Dijagrami klasa

Dijagram klasa

- Jedan od najviše korišćenih tipova UML dijagrama

- Prikazuje: **tipove** objekata i **statičke veze** među njima

- 2 vrste veza: asocijacije i podtipovi
  - Primjer asocijacije: *Klijent* u banci ima više *Računa*
  - Primjer podtipa: *ŠtedniRačun* je podtip klase *Račun*

Dijagram klasa

- Šta još prikazuje klasni dijagram ?

- Za svaku klasu:
  - Atributi
  - Operacije

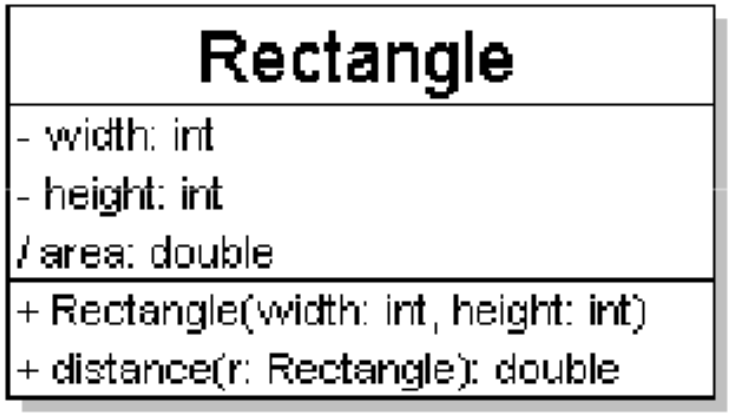
- Ograničenja o tome kako su objekti povezani među sobom
  - Npr.:
    - *Klijent* u banci može imati **više** *Računa*
    - *Klijent* u banci može imati **najviše jedan** *TekućiRačun*

Šta možemo modelovati dijagramom klasa?

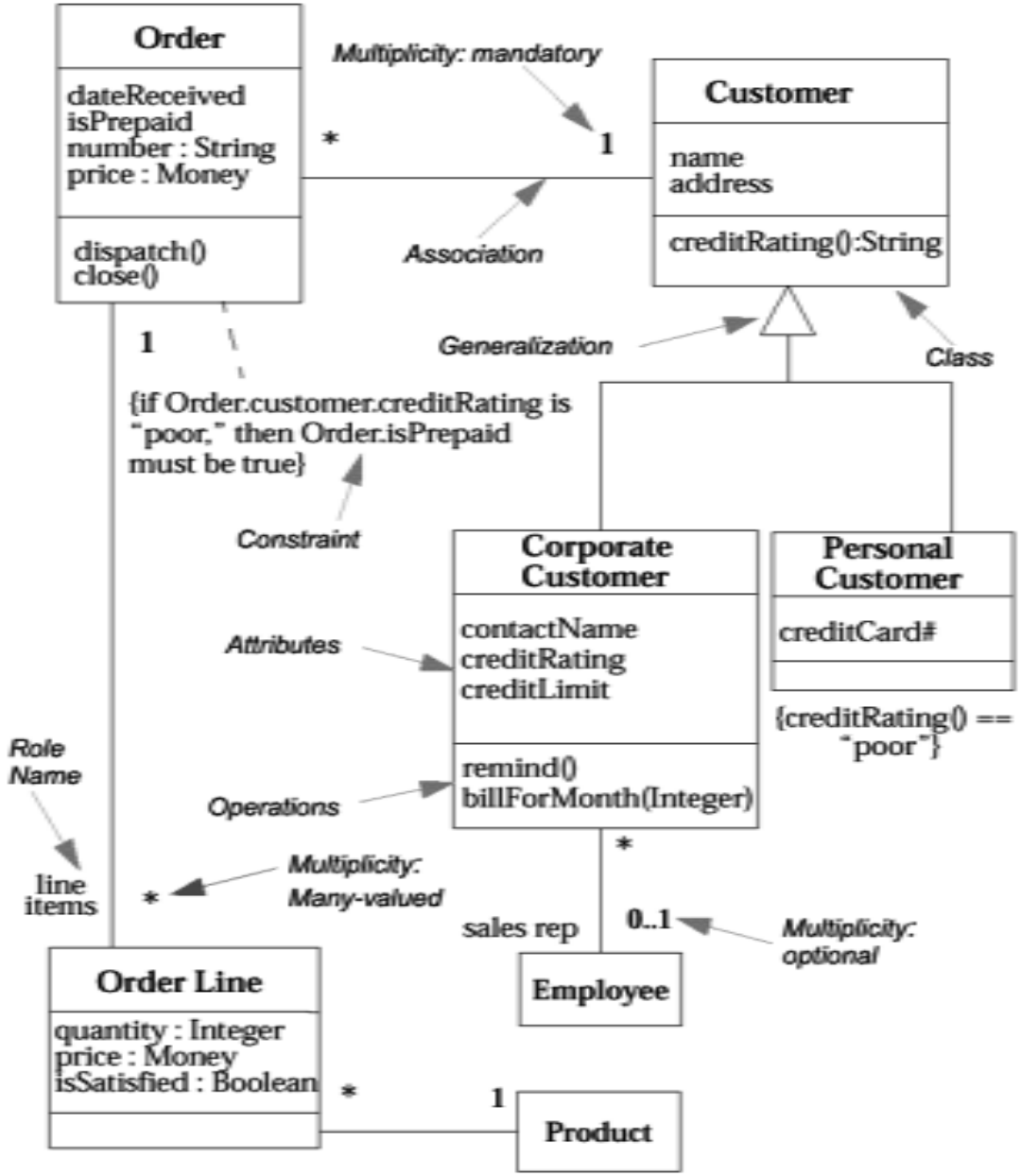
- Perspektive:
  1. Konceptualna, stereotip *<<type>>*
     - **koncepti domena** – često se mapiraju na klase programa ali ne nužno
     - Model nezavisan od jezika
  2. Specifikacija, stereotip *<<type>>*
     - Nivo softvera ali samo **interfejsa** ili **tipa** (još ne implementacije)
     - Ključ dobrog OO dizajna je odvajanje interfejsa od implementacije
  3. Implementaciona, stereotip *<<implementation class>>*
     - Konkretne klase programa i preciziranje koja klasa implementira koje metode

