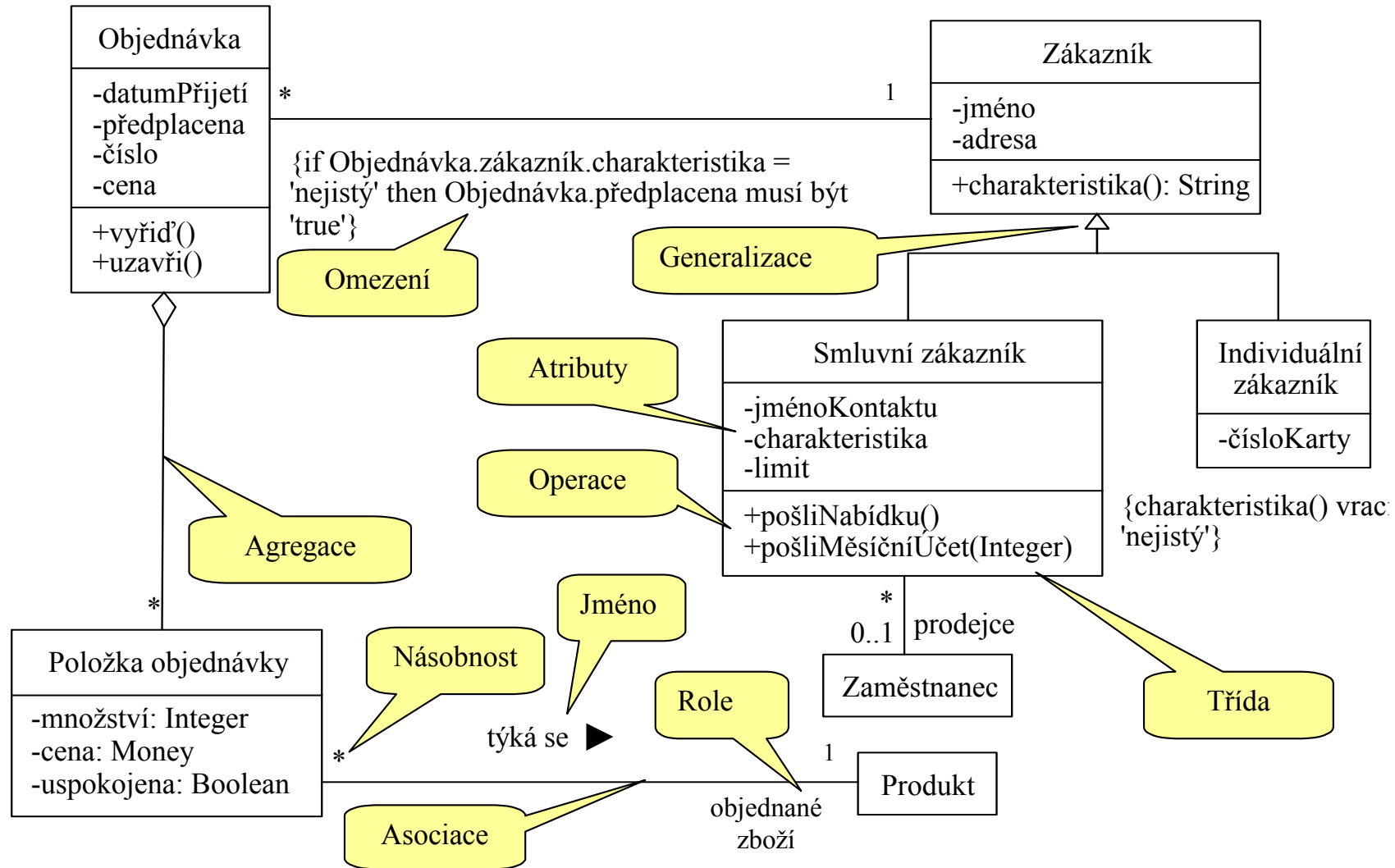


7.3 Diagramy tříd - základy

- popisuje typy objektů a statické vztahy mezi nimi



- **Tři pohledy na systém při použití diagramu tříd**
 - ◇ ***Konceptuální – koncepty aplikační domény, bez vztahu k implementaci, jazykově nezávislá.***
 - ◇ ***Specifikační – pohled na program, specifikace rozhraní bez specifikace implementace.***
 - ◇ ***Implementační – je vidět implementace***
- hranice nejsou ostré, UML podporuje všechny tři pohledy
- **Asociace**
 - reprezentuje statické vztahy (spojení) mezi instancemi tříd
- Př) Položka objednávky - Produkt**
- **Konceptuální pohled** - reprezentuje konceptuální vztah mezi třídami
- **vlastnosti asociace**
 - ◇ ***stupeň asociace –binární (včetně reflexivní), ternární, ...***
 - ◇ ***zakončení (role)***
 - ***jméno role***
 - ***násobnost (kardinalita) – dolní a horní mez počtu objektů, které mohou být ve vztahu daného typu s jedním objektem protější***

**třídy – nejčastější případy: 1, 0..1, * (0..nekonečno, M (many)),
– obecně seznam celých čísel a rozsahů**

Př) Student – navštěvuje – Předmět – garantuje – Ústav

- Specifikační pohled - reprezentuje zodpovědnost (za dostupnost informace)

Př) Co navštěvuje daný student, kdo navštěvuje daný předmět, kdo garantuje předmět

- existují-li konvence pro pojmenování takových operací, lze usoudit, jak bude vypadat rozhraní

**Př) class Predmet {
 public Set getStudent();
 public Ustav getUstav();
 }**

- rovněž zodpovědnost za aktualizaci vztahu

Př) Zápis studenta do předmětu, zrušení registrace

- zodpovědnost neimplikuje datovou strukturu pro implementaci (jen rozhraní)

Př) ukazatel, OID, množina ukazatelů, množina OID, dotazovací funkce, ...

- **Implementační pohled** – „vidí“ implementaci

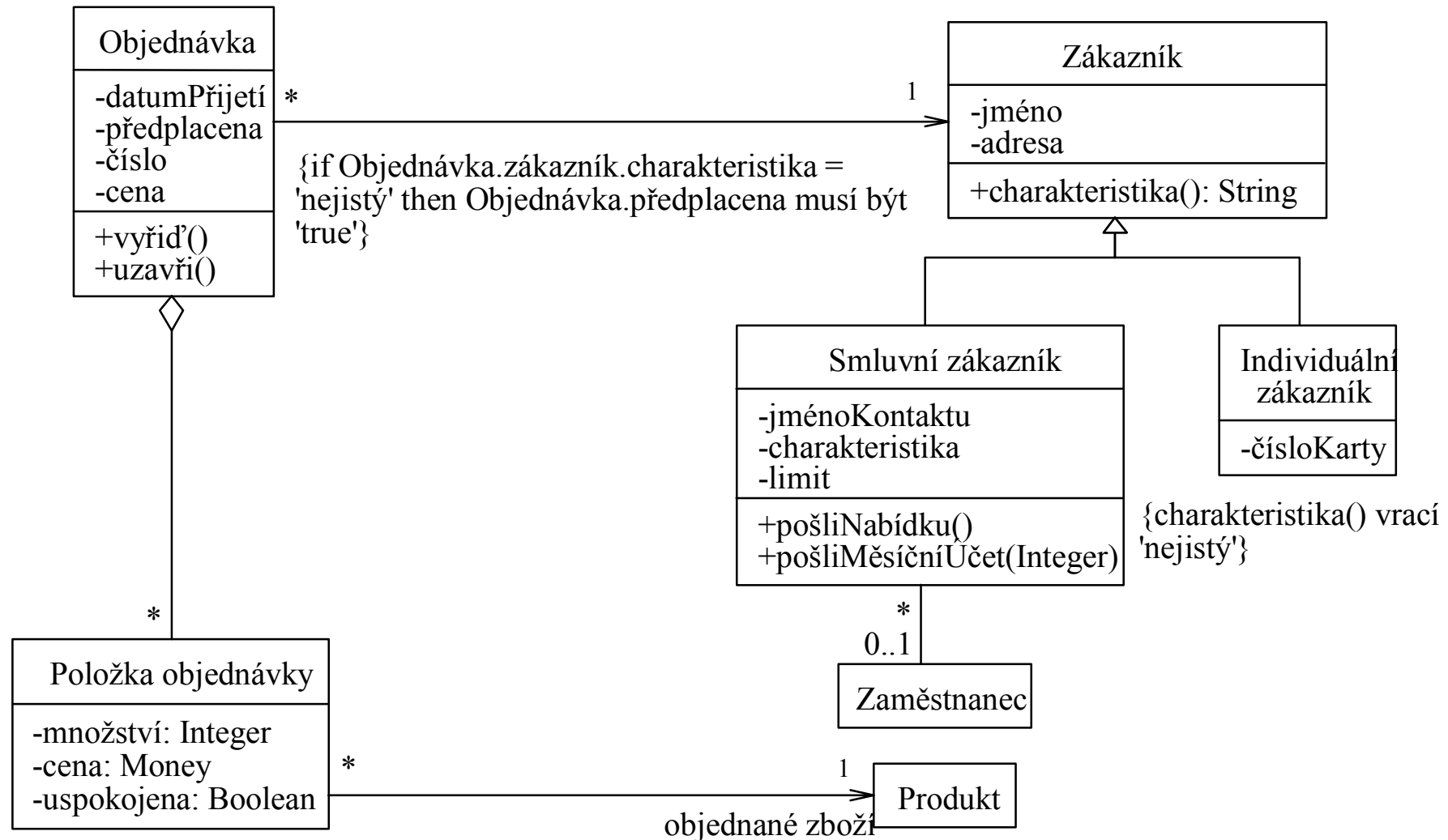
```
Př) class Predmet {  
        private Set Studenti;  
    .... }  
    class Student {  
        private Set Predmety;  
    .... }
```

- asociace může být orientovaná - ukazuje *navigovatelnost* (jednosměrná/obousměrná asociace)

Př) viz další strana

- význam pro specifikační (zodpovědnost) a implementační diagramy
- podle UML neorientovaná asociace znamená buď obousměrnou nebo „nerozhodnutá“ – potřeba dohodnout význam
- omezení plynoucí z obousměrné asociace (dvě inverzní navigace)
- pojmenování asociací:
 - sloveso
 - jméno role

- není nutné pojmenovávat vždy (srozumitelnost), zavedení konvencí (jméno role je jméno třídy)
- asociace je statickým vztahem (existuje po celou dobu života objektů), zasílání zpráv a závislost se modelují jiným způsobem



- **Atributy**

- vlastnost, jejíž hodnotu objekt nese

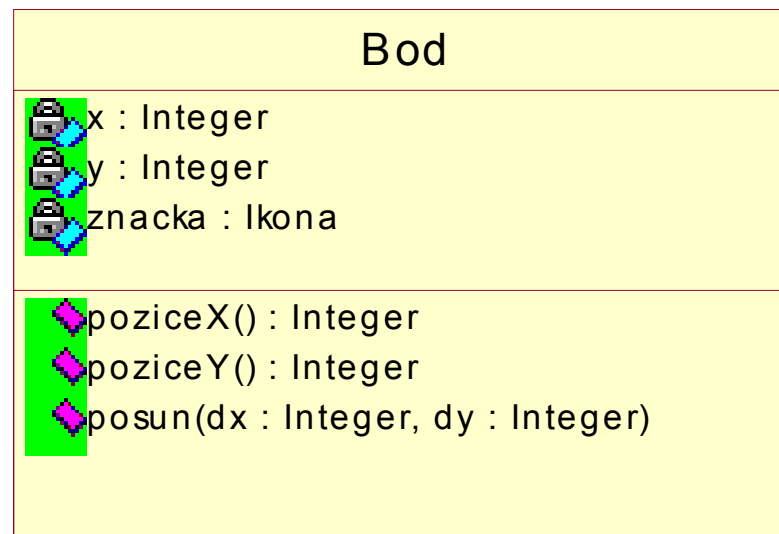
- Př) Student - jméno**

- Konceptuální pohled – student má jméno
- Specifikační pohled – objekt třídy Student má přístupnou hodnotu jména
- Implementační pohled – každý objekt třídy student má pole (instanční proměnná, datový člen) pro uložení jména
- syntax podle UML (v závislosti na míře detailů):
viditelnost jméno: typ = implicitní_hodnota
- rozdíl mezi asociací a atributem?
 - konceptuální – různá notace pro totéž
 - specifikační a implementační – atribut v sobě zahrnuje navigaci, kopie objektu – hodnota, ne reference

- **Operace**

- procesy, které umí třída provádět
- syntax podle UML (v závislosti na míře detailů):
viditelnost jméno (parametry): návrat_hodnoty
 - *viditelnost*: public (+), protected (#), private (-), případně další

- **parametry: seznam**
 směr jméno: typ = implicitní_hodnota
- **návrat_hodnoty: seznam návratových hodnot (zpravidla jedna)**
- **Typy operací:**
 - **dotaz, zpřístupňující (getting)– nemění stav**
 - **modifikátor, nastavující (setting) – může měnit stav**
- **vztah operace – metoda (vlastní terminologie programovacích jazyků (C++: členská funkce))**
- **operace a atributy třídy (class scope) - podtržené**



- **Generalizace**

- třídy s řadou společných vlastností a některými odlišnými → zobecňující třída

Př) Smluvní zákazník, Individuální zákazník → Zákazník

- Konceptuální pohled: všechny instance podtypu jsou také instancemi nadtypu (s danými atributy, operacemi, asociacemi).
- Specifikační pohled: rozhraní podtypu musí zahrnovat všechny prvky rozhraní nadtypu.

Princip nahraditelnosti (substituability): všude, kde se počítá s objektem nadtypu, lze použít objekt podtypu (není třeba si dělat starosti s rozdíly – rozhraní je stejné).

- Implementační pohled: souvisí s dědičností v programovacích jazycích - podtřída dědí všechny metody a pole atributů, metody může předefinovat.

- **Omezení (constraint)**

- důležitá úloha diagramu
- řada omezení vyjádřena prvky diagramu (asociace – násobnost, atribut podtypu – odlišnost)
- notace pro obecná omezení v UML: *{popis_omezení}*

- slovní popis, formule, OCL, ...
 - **Kdy použít diagramy tříd?**
 - základ většiny OO metod
 - problém bohatosti výrazových prostředků
- Tipy:**
- nesnažit se použít veškerou dostupnou notaci
 - přizpůsobit pohled etapě projektu:
 - analýza: konceptuální pohled
 - návrh: specifikační + implementační
 - koncentrovat se na klíčové oblasti, raději méně aktuálních diagramů
 - nezabřednout brzy do implementačních detailů