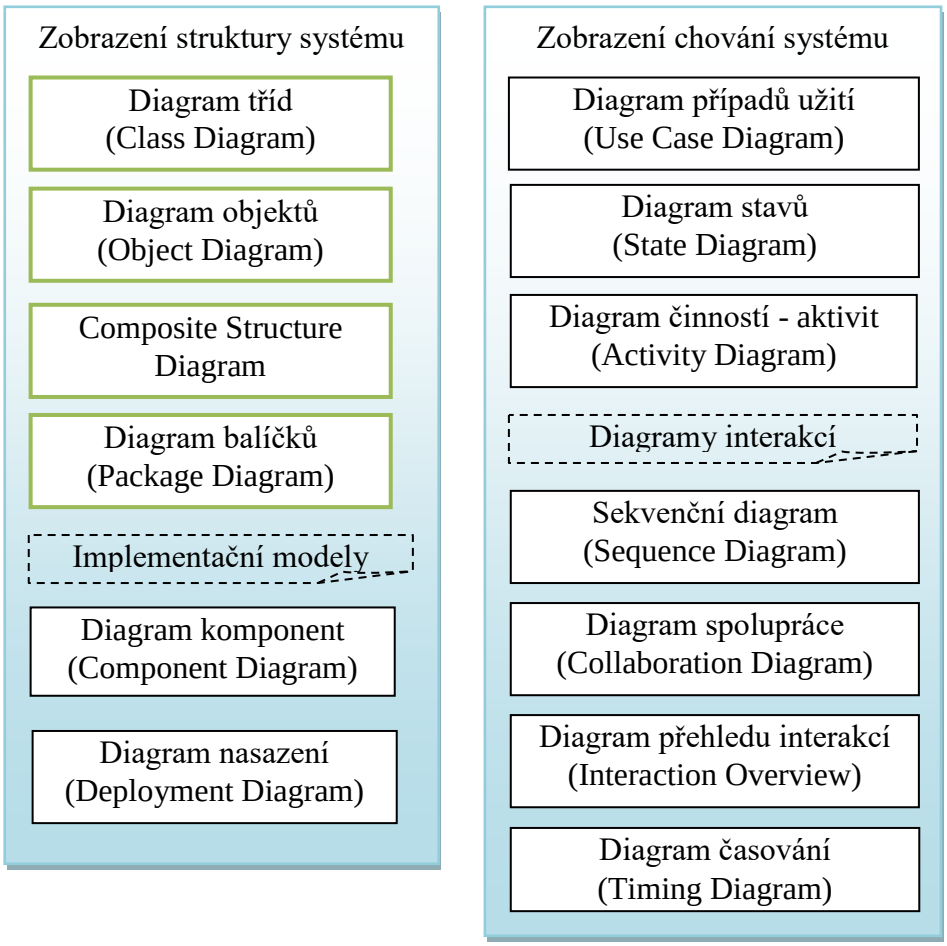
Předmět:	**Projektování IS**
Téma:	**Objektový přístup k vývoji IS (část B)**

Vyučující:	dr. Dušan Kajzar	Školní rok:	2020/2021

Obsah:

Kde se nacházíme ?



Obrázek: Schéma modelů objektově orientované analýzy a návrhu IS

1. Diagram tříd (Class Diagram)

Účel modelu:

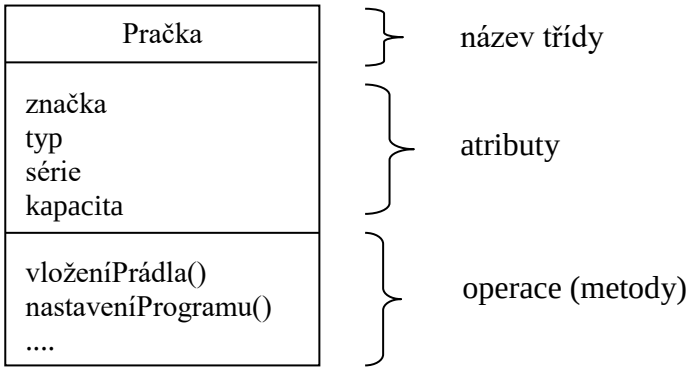
- zobrazit vnitřní strukturu systému,
- třídy (objekty) - jejich vlastnosti a vztahy.

Základní pojmy:

- objekt
  - konkrétní prvek systému,
  - jednoznačně identifikovatelná entita obsahující data + operace s daty,
  - součástí objektu - identifikátor, vlastnosti (data), chování (operace),
- třída
  - kategorie (zobecnění skupiny) objektů,
  - stejné či podobné vlastnosti a chování,
  - obsahuje atributy a popis operací (metod),
- vtah třídy a objektu
  - třída je „šablona“ k vytváření objektů,
  - objekt je konkrétním prvkem dané třídy,
  - objekt je instancí (tj. konkrétním výskytem) dané třídy,
  - „objekty jsou dělníci systému – vykonávají všechny činnosti“,
  - „navrhujeme třídy, pracují však objekty“,
- atributy třídy (objektu)
  - položky popisující vlastnosti objektů (charakteristiky),
  - hodnoty atributů (stálé a proměnlivé),
- operace (resp. metody) třídy (objektu)
  - algoritmy popisující aktivity (akce a reakce) objektu,
- vazby mezi třídami (resp. mezi objekty)
  - objekt třídy A může použít operaci či hodnotu objektu třídy B,
  - asociace – vazby mezi třídami,
  - linky – vazby mezi objekty.

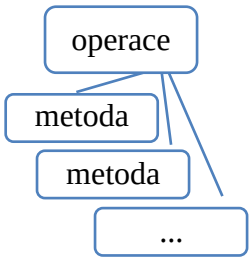
Třída v jazyce UML:

Obrázek: Základní grafické znázornění třídy



Obrázek: Grafické znázornění třídy s oddíly

Názvosloví „operace a metoda“:



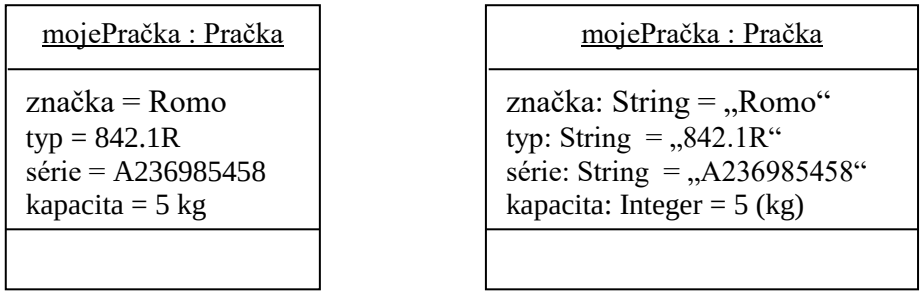
- názvosloví v literatuře není zcela jednotné (starší, novější),
- v etapě analýzy
  - hovoříme o operacích (abstraktní specifikace algoritmu),
- v etapě návrhu
  - hovoříme o metodách (konkrétní implementace),
- metoda – je implementací operace,
- operace – může být implementována jednou nebo i několika metodami.

Doplnění podrobností popisu třídy:

- doplnění tzv. adorments (ozdoby),
- analytická úroveň --> návrhová úroveň zobrazení,
- detaily rozpracováváme v závislosti na účelu diagramu,
- např. podrobný popis atributů třídy:
  - viditelnost název : typ [násobnost] = počáteční hodnota,
- podrobnosti mohou být implementačně závislé na cílovém jazyce.

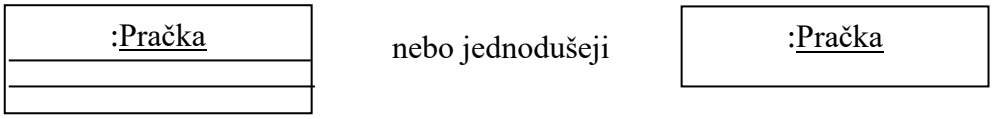
Objekt v jazyce UML:

- popis atributů objektu - název: datový typ = hodnota,
- stav objektu - je určen hodnotami jeho atributů.



Obrázek: Grafické znázornění objektu

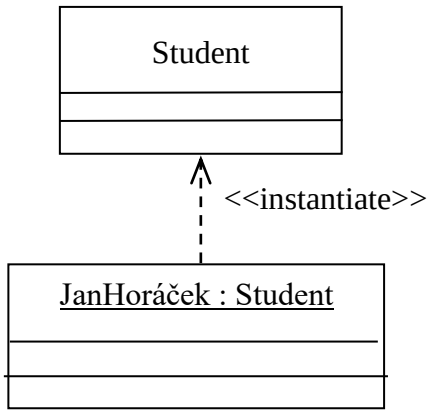
Anonymní objekt:



Obrázek: Libovolný (anonymní) objekt třídy „Pračka“

Relace (vztah) „třída -> objekt“:

- říkáme, že objekt je instancí (konkrétním výskytem) dané třídy,
- <<instantiate>> - jedná se o speciální případ relace závislosti,
- objekt „dědí“ atributy a operace (metody) dané třídy,
- atributy objektu nabývají konkrétních hodnot.