- Granice između perspektiva nisu uvijek jasne
  - posebno važno praviti razliku između 2. i 3. perspektive

Primjer klase



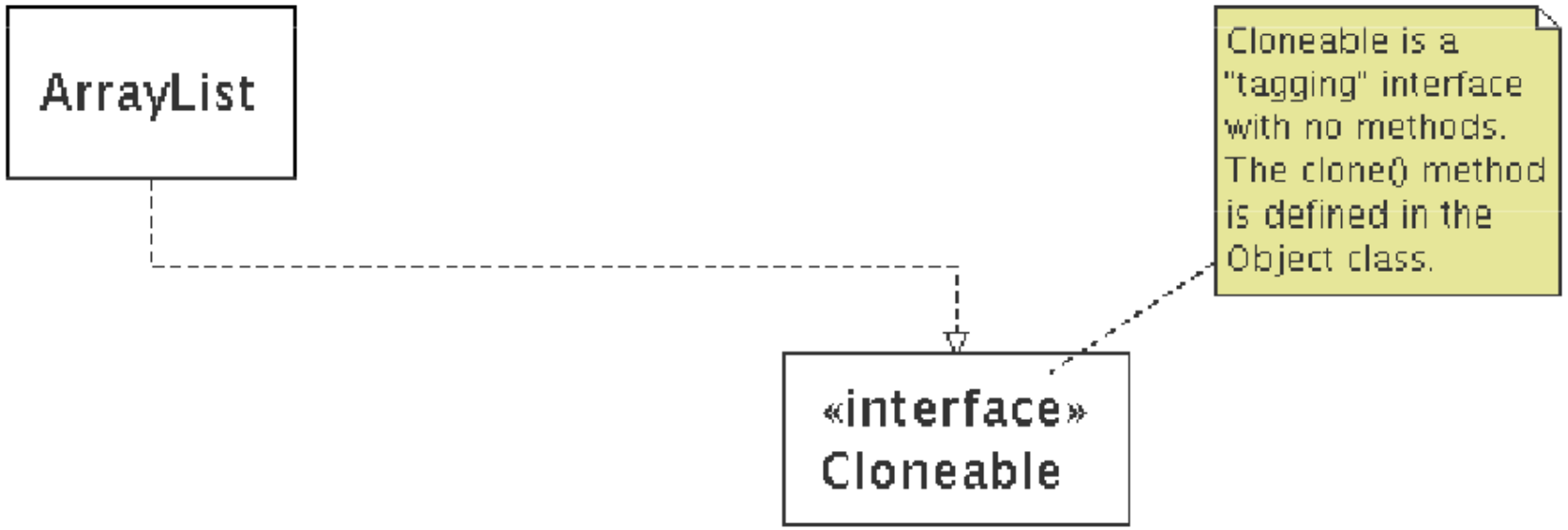
Primjer dijagrama klasa



- brz pregled primjera
- vratićemo se na svaki element

•ime klase počinje velikim slovom
•stereotip <<interface>>
•italic font za abstraktnu klasu

Komentari


ArrayList
«interface»
Cloneable
Cloneable is a "tagging" interface with no methods. The clone() method is defined in the Object class.

Asocijacija

- