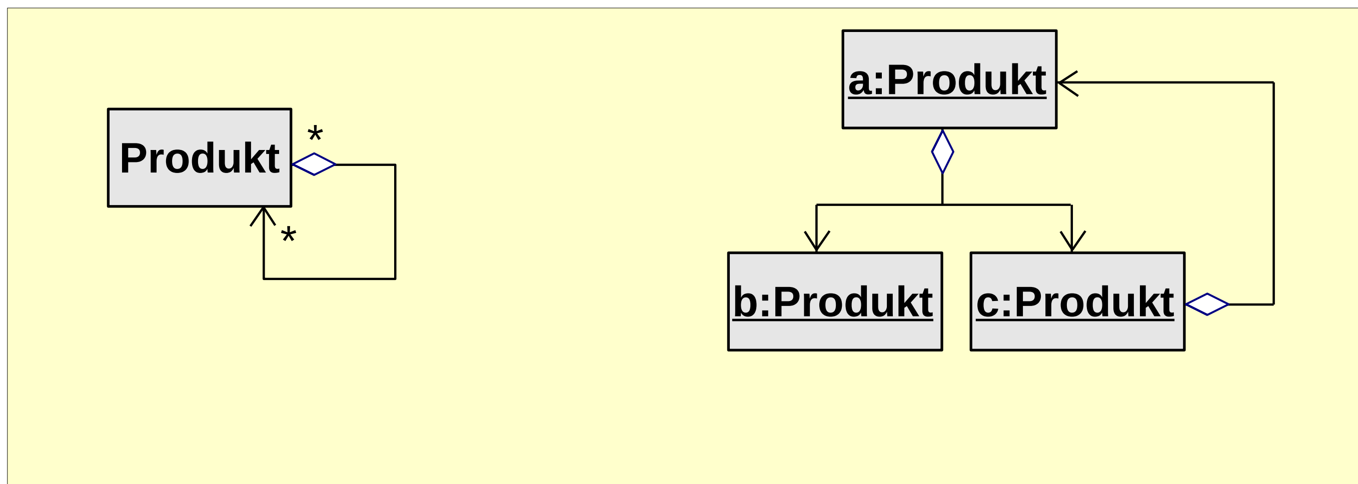


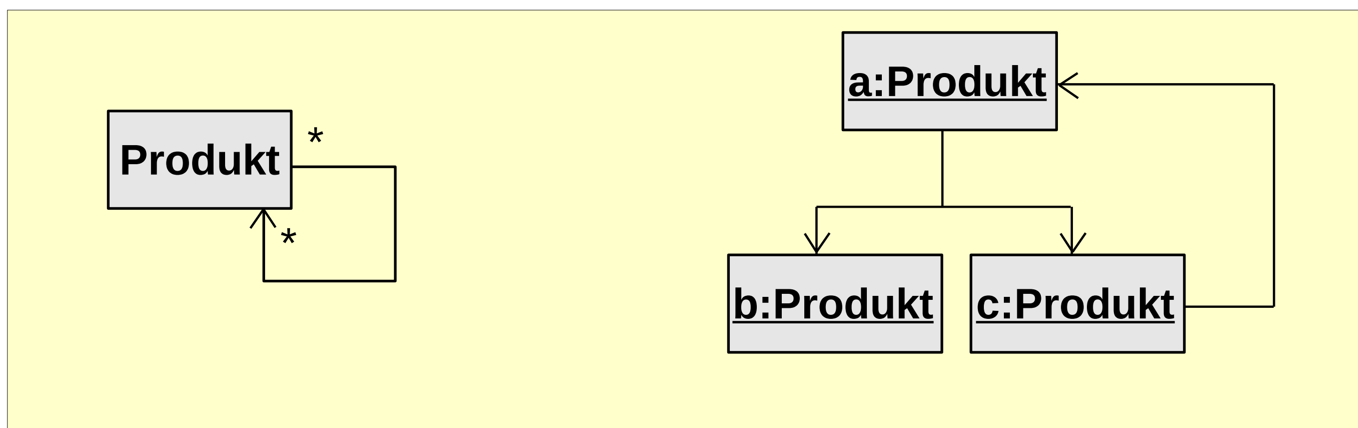





# Vlastnosti agregace a kompozice (VI)

- Které modely jsou správně a které naopak špatně z důvodu asymetrie agregace?



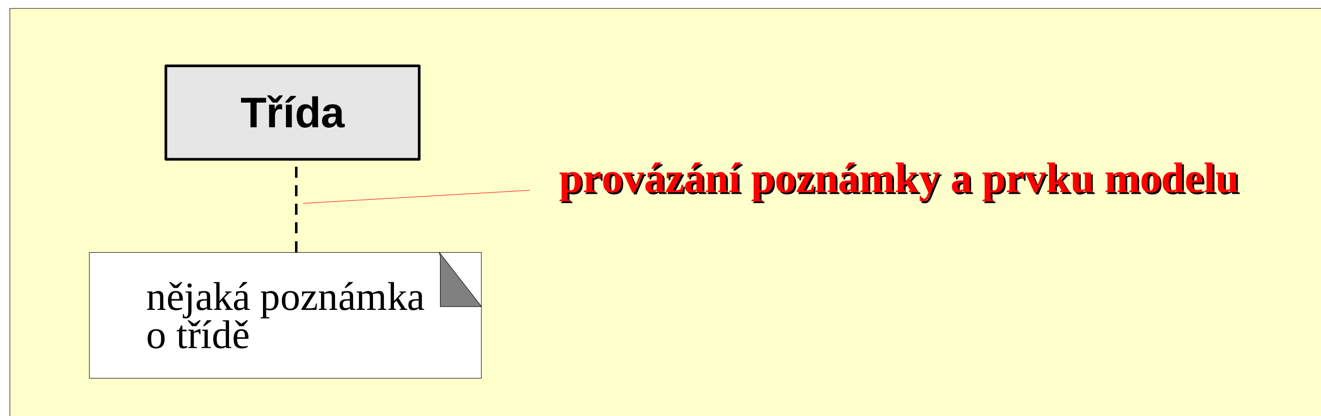



# Vlastnosti agregace a kompozice (VII)

... odpovědi z předchozí stránky:

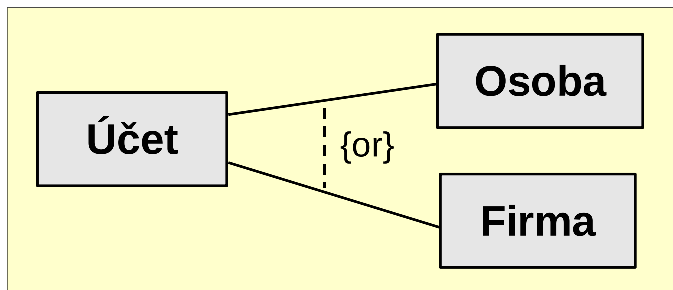
# Poznámky

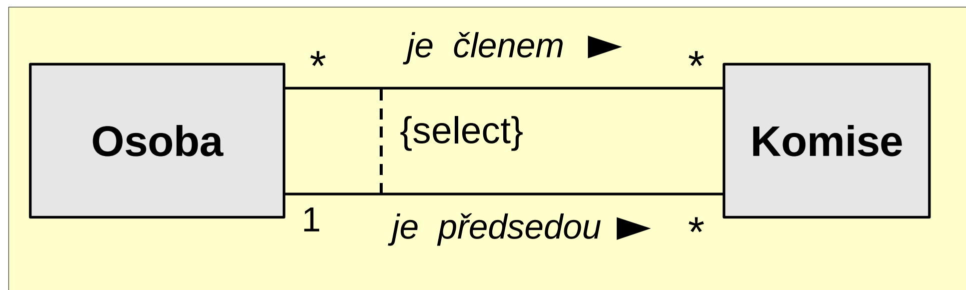
- Poznámka může připojit libovolnou textovou informaci k libovolnému prvku pole
  - v poznámkovém okénku
  - např. doplňující vysvětlení, otevřené otázky, stav prověrky, seznam nevyřešených úkolů, ...
- Poznámky jsou k dispozici ve všech UML diagramech

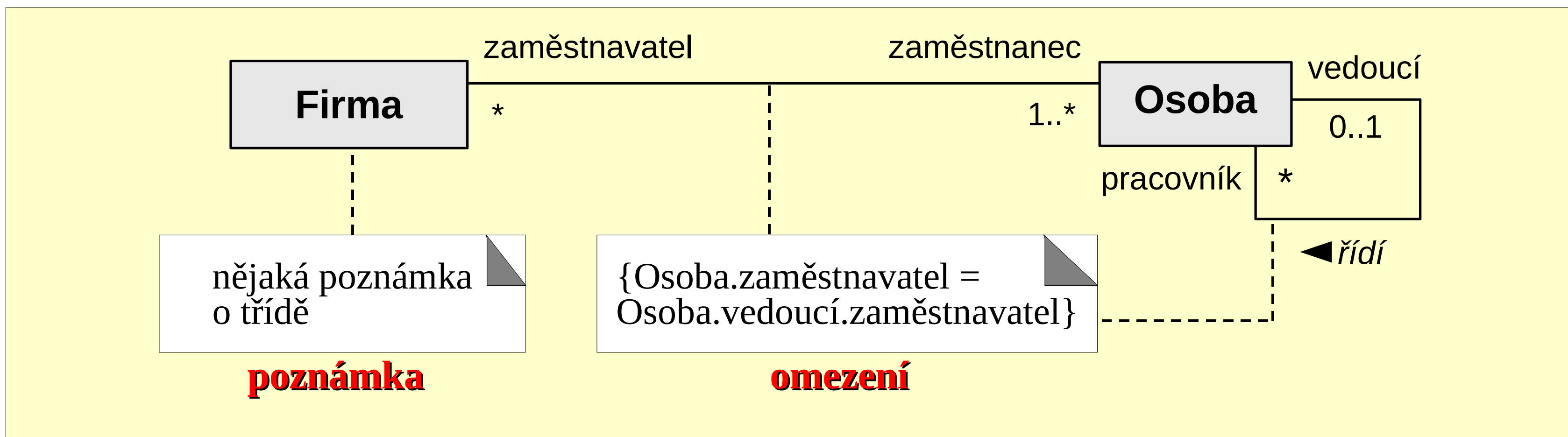


# Omezení v CD

- 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 {}



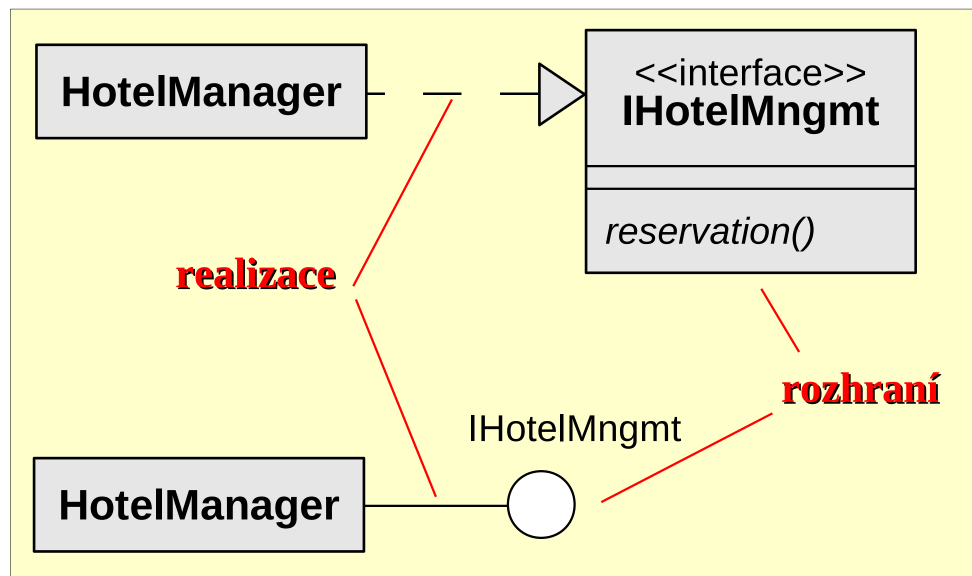
# Odvozené atributy a asociace

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

# Rozhraní

- Angl. *Interface*
- Speciální třídy, které definují *externě viditelné služby* nějaké třídy nebo komponenty, bez *specifikace interní struktury* (atributy, stavy, implementace metod).
- Často popisují pouze část, nikoliv celé chování příslušné třídy.
- Používají stejné vztahy jako třídy, navíc ještě vztah *realizace* (angl. *realization*).
- Dvě notace.

Rozhraní = nabídka služeb realizovaných třídami



# Rozhraní: dvě notace (příklad)

<<interface>>
**Hashable**

*hash(): Integer*

**String**

hash(): Integer
equals(Object): Boolean

<<use>>

**HashTable**

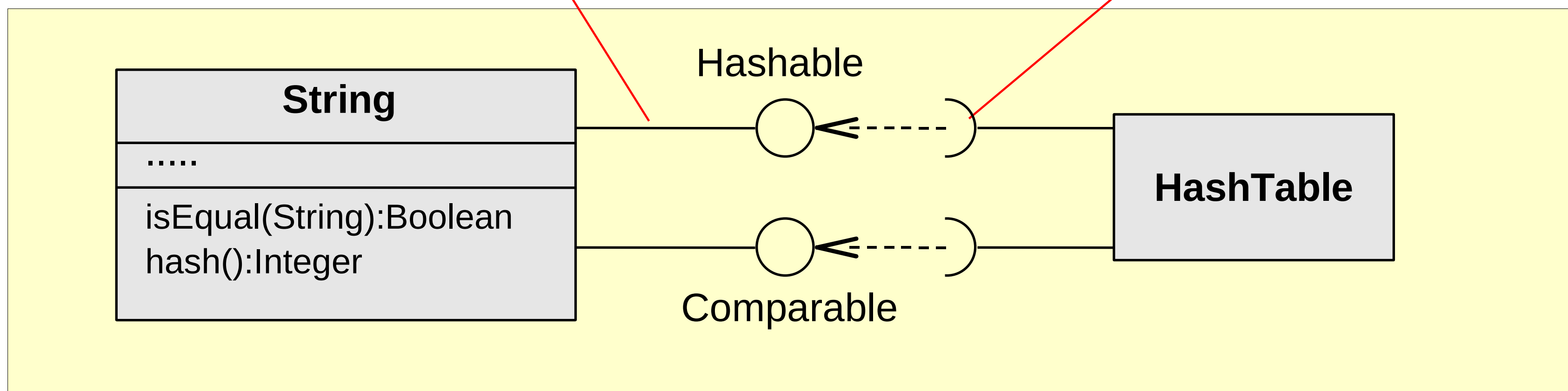
<<interface>>
**Comparable**

*equals(Object): Boolean*

<<use>>

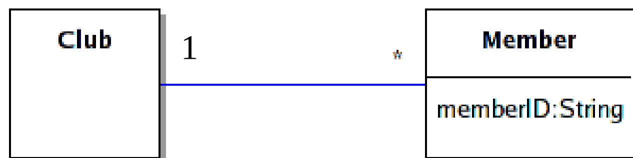
**realizace**

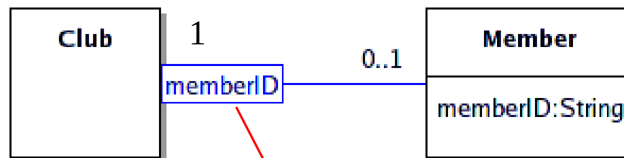
**závislost**



# Asociace s kvalifikátorem

- Angl: *Qualified association*
- Slouží k výběru jednoho člena cílové množiny pomocí jednoznačného identifikátoru (klíče)
- Kvalifikátor se obvykle odkazuje na atribut cílové třídy
- Kvalifikátor je součástí asociace, nikoliv třídy!




kvalifikátor

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

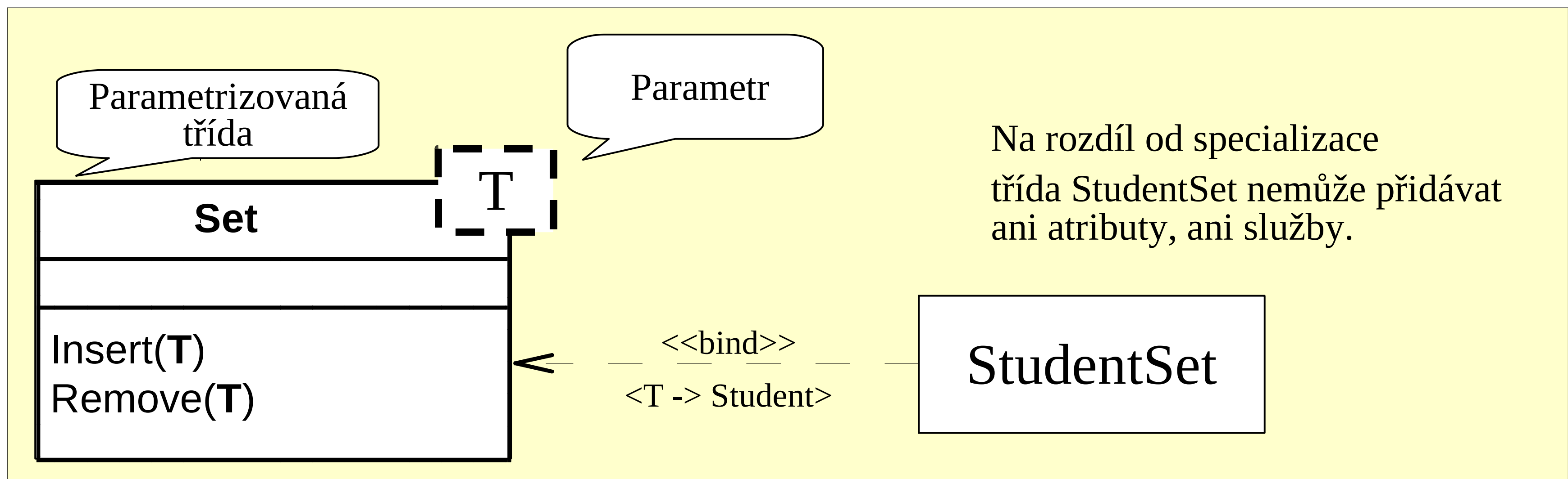
Kombinace {Club, memberID} specifikuje jedinečný cíl

# Parametrizované třídy, šablony

Parametrizovaná třída definuje *rodinu* tříd.

Nelze ji přímo použít, je nutné ji nejprve instanciovat přiřazením potřebných typů. Instanciace teprve vytvoří použitelnou třídu.

<u>Příklad</u>: třída **Set** – reprezentuje kolekci objektů nějaké *třídy.* Tato třída se stává parametrem parametrizované třídy *např. množina* ***mincí****, množina* ***studentů****, množina <T>*



# Př: Analytický model tříd

Visual Paradigm for UML Standard Edition(Masaryk University)

**InterniLektor**
mesicniMzda

**ExterniLektor**
hodinovaMzda

**Lektor**
jmeno
adresa
email
pridatLektora()
smazatLektora()
upravitUdaje()

zodpovědná osoba
1
vyučující
1..*

**Student**
jmeno
adresa
email
pridatStudenta()
smazatStudenta()
upravitUdaje()

0..*

1
zadává
0..*

0..* kurzy 0..* kurzy

**Kurz**
nazev
popis
prerekvizity
kapacita
rozvrh
pridatKurz()
smazatKurz()
upravitUdaje()
prihlasitStudenta()
odhlasitStudenta()
zaslatUpozorneni()

studuje
0..*

**PracovniVykaz**
odpracovaneHodiny
datum
vlozitVykaz()
smazatVykaz()

0..* 1

**PrezencniKurz**

**DistancniKurz**

# Př: Návrhový model tříd (bez balíků)

Visual Paradigm for UML Standard Edition(Masaryk University)

**InterniLektor**
-mesicniMzda : int
+InterniLektor()
+getMesicniMzda() : int
+setMesicniMzda(mzda : int) : void

**ExterniLektor**
-hodinovaMzda : int
+ExterniLektor()
+getHodinovaMzda() : int
+setHodinovaMzda(mzda : int) : v...
+getVykazy() : PracovniVykaz
+addVykaz(kurz : Kurz, odpracova...
+remVykaz(vykaz : PracovniVykaz...

***Lektor***
-jmeno : string
-adresa : string[]
-email : string
+getJmeno() : string
+setJmeno(jmeno : string) : void
+getAdresa() : string []
+setAdresa(ulice : string, mest...
+getEmail() : string
+setEmail() : void

**SpravceUziv**
+SpravceUziv()
+pridatLektora(typ : TypLekt...
+smazatLektora(lektor : Lekt...
+vyhledatLektora(jmeno : str...
+pridatStudenta(jmeno : stri...
+smazatStudenta(student : ...
+vyhledatStudenta(jmeno : ...
+prihlasitUziv(login : string, ...
+odhlasitUzivatele(uzivatel : ...

**Student**
-jmeno : string
-adresa : string[]
-email : string
+Student(jmeno : string, ulice :...
+getJmeno() : string
+setJmeno(jmeno : string) : void
+getAdresa() : string []
+setAdresa(ulice : string, mest...
+getEmail() : string
+setEmail(email : string) : void
+getKurzy() : Kurz []

**PracovniVykaz**
-odpracovaneHodiny : int
-datum : Date
+PracovniVykaz(kurz : Kurz, o...
+getKurz() : Kurz
+getOdpracovaneHodiny() : int
+getDatum() : Date

**Vyucuje**
-semestr : Semestr
+Vyucuje(vyucujici : Lek...
+getSemestr() : Semestr
+getVyucujici() : Lektor

***Kurz***
-nazev : string
-popis : string
-prerekvizity : string[]
+getNazev() : string
+setNazev(nazev : string) : void
+getPopis() : string
+setPopis(popis : string) : void
+getPrerekvizity() : string []
+setPrerekvizity(prerekvizity : string ...
+getStudujici(semestr : Semestr) : S...
+addStudujici(studuje : Studuje) : void
+remStudujici(student : Student, se...
+getVyucujici(semestr : Semestr) : L...
+addVyucujici(vyucuje : Vyucuje) : v...
+remVyucujici(vyucujici : Lektor, se...
+getZodpovednaOsoba() : Lektor
+setZodpovednaOsoba(osoba : Lekt...
+zaslatUpozorneni() : void

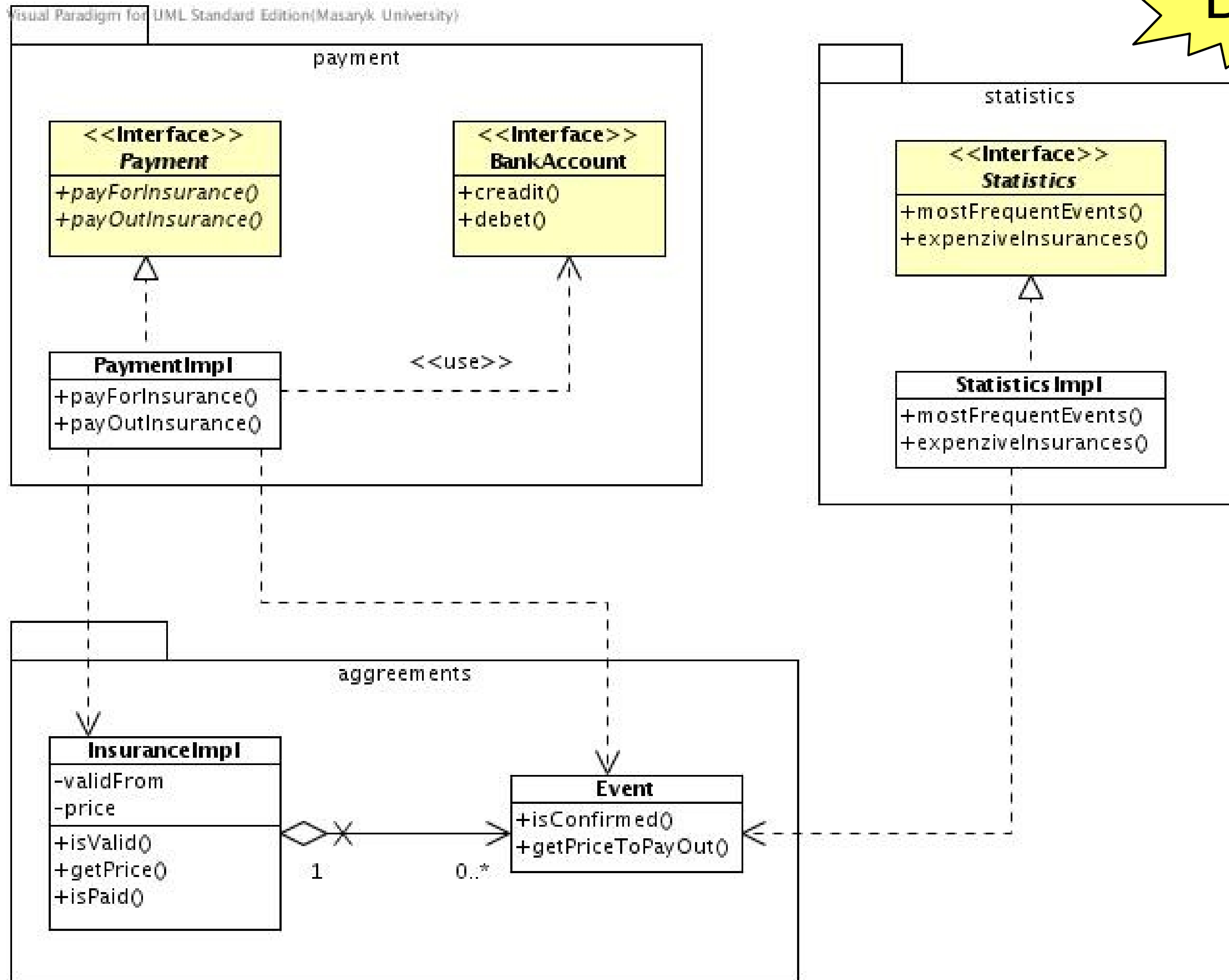
**Studuje**
-semestr : Semestr
+Studuje(student : Studen...
+getSemestr() : Semestr
+getStudent() : Student

**PrezencniKurz**
-rozvrh : RozvrhP
-kapacita : int
+PrezencniKurz()
+getRozvrh() : Rozv...
+setRozvrh(rozvrh : ...
+getKapacita() : int
+setKapacita(kapac...

**DistancniKurz**
-rozvrh : RozvrhD
+DistancniKurz()
+getRozvrh() : Rozv...
+setRozvrh(rozvrh : ...

**KatalogKurzu**
+KatalogKurzu()
+pridatKurz()
+smazatKurz()
+vyhledatKurz(kurz : Kurz)
+vyhledatKurzy(student : Student)
+prihlasitStudKurz(student : Stu...
+odhlasitStudKurz(student : Stu...

# Pojišťovna: Část návrhového modelu




Demo

# **Diagram balíků**

*(Package Diagram)*

# 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

<<subsystem>>
Finances

<<subsystem>>
Finances

<<subsystem>>
Credits

<<subsystem>>
BankInterface

Accounts

<<subsystem>>
Finances

+ Credits
+ BankInterface
- Accounts

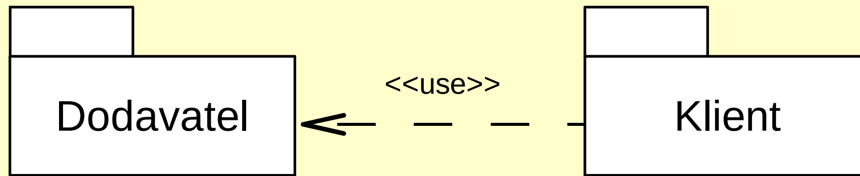
<<subsystem>>
Finances

<<subsystem>>
Credits

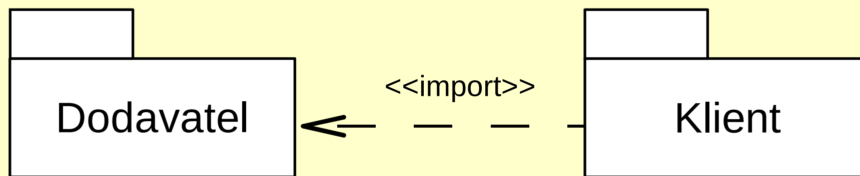
<<subsystem>>
BankInterface

Accounts

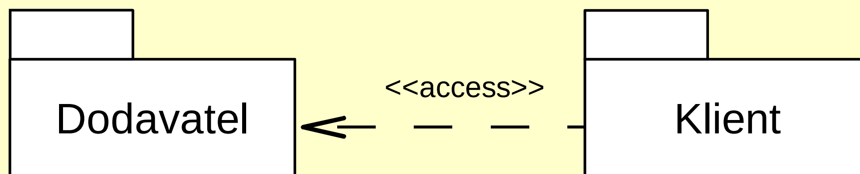
# Balíky - závislosti



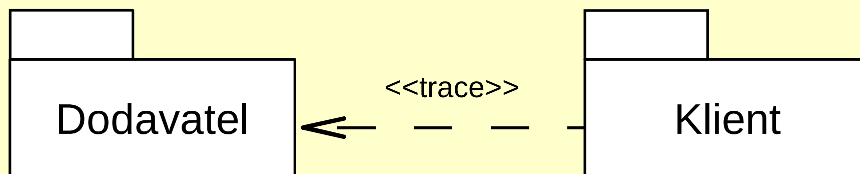

Některý element klienta závisí na některém elementu z dodavatele.
Totéž co bez stereotypu.
Vhodné pro analýzu, v návrhu se upřesní <<import>> nebo <<access>>




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




<<use>> + jmenné prostory zůstávají oddělené.

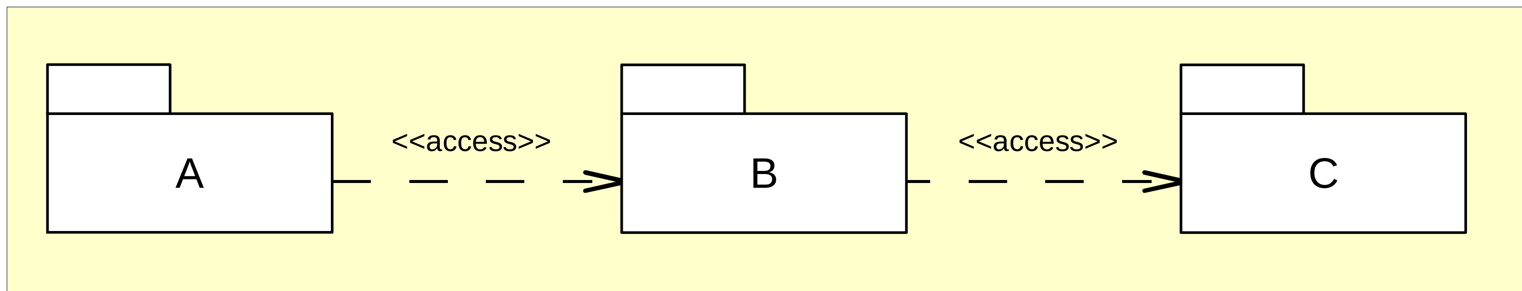



Klient je dalším vývojovým stupněm dodavatele (v průběhu analýzy se balík *Dodavatel* změnil na *Klient*)

# 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

<<subsystem>>
User Interface

<<subsystem>>
Business Rules
Benefits

<<subsystem>>
Business Rules
Salary

<<subsystem>>
Database

# Subsytémy – kritéria návrhu

- Každý subsystém by měl mít:
  - jedinou funkcionalitu
  - silnou kohezi
    - silné vztahy mezi třídami uvnitř subsystému
  - volné (externí) propojení
    - minimalizovat závislosti mezi subsystémy
  - znovupoužitelnost
    - komponenty
    - návrhové vzory – viz. další přednášky
- Vyhněte se cyklické závislosti mezi balíky
  - cyklicky závislé třídy/podbalíky patří do jednoho balíku
    - spojení do jednoho balíku
    - přerozdělení na jiné balíky s acyklickými vztahy

# Pojišťovna: Diagram balíků

Demo

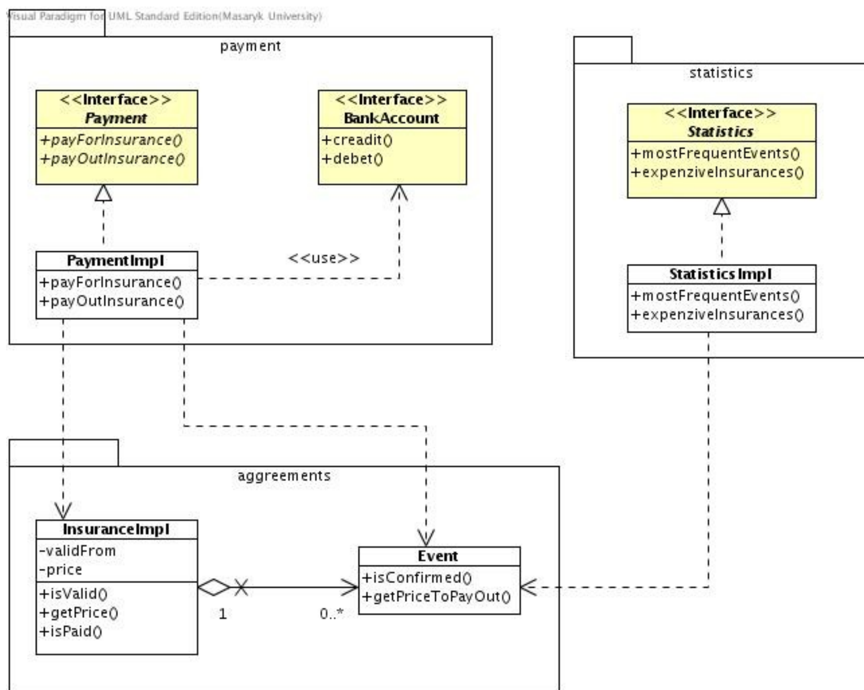
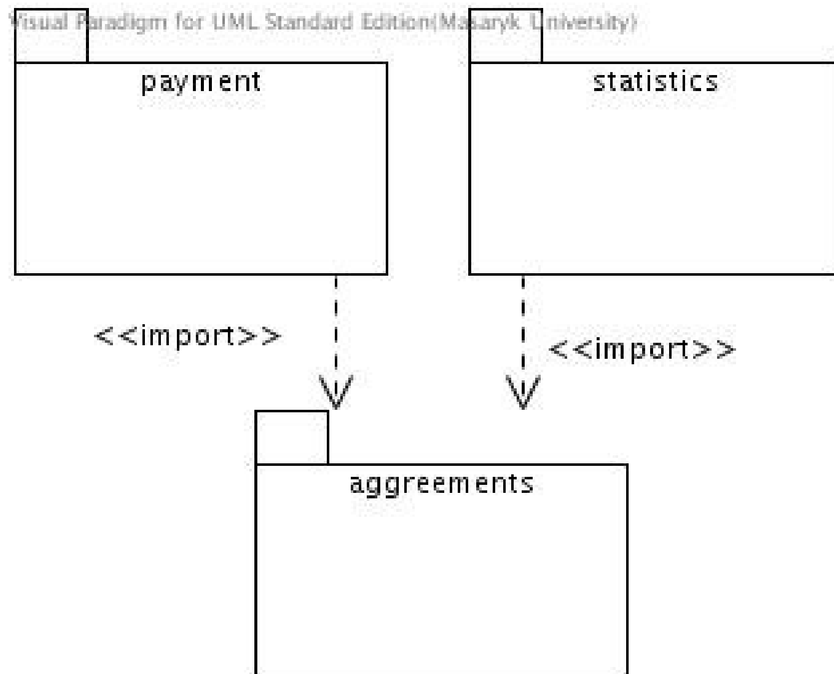



Diagram tříd včetně balíků




Přehlednější diagram balíků

# **Diagram komponent**
*(Component Diagram)*

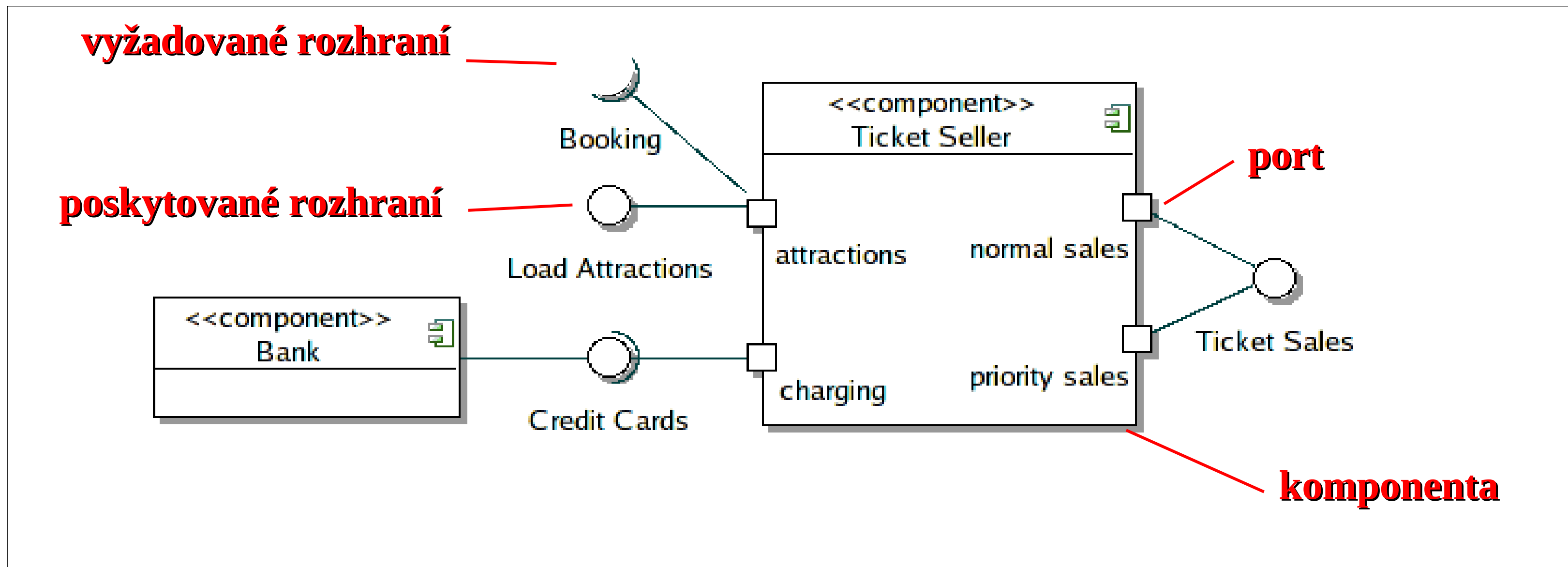
# Modelování komponent

- Model v etapě návrhu
- Spojení konceptu balíků  a rozhraní
- Co je to komponenta?
  - Komponenta je jeden nebo více objektů sdružených do definovaného programového rozhraní
  - Komponenta poskytuje ucelený soubor služeb a je navržena pro vícenásobné použití
  - Komponenta je samostatně spustitelná a připojitelná za běhu

| **Komponenty** | **Objekty/třídy** |
|---|---|
| větší celky | jemné subjekty |
| více rozhraní | jedno rozhraní |
| poskytují služby | poskytují operace |
| plně zapouzdřené | využívají dědičnost, asociace |
| obecně pochopitelné | pochopitelné pro vývojáře |

# Diagram komponent

- Angl: *Component Diagram*
- Ukazuje strukturu kódu
  - *rozhraní* – kolekce operací které jsou poskytovány nebo vyžadovány komponentou, viz. třídy
  - *komponenta* – vyměnitelná část systému; může obsahovat vnitřní strukturu (propojené komponenty nižší úrovně)
  - *port* – “okno” do uzavřené komponenty akceptující zprávy z/do komponenty odpovídající specifikovanému rozhraní



# Pojišťovna: Diagram komponent

Demo

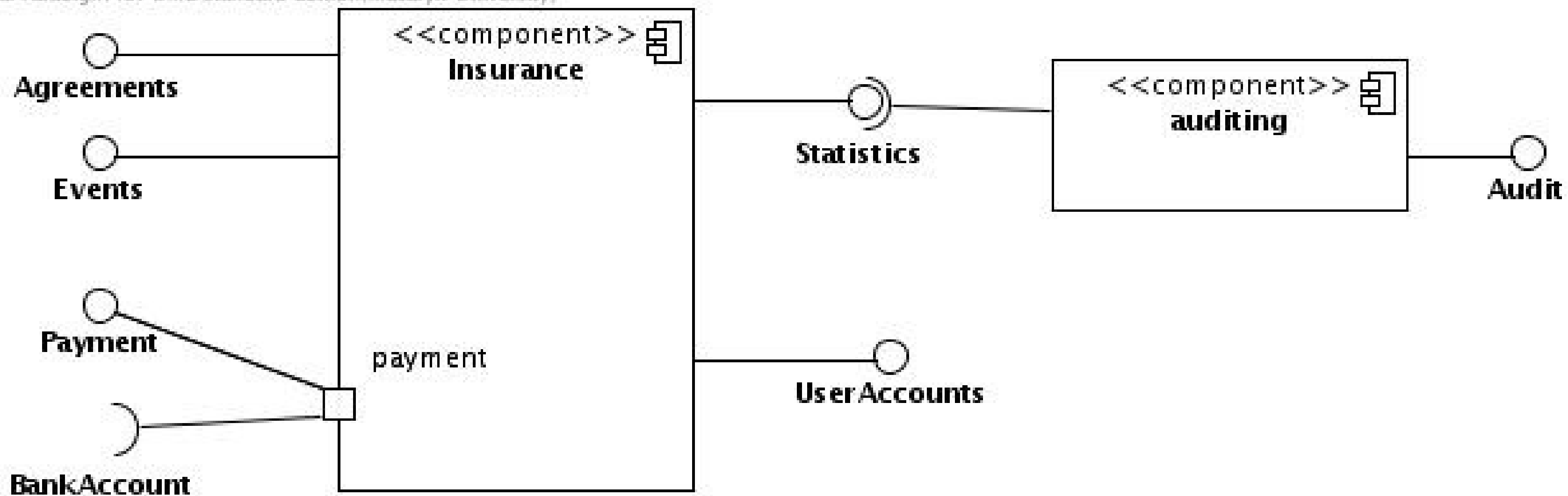


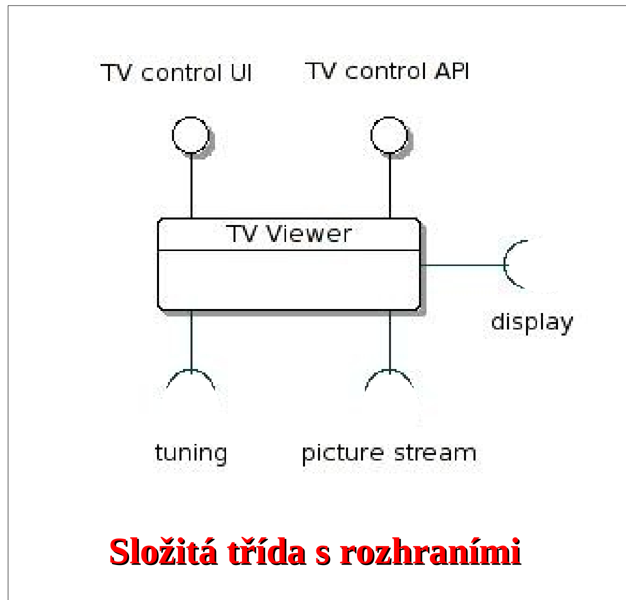
**Diagram vnitřní struktury**

*(Composite Structure Diagram)*

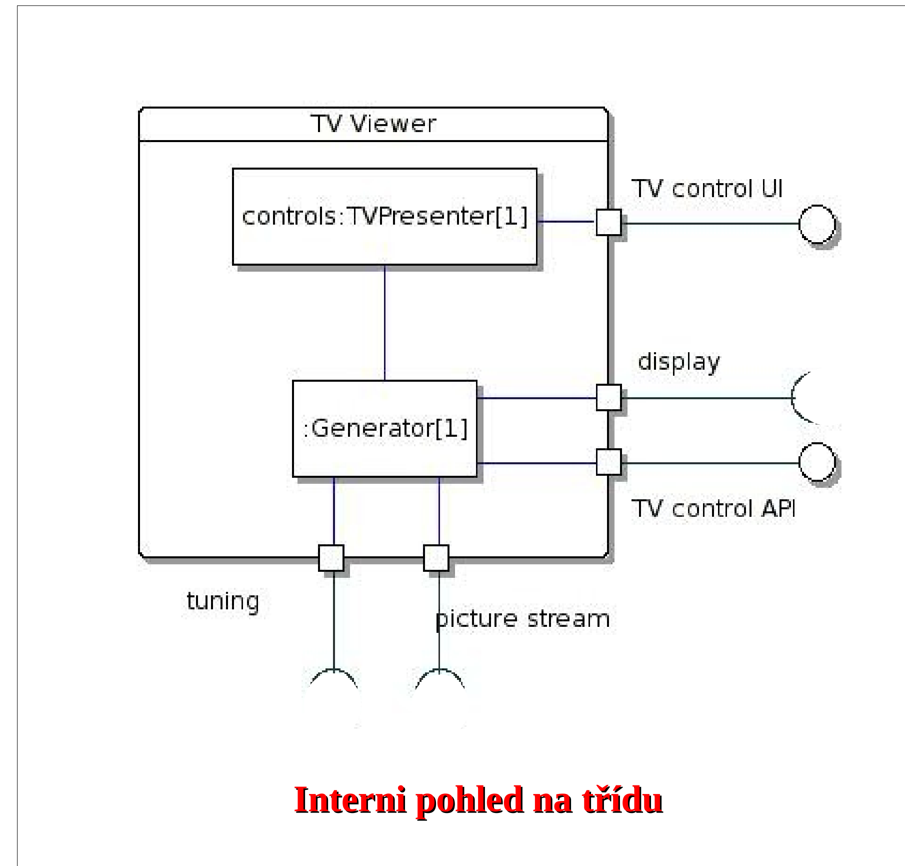
# Diagram vnitřní struktury

- Angl: *Composite Structure Diagram, CSD*
- Novinka od UML 2.0
- Ukazuje strukturu složitých tříd
  - *rozhraní* – kolekce operací které jsou poskytovány nebo vyžadovány komponentou, viz. třídy
  - *část* – nejsou to instance tříd, jsou vnímány obecněji
  - *port* – “okno” do uzavřené komponenty akceptující zprávy z/do komponenty odpovídající specifikovanému rozhraní
- **Na rozdíl od balíků ukazuje rozdělení systému za běhu (runtime grouping)**
- Podobná notace jako u komponent.

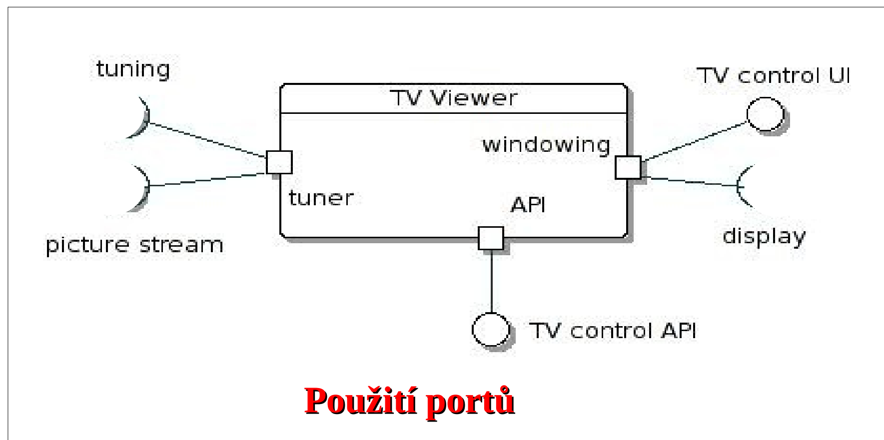
# CSD: Příklad




**Složitá třída s rozhraními**




**Interni pohled na třídu**

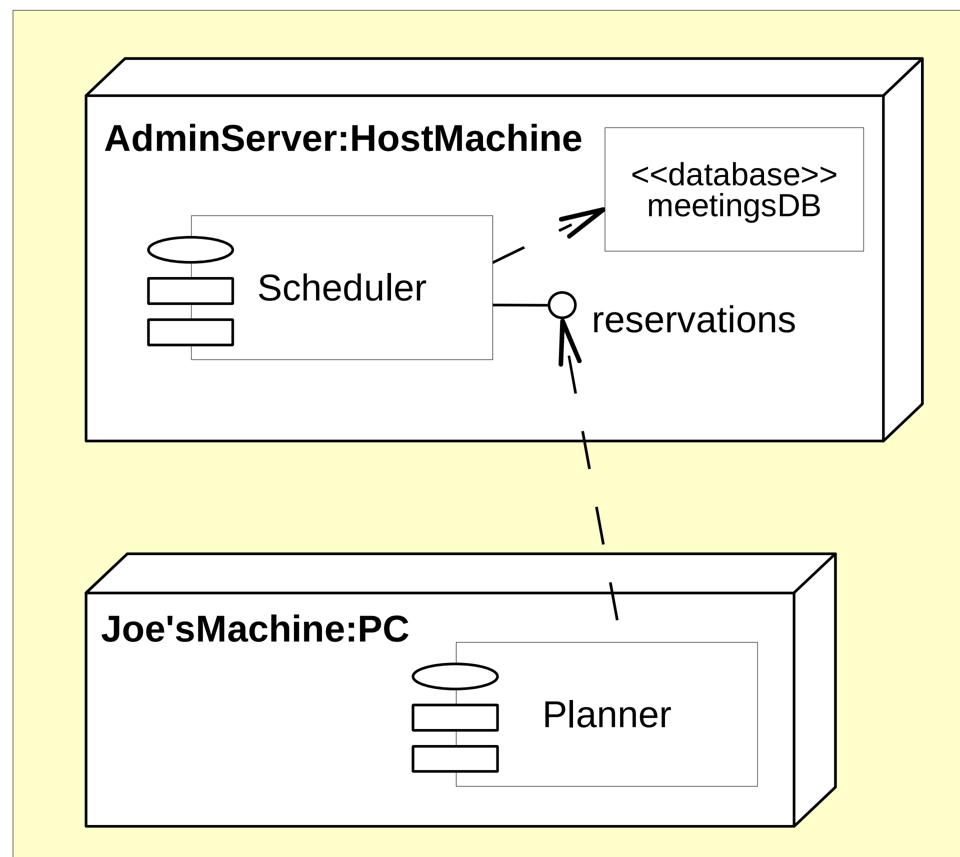



**Použití portů**

## **Diagram rozmístění**
## *(Deployment Diagram)*

# Diagram rozmístění

- Angl: *Deployment Diagram*
- Model v etapě návrhu a implementace
- Ukazuje strukturu spustitelného programu
- Fyzické rozmístění systému



# Pojišťovna: Diagram rozmístění

Demo

Visual Paradigm for UML Standard Edition(Masaryk University)

WebServer:Apache

<<artifact>>
PHP web pages

SOAP

HTML

ApplicationServer:Tomcat

<<component>>
Insurance

Statistics

<<component>>
auditing

HTML/SSL

User

<<artifact>>
WebBrowser

Auditor:PC

<<artifact>>
JavaClient

XML/SSL

Oracle

<<artifact>>
DB

BackupServer:Oracle

<<artifact>>
DB

# Tipy pro strukturální modelování

- Definujte „kostru“ (nebo „páteř“), kterou lze rozšiřovat a upravovat poté, co získáte více poznatků o předmětné oblasti.
- Soustřeďte se na správné použití základních konstrukcí; pokročilejší konstrukce a/nebo notaci přidávejte jen, pokud je to požadováno.
- Odložte implementační úvahy na pozdější etapu modelování.
- Strukturální diagramy mají
  - zvýraznit určitý aspekt strukturálního modelu
  - obsahovat klasifikátory na stejné úrovni abstrakce
- Při větších počtech klasifikátorů by měly být zavedeny balíky.