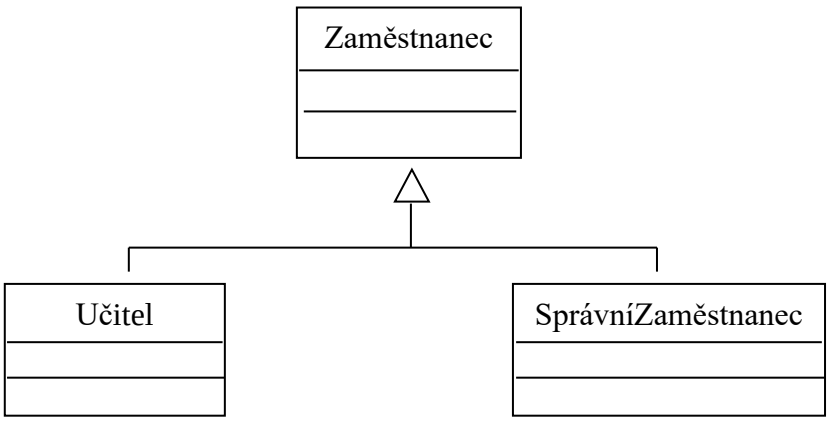
Obrázek: Vztah třída - objekt dané třídy

Tři pilíře objektového programování:

- dědičnost,
- polymorfismus,
- zapouzdření.

Dědičnost a hierarchie tříd:

- jde o vztah generalizace a specializace mezi třídami,
- vzniká vztah nadtřída a podtřída,
- potomci (podtřídy)
  - dědí - atributy, operace, relace, omezení,
  - mohou – přidávat charakteristiky, stávající předefinovat.



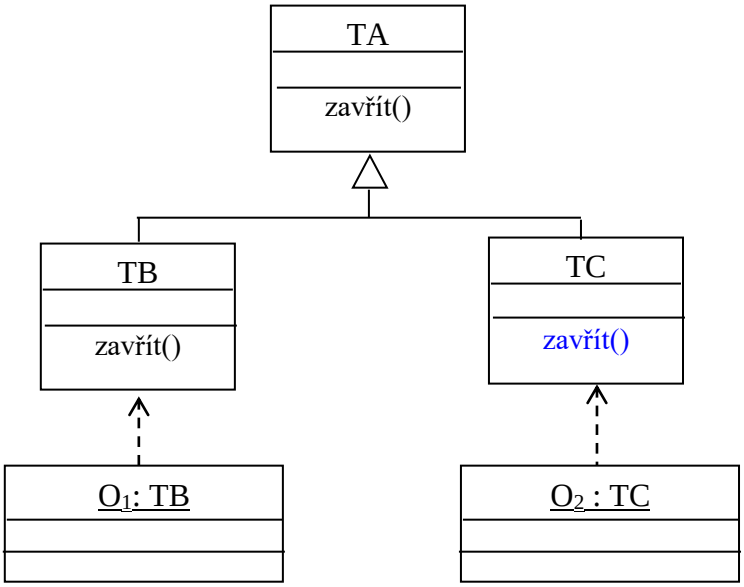
Obrázek: Grafické znázornění vztahu zobecnění (resp. dědičnosti)

K použití dědičnosti:

- myšlenkové procesy - zobecnění (generalizace) a specializace,
- jde o vztah mezi obecnějším prvkem a prvkem přesněji specifikovaným,
- jde o vysoký stupeň závislosti mezi prvky,
- platí zákon o nahraditelnosti
  - obecnější prvek lze nahradit speciálnějším typem,
  - spec. typ má tytéž atributy a metody, může mít i další navíc,
- platí různá pravidla a doporučení, např.
  - podtřídy by neměly reprezentovat role, nýbrž speciální druh třídy,
  - podtřída „je druhem“ (čeho) – nikoliv „je rolí“ (jiným stavem téhož),
- dědění od více předků – podporováno pouze v C++.

Polymorfismus (mnohotvárnost):

- potomci mohou změnit definici zděděných operací,
- polymorfní operace - mají více implementací,
- díky polymorfismu - objekty různých tříd mohou reagovat na stejné zprávy různým způsobem.



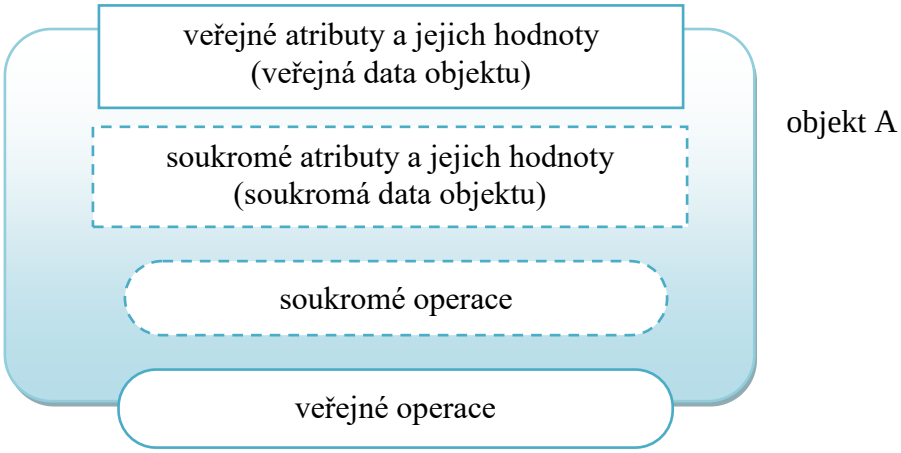
Obrázek: "Překrytí" operace zavřít() ve třídě TC

K polymorfismu operací:

- překrývání operací je nutné u abstraktních operací (viz dále),
- překrývání konkrétních operací se obecně považuje za špatné (!)
  - v podstatě tím ignorujeme existující implementaci,
  - otázka bezpečnosti (skryté vedlejší efekty),
- uznávaný způsob překrytí operace
  - pokud daná operace zavolá operaci nadtřídy a pak vykoná něco navíc (přidá vlastní chování),
- v jazyce Java
  - klíčové slovo „final“ - zamezení překrytí operace u potomků (!)

Zapouzdření atributů a operací:

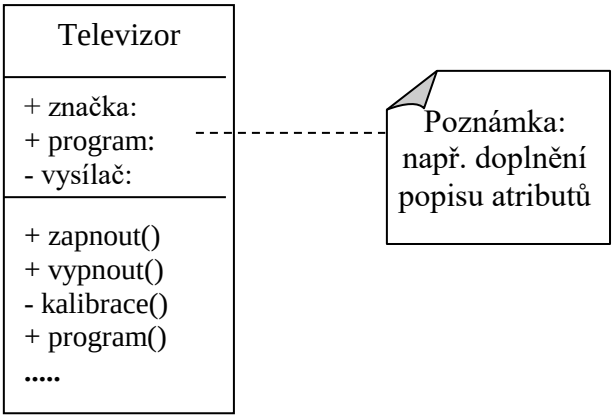
- ukrytí atributů resp. operací uvnitř objektu,
- „neviditelné“ zvenku.



Obrázek: Soukromé a veřejné prvky objektu

Viditelnost atributů a operací (metod):

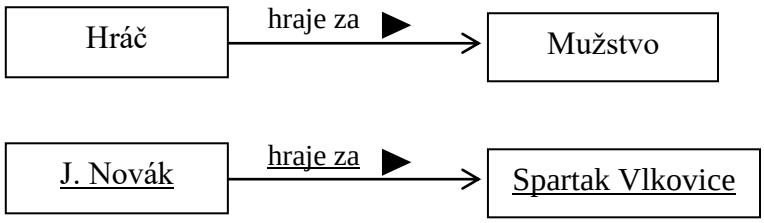
- \+ ... public - viditelné veřejně,
- \- ... private - veřejně neviditelné,
- # ... protected - viditelné pouze pro nejbližší podtřídy dané třídy,
- ~ ... package - viditelné pouze v rámci tříd téhož balíčku,
- viditelnost je implementačně závislá vlastnost (C++, Java, VBasic, C#, ...).



Obrázek: Zobrazení viditelnosti atributů a operací

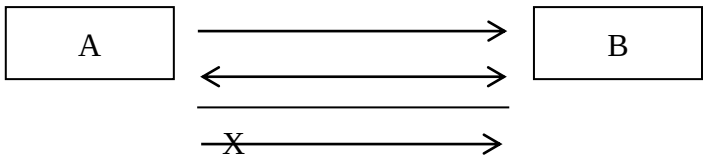
Vazby (relace) mezi třídami a objekty:

- zasílání zpráv - volání veřejné operace jiného objektu,
- mezi třídami - asociace,
- mezi objekty - linky.

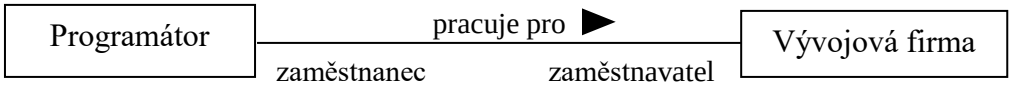


Obrázek: Grafické znázornění vazby mezi třídami a mezi objekty

Znázornění průchodnosti relace:

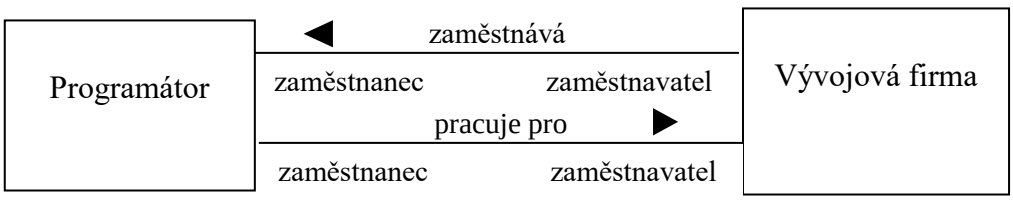


Role v asociaci:



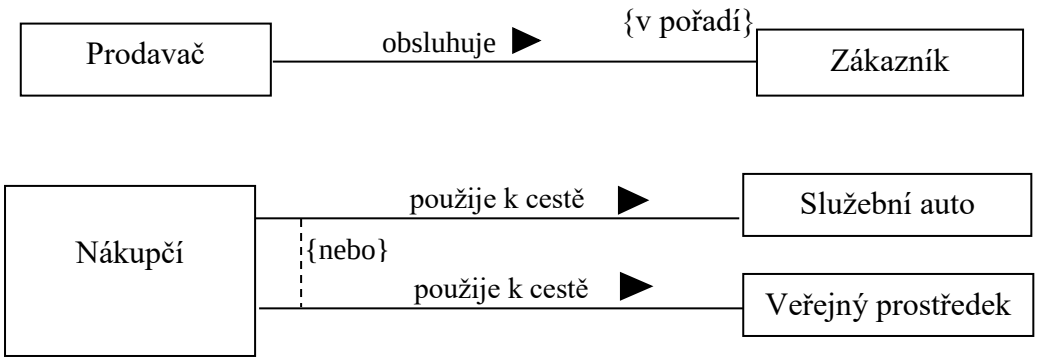
Obrázek: Grafické znázornění rolí v asociaci

Více asociací mezi třídami:



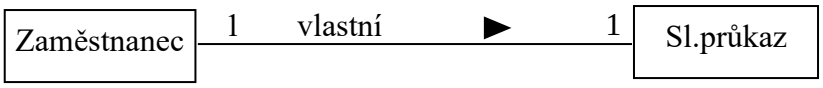
Obrázek: Grafické znázornění dvou asociací mezi třídami

Omezující podmínky v asociaci:



Obrázek: Grafické znázornění omezení v asociaci

Násobnost (kardinalita) asociace:



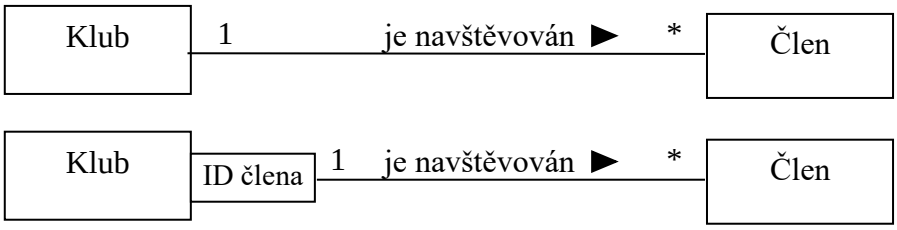
Obrázek: Grafické znázornění vztahu 1:1

1 ----------	*	znamená 1:M
1 ----------	3	znamená 1:3
1 ----------	1..*	znamená 1:1 nebo 1:M
1 ----------	0..*	znamená 1:0 nebo 1:M
1 ----------	0,1	znamená 1:0 (volitelná vazba) nebo 1:1
1 ----------	12..18	znamená 1:12 až 1:18
1 ----------	3..7, 10..12	znamená 1: (3 až 7), nebo 1: (10 až 12)
* ----------	*	znamená M:N

Povinnost, volitelnost vazeb:

- vazba povinná a volitelná,
- viz přehled výše – 1 : 0..*, 1 : 1..*

Kvalifikovaná asociace:



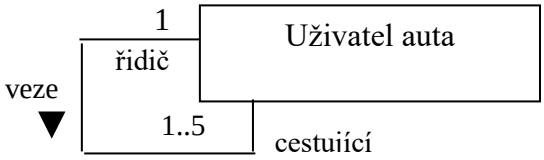
Obrázek: Kvalifikovaná asociace

Kvalifikátory asociací:

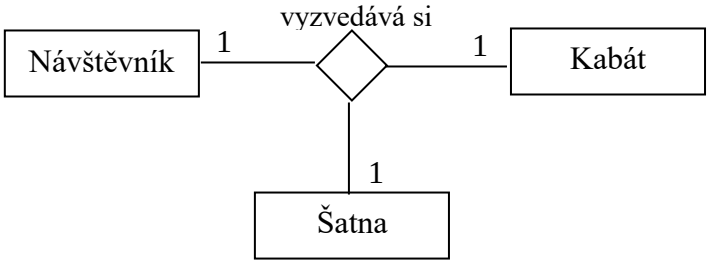
- specifikují jedinečný objekt (z množiny objektů) cílové strany,
- znázorňují způsob vyhledávání konkrétního objektu cílové strany,
- hodnota kvalifikátoru (např. „ID člena“) určuje jednoznačně daný objekt (jednoznačný klíč).

N-ární asociace (stupeň asociace):

- asociace binární – viz ukázky výše,
- asociace unární (reflexivní), ternární.



Obrázek: Unární (reflexivní) asociace



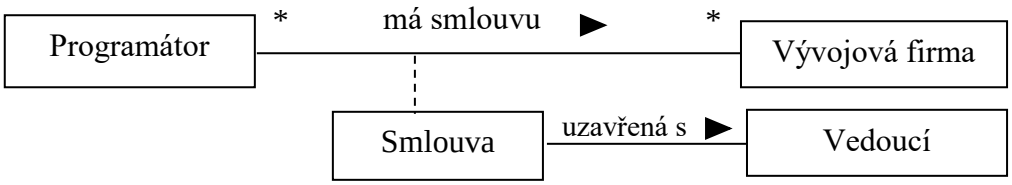
Obrázek: Ternární asociace

Poznámka k n-árním asociacím (pro n>=3):

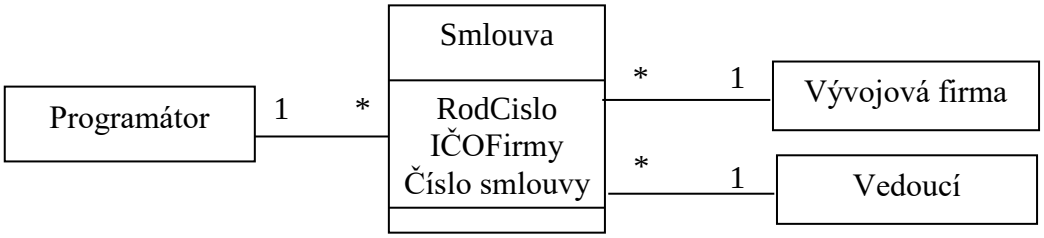
- v diagramu tříd návrhové úrovně - převést na binární asociace.

Asociační třída:

- převádí vazbu M:N na dvě vazby 1:M,
- např. Firma (M) – (N) Osoba,
- na ... Firma (1) – (M) Smlouva (N) – (1) Osoba,
- jde o konstrukci analytické úrovně zobrazení diagramu tříd,
- v diagramu tříd návrhové úrovně – nahradit klasickou třídou.



Obrázek: Grafické znázornění asociační třídy

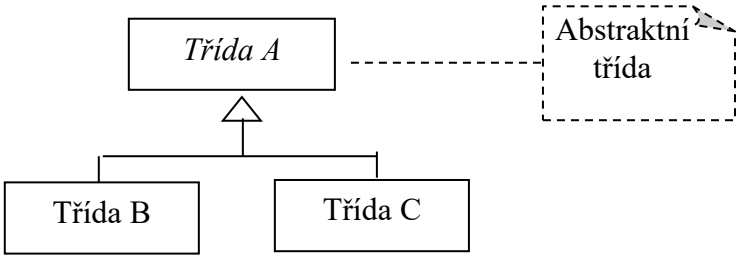


Obrázek: Převod asociační třídy na klasickou třídu a dvě relace

Abstraktní třída:

- obsahuje aspoň jednu abstraktní operaci
  - tj. operaci bez konkrétní implementace,
  - bez algoritmu provedení,
- v tomto smyslu je třída neúplná
  - instance (objekty) takové třídy nelze vytvořit (!),
- slouží pouze k odvozování podtříd (potomků),
  - s tím, že algoritmy operací se popíší až u jednotlivých typů potomků,

- jde o určitý druh dohody mezi rodičem a potomkem,
  - kterou konkrétní potomci musí implementovat,
  - tj. dodržet název operace, parametry a typy jejich hodnot, typ návratové hodnoty, ...
- označení abstraktní třídy
  - název třídy – kurzívou,
  - nebo popisem <<abstract>>.



Obrázek: Grafické znázornění abstraktní třídy "Třída A"

Abstraktní operace třídy:

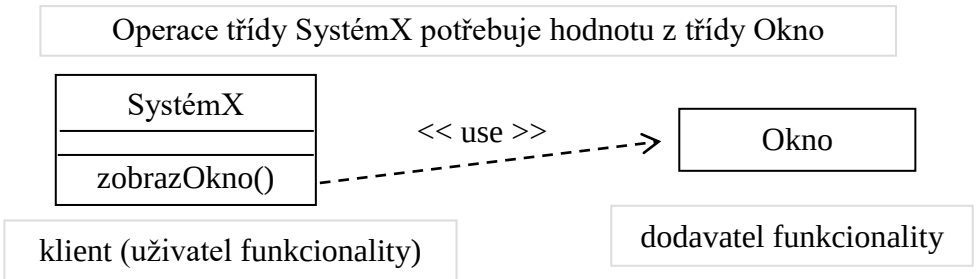
- nemají implementaci, slouží jako „držitelé prostoru“,
- implementace operace je přenechána potomkům třídy,
- označení abstraktní operace – kurzívou,
- např. *nakreslitTvar(), urcitObsah()* ... pro čtverec, kruh, trojúhelník,
- nebo *vypocetUroku()* ... pro různé druhy účtů apod.

Konkrétní třída:

- musí implementovat všechny zděděné abstraktní operace.

Závislost mezi třídami:

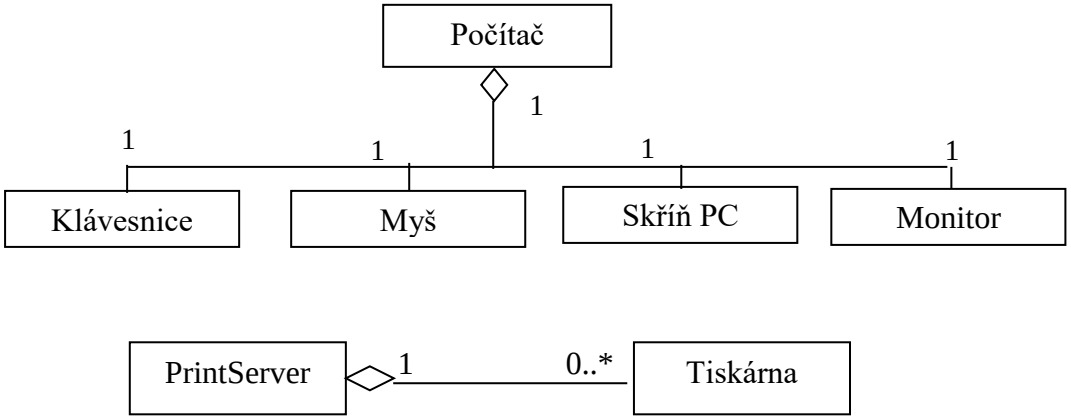
- vazba mezi dvěma prvky, v níž změna jednoho prvku (dodavatele) se promítá do druhého prvku (klienta),



- klient (uživatel dodávané funkcionality) do určité míry závisí na dodavateli,
- základní typy závislosti
  - užití (use) - klient využívá některou službu či hodnotu dodavatele,
  - abstrakce - dodavatel je určitým zobecněním klienta,
  - oprávnění - dodavatel uděluje klientovi oprávnění k něčemu.

Agregace tříd:

- vyjadřuje vztah „je částí“,
- jeden objekt (celek) využívá služeb jiných objektů (svých součástí),
- součást poskytuje služby, reaguje na požadavky celku,
- celek ví o svých součástech, ale součást nemusí vědět nic o celku.

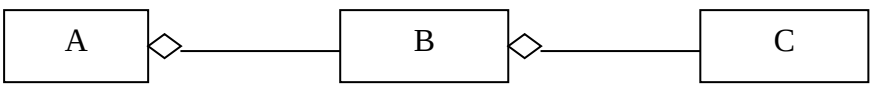


Součást agregace:

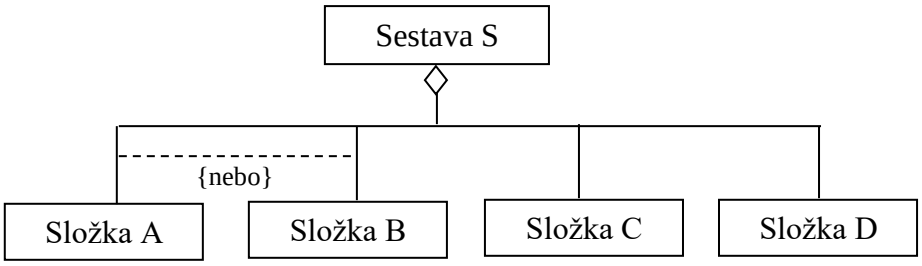
- může existovat ve více celcích,
- může existovat nezávisle na celku,
- objekt nemůže být součástí sama sebe (přímo ani nepřímo).

Tranzitivnost agregace:

- relace agregace je tranzitivní.



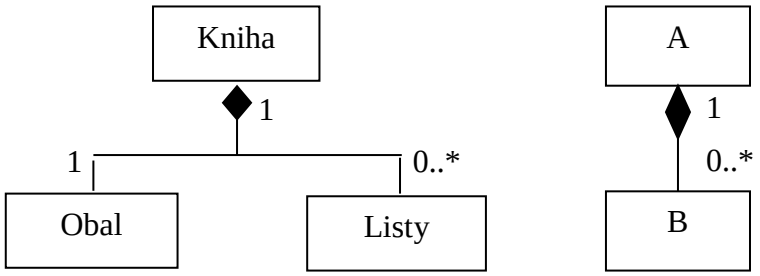
Omezení v agregaci:



Obrázek: Omezení agregace - vztah "nebo"

Kompozice (silná agregace):

- každá součást patří jen jednomu celku,
- součást kompozice nemůže existovat mimo celek,
- např. „strom a jeho listy“,
- celek má výhradní odpovědnost za své součásti,
- je-li celek zrušen
  - musí zrušit i své součásti,
  - nebo je převést k jinému celku.



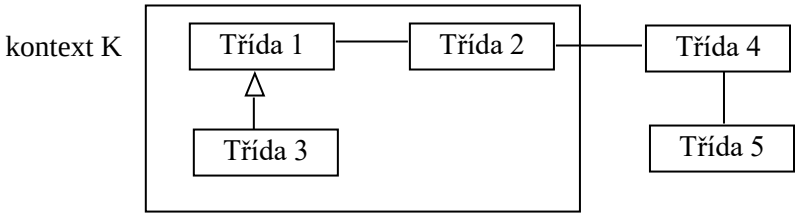
Obrázek: Kompozice - složenina

Kompozice a atributy třídy:

- prvky kompozice v podstatě odpovídají atributům třídy,
- atributy třídy lze chápat jako vztah kompozitní relace,
- doporučení – primitivní nebo jinak nezajímavé kompozitní prvky redukovat na atributy třídy celku.

Kontext (vyznačení kontextu prvků):

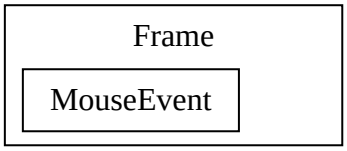
- seskupení logicky souvisejících prvků,
- jde o vyznačení užší logické souvislosti prvků.



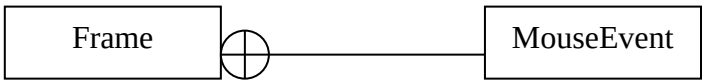
Obrázek: Seskupení tříd – znázornění kontextu

Vnořená třída:

- je deklarována uvnitř jmenného prostoru obklopující třídy,
- instance vnořených tříd mohou být vytvářené instancí vnější třídy,
- využití v oblasti návrhu – jak má být nějaká funkce implementována.



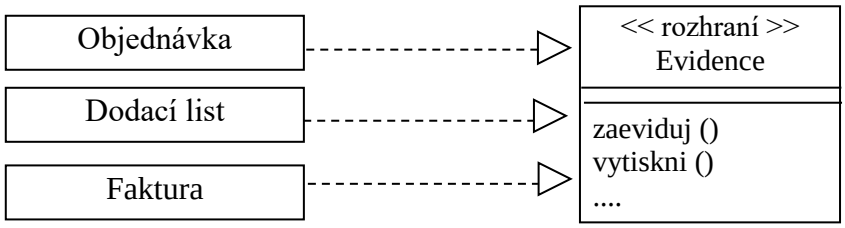
Obrázek: Znázornění vnořené třídy MouseEvent



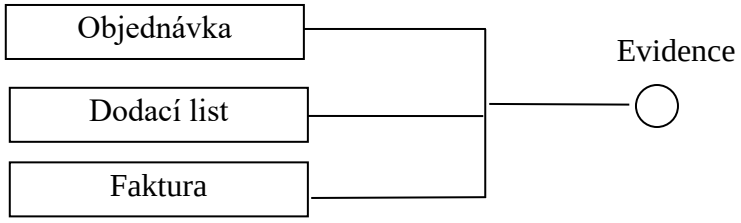
Obrázek: Znázornění vnořené třídy pomocí kotvy

Rozhraní (interface):

- pojmenovaná třída veřejných operací,
- tj. operací (metod), které zpřístupňujeme jiným třídám.



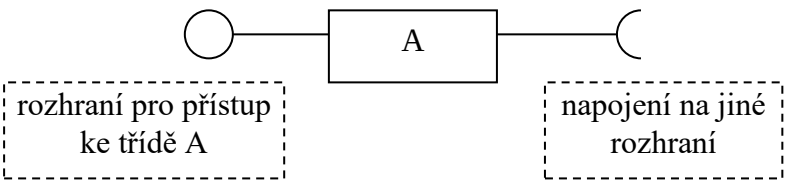
Obrázek: Rozhraní – zápis ve formě třídy



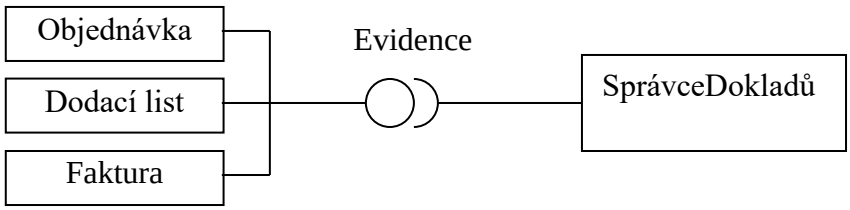
Obrázek: Rozhraní - zápis ve formě „lízátka“

Vlastnosti rozhraní:

- definuje služby poskytované danými třídami,
- jiné třídy využívají těchto operací ve vztazích k daným třídám,
- deklaruje kontrakt k určeným třídám, který má být realizován,
- definuje signaturu a sémantiku operací, jejich omezení, parametry (ale nic více),
  - odděluje tak specifikaci funkčnosti od implementace,
  - nespecifikuje žádnou formu implementace,
- je základem pro komponentovou tvorbu systému,
- pokud jazyk nepodporuje rozhraní – použijte abstraktní třídu.



Obrázek: Zpřístupněné a požadované rozhraní



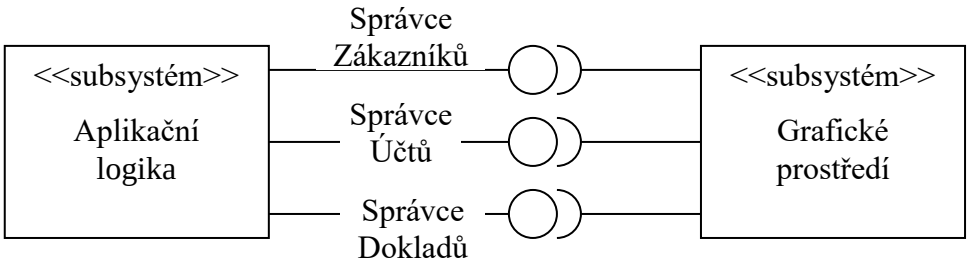
Obrázek: Využití služeb rozhraní

Využití služeb rozhraní:

- třídy Objednávka, Dodací list, Faktura
  - nabízejí služby třídě SprávceDokladů,
- třída SprávceDokladů
  - „rozumí“ protokolu definovanému rozhraním „Evidence“,
  - „ví“, že tyto položky lze zaevidovat, vytisknout, ...

Realizace (implementace) rozhraní:

- realizace – konkrétní vztah mezi třídou a jejím rozhraním,
- jde o implementaci dohody, která je předepsána rozhraním,
- rozhraní musí být realizováno (implementováno) těmi třídami, které dané rozhraní vystavují (v našem příkladu - Objednávka, DodacíList, Faktura).



Obrázek: Vazba subsystémů přes rozhraní

Hledání rozhraní:

- vyčlenit skupiny operací, které by šlo použít i jinde, které se opakují ve více třídách, mají v systému stejnou roli,
- hledat závislosti mezi komponentami – zvážit řešení přes rozhraní.

Přehled probraných vazeb mezi třídami (od nejtěsnější počínaje):

- dědičnost,
- kompozice,
- agregace,
- závislost,
- běžná asociace.

Stereotypy (významové popisy):

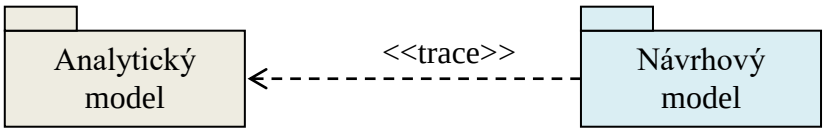
- např. <<rozhraní>>, <<use>>, ...
- slouží k popisu grafických prvků
  - větší názornost,
  - resp. jde o nové prvky vzniklé na základě grafických znaků UML.

Šablony (template):

- umožňují tvorbu parametrizovaných tříd - tříd s parametry,
- např. několik různých tříd se liší pouze v datových typech atributů, v typech návratových hodnot operací apod.,
- pak lze využít třídu s parametry
  - tj. místo uvedení skutečných typů atributů, typů návratových hodnot operací apod.
  - -> uvedeme parametry (zástupné symboly),
- parametry jsou při vytvoření třídy nahrazeny skutečnými hodnotami,
- podporováno pouze v C++.

2. Diagram tříd ve fázi návrhu

Analytický a návrhový model:



Obrázek: Mezi analytickým a návrhovým modelem je vztah závislosti

Poznámka:

- srovnejte s 3-úrovňovou architekturou zobrazení systému,
- koncepční, logická (technologická), fyzická (implementační).

Během fáze návrhu upřesňujeme:

- popisy tříd (adorments – „ozdoby“),
- popisy relací (asociací) mezi třídami (násobnost, role, volitelnost, ...),
- převody:
  - n-ární relace (n>=3) --> binární,
  - asociační třídy --> třídy, ...
- úprava konstrukcí, které programovací jazyk podporuje (resp. nepodporuje).

Doplnění tzv. návrhových tříd:

- fáze tvorby analytických tříd --> fáze tvorby návrhových tříd,
- návrhové třídy lze získat:
  - z problémové domény – např. rozklad analytické třídy na několik podrobných tříd návrhových,
  - z domény řešení - třídy pro uchování mezivýsledků, protokolování akcí uživatelů, ...
- návrhová třída obsahuje:
  - kompletní sadu atributů – název, typ, implicitní hodnoty, viditelnost,
  - převod operací specifikovaných v analytické třídě na úplnou sadu metod.

Zásady pro návrhové třídy:

- úplnost, jednoduchost, soudržnost, minimalizace vazeb na jiné třídy (bez těsných vazeb),
- snažit se o omezení množství relací mezi třídami,
- těsné vazby mezi třídami balíčku jsou v pořádku (soudržnost balíčku).

Upřesnění asociací v návrhovém modelu:

- asociace 1:1
  - vyjadřují silný vztah mezi třídami,
  - mohou vyjadřovat kompozici s vazbou „celek : část“ = „1:1“,
  - uvažujeme i o sloučení do 1 třídy resp. i o atributu třídy,
- asociace M:1
  - mohou vyjadřovat agregaci „celek (M) – součást (1)“,
  - tj. součást může být zahrnuta ve více celcích,
- asociace 1:N
  - možné použití pole,
  - resp. třídy kolekce (obsahují metody pro práci s prvky kolekce),
  - nezdokonalujte zbytečně relace typu 1:N – přenechejte to programátorům,
- asociace typu M:N
  - převod do normálních tříd, agregací, kompozic resp. závislostí,
  - převod na dvě asociace typu 1:N,
- obousměrné asociace
  - převod do dvou jednosměrných (asociací, agregací, kompozic, závislostí),
- asociační třídy
  - nemají podporu v žádném programovacím jazyce,
  - převod do podoby normální třídy,
- kolekce prvků
  - možný popis v jazyku OCL (Object Constraint Language),
  - Bag – unordered, nonunique,
  - Set – unordered, unique,
  - OrderedSet – ordered, unique,
  - Sequence – ordered, nonunique.

Doporučení k informační hodnotě modelu:

- dbát na čitelnost modelu,
- většina diagramů není určena pro ty, kteří je vytvářejí,
- číst model – tak, jak je skutečně napsán (!).

3. K postupům tvorby diagramu tříd

Rámcový postup tvorby diagramu tříd:

1) identifikace základních tříd + základních atributů a metod,
2) průběžná tvorba slovníku dat – popis prvků a jejich významu,
3) identifikace vazeb - asociací, agregací, operací,
4) doplňování - dalších tříd, atributů a metod tříd, vazeb,
5) vytvoření hierarchie tříd,
6) ověřování úplnosti modelu
    - Skutečně řeší tento model požadované úkoly? Nechybí v něm něco? ...
    - obdobně jako např. ověření úplnosti ERD (viz tam),
7) upřesňování modelu pomocí iterací,
8) seskupení tříd do SW modulů.

Metody (techniky) hledání tříd:

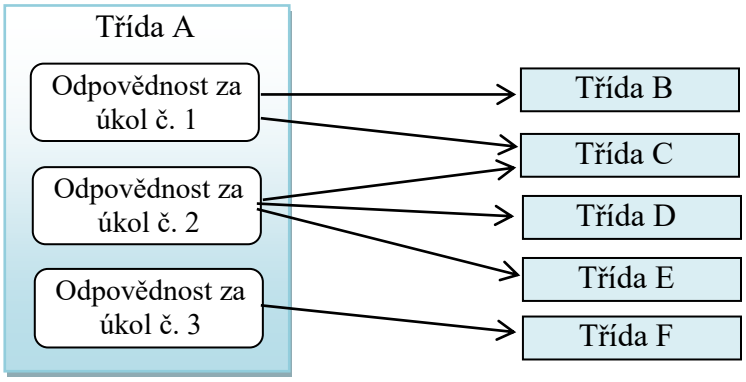
- analýza textu (podstatná jména, slovesa),
    - podstatná jména v zápise zadání – jsou kandidáty na třídy nebo atributy,
    - slovesné fráze – jsou kandidáty na operace nebo asociace,
- odpovědnost tříd (k čemu třída slouží) => vazby mezi třídami,
    - metoda CRC (Class, Responsibilities, Collaborators),
    - metoda RUP – pomocí stereotypů boundary, control, entity.

Štítek metody CRC:

Název třídy: BankovníÚčet	
Odpovědnosti: - udržovat zůstatek -	Spolupracovníci: Banka Referent ....

Pomocný diagram odpovědností tříd:

- tj. znázornění spolupráce třídy A s jinými třídami při plnění úkolů.



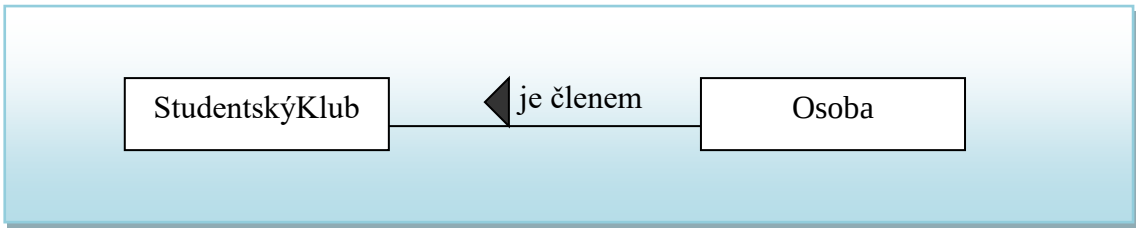
Shrnutí doporučeného postupu tvorby diagramu tříd:

- vycházíme ze zápisu požadavků na vyvíjený IS (ze specifikace IS),
- provádíme tzv. analýzu textu specifikace zadání IS,
- podstatná jména v zápise - představují kandidáty na třídy a atributy,
- slovesné fráze - jsou kandidáty na operace, asociace, agregace,
- souběžně s vytvářením diagramu tříd - i potřebné popisy (slovník dat),
- průběžně doplňujeme popis tříd - další atributy a operace,
- třídy seskupujeme - vytváříme hierarchii tříd,
- tvorba diagramu tříd - probíhá iteračním postupem,
  - celý diagram tříd doplňujeme a upravujeme,
  - i pomocí informací z dalších souběžně vytvářených modelů,
  - až do návrhové úrovně zobrazení,
- ověříme úplnost diagramu tříd
  - „Bude IS vytvořený dle daného modelu řešit požadované úkoly?“

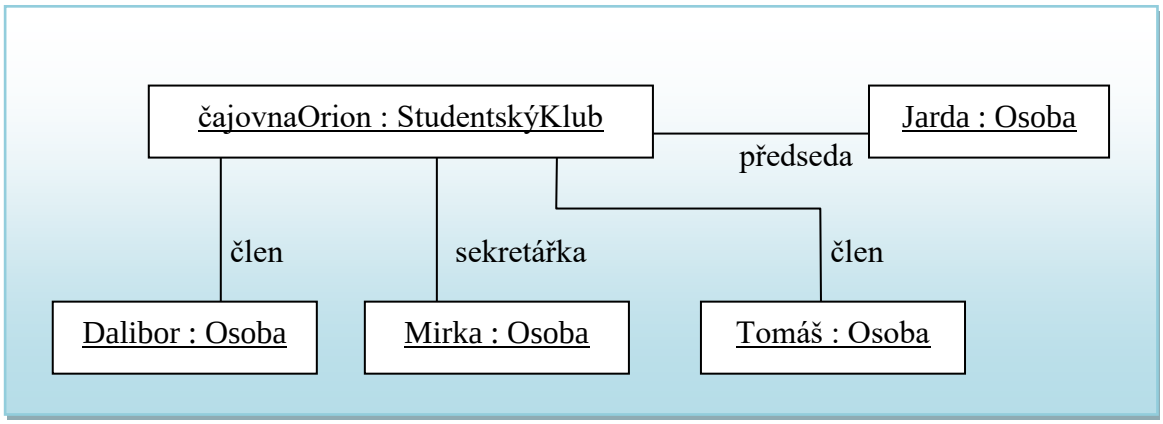
4. Objektový diagram (Object Diagram)

Charakteristika diagramu:

- v podstatě jde o snímek části běžícího systému v konkrétním okamžiku,
- jsou v něm zachyceny aktuální objekty a vazby v daném okamžiku.



Obrázek: Fragment diagramu tříd



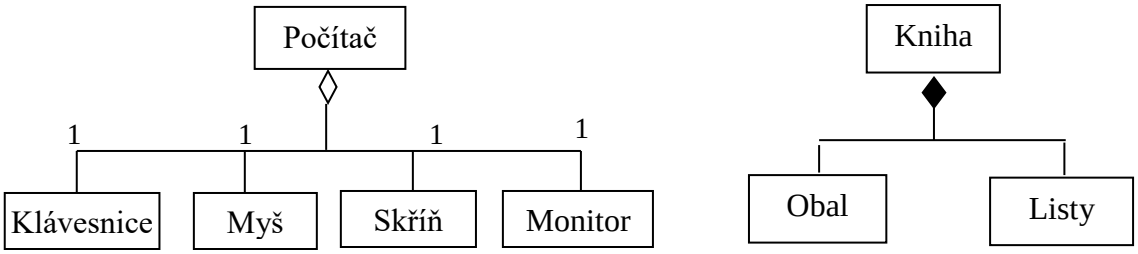
Obrázek: Fragment objektového diagramu

5. Diagram kompozitní struktury (Composite Structure Diagram)

Diagram kompozitní struktury (Composite Structure Diagram):

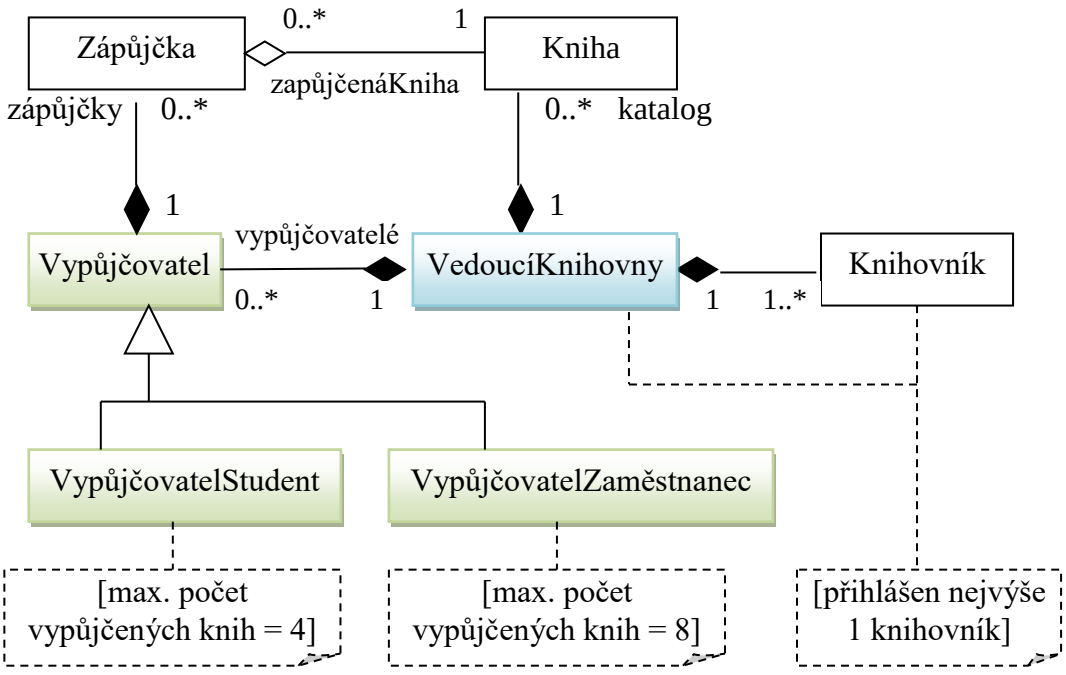
- slouží ke znázornění vnitřní struktury strukturovaných klasifikátorů,
- strukturované klasifikátory - agregace, kompozice, strukturované třídy.

Jednoduchý příklad agregace a kompozice:



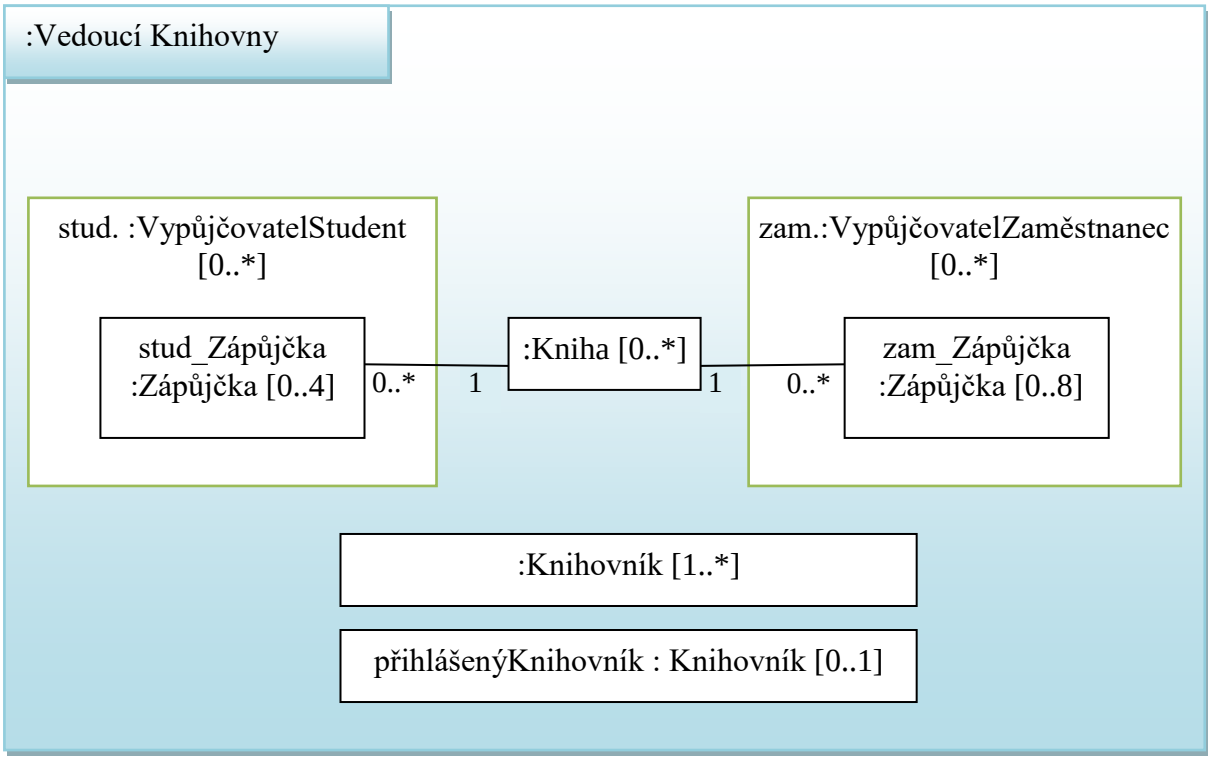
Příklad složitější kompozice:

- zobrazena v Diagramu tříd,
- podle Arlow, Neustadt: UML2 a unifikovaný proces vývoje aplikací.



Příklad složitější kompozice:

- zobrazena Diagramem kompozitní struktury,
- podle Arlow, Neustadt: UML2 a unifikovaný proces vývoje aplikací.



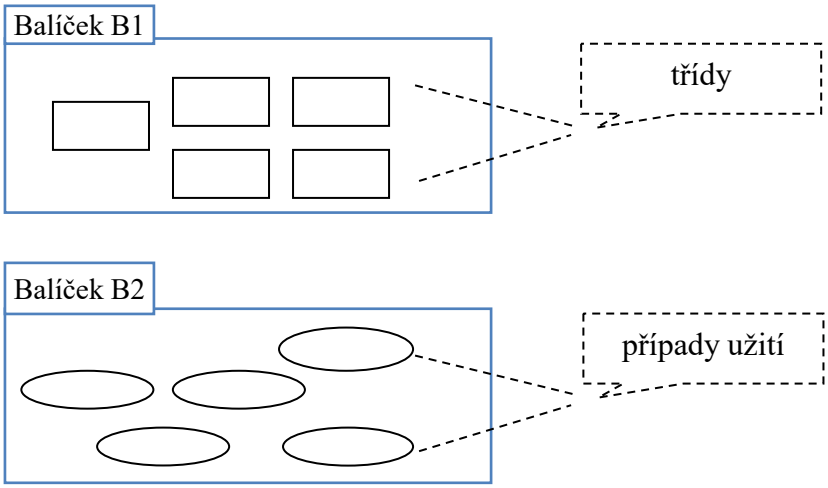
6. Diagram balíčků (Package Diagram)

Balíček (Package):

- mechanismus UML pro logické seskupování prvků podle významu (z hlediska podnikových procesů),
- vytváří vlastní jmenný prostor s jednoznačnými názvy prvků,
- v programovacích jazycích
  - Java ... package,
  - C# ... namespace,
- balíčky mohou vytvářet hierarchii.

Balíček - seskupování prvků:

- balíčky mohou obsahovat (seskupovat) - třídy, případy užití.



Balíček jako jmenný prostor:

- úplný název prvku – včetně „cesty“,
- název_balíčku_1:: název_balíčku_2:: název_prvku

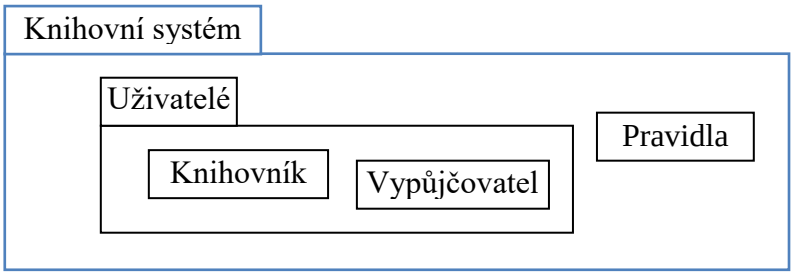
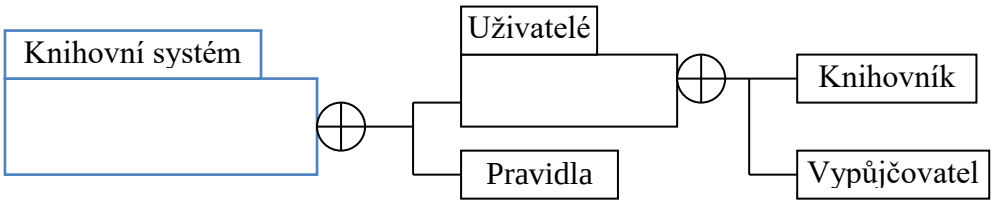


Diagram balíčků (Package Diagram) znázorňuje:

- strukturu vnitřku balíčků,
- a hierarchickou strukturu vazeb mezi balíčky.

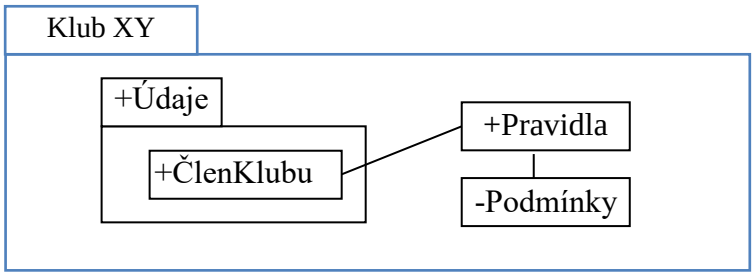
Balíčky vnořené:

- je vhodné použít symbol kotvy.



Viditelnost prvků:

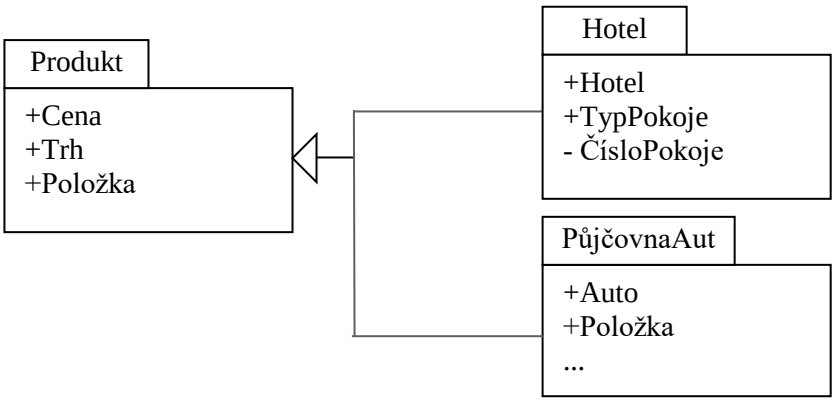
- určení, zda prvky jsou viditelné vně balíčku,



- prvek vnitřního balíčku
  - vidí veřejné členy vnějšího balíčku,
- prvek vnějšího balíčku
  - nevidí žádné členy vnitřního balíčku - pokud na nich není závislý pomocí <<access>>, <<include>>.

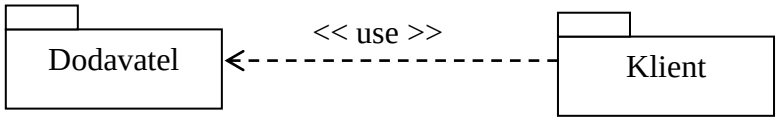
Hierarchie (zobecňování) balíčků:

- odvozené balíčky dědí prvky od svých předků,
- mohou překrývat rodičovské prvky,
- obdobně, jako dědění tříd.



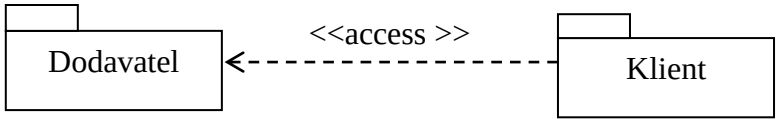
Závislost balíčků:

- existuje několik typů závislostí mezi balíčky,
  - např. <<use>>, <<import>>, <<access>>, ...
- <<use>>
  - prvek klientského balíčku používá veřejný prvek z dodavatelského balíčku (bez bližší specifikace závislosti),



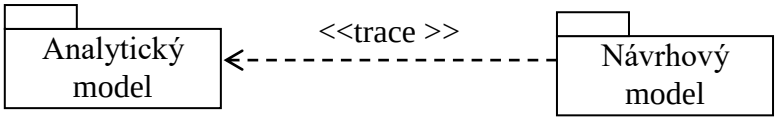
- <<import>>
  - sloučení jmenných prostorů,
  - veřejné prvky dodavatele se stanou součástí (veřejnými prvky) jmenného prostoru klienta,

- <<access>>
  - klient má přístup k veřejným prvkům dodavatele,
  - jmenné prostory zůstávají oddělené,



- <<trace>>
  - znázornění historického vývoje, klient je vyšším vývojovým stupněm (vyšší verzí) dodavatele – v podstatě jde o relaci mezi modely,

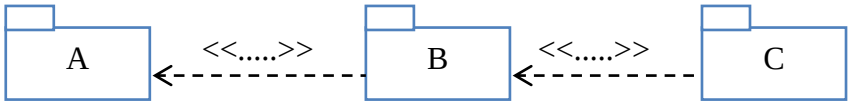
- např. analytický model <--- návrhový model,



- <<merge>>
  - sloučení - obsah klientského balíčku je rozšířen o obsah dodavatele,
  - prvky stejného názvu jsou sloučeny (v tzv. rozšířené klientské prvky),
  - použití v meta-modelování (modely modelů pro danou řešenou oblast).

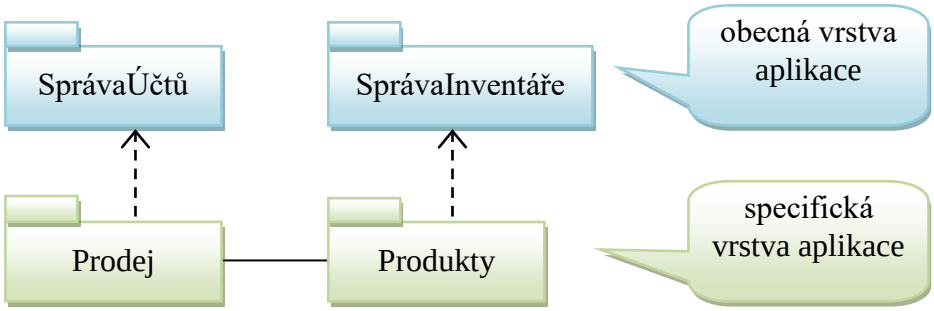
Tranzitivita závislostí:

- relace založené na <<import>> jsou tranzitivní,
- relace založené na <<access>> nejsou tranzitivní.



Proces tvorby balíčků:

- proces tzv. architektonické analýzy,
- proces, kterým dělíme logicky související třídy do balíčků,
- následně balíčky rozdělujeme do vrstev (hierarchie),
- cílem je minimalizace vazeb uvnitř modelovaného systému, minimalizace závislostí mezi balíčky,
- viz obrázek – balíčky obecné vrstvy (SprávaÚčtů, SprávaInventáře) lze používat v několika různých aplikacích.



Obrázek: Architektonická analýza

Poznámky a doporučení:

- analytické balíčky vytváříme až po určité době vývoje diagramu tříd
  - po jeho dozrání, usazení,
- balíčky sdružují sémanticky příbuzné prvky, soudržná seskupení tříd,
- zdroje informací pro tvorbu balíčků jsou
  - model případů užití (Use Case diagram)
  - a podnikový (business) proces,
- případ užití (funkcionalita systému)
  - může mířit skrz několik balíčků,
- dbát na vysokou soudržnost jednotlivých balíčků
  - snaha o minimalizaci vazeb mezi balíčky,
  - přesouvat třídy, přidávat a odstraňovat balíčky,
- vyhýbat se cyklickým závislostem mezi balíčky
  - raději pak sloučit balíčky, vytvořit balíček se společnými prvky,
- udržovat přehledný a srozumitelný model
  - omezovat zbytečné vnořování,
  - „rozumný“ počet tříd v balíčku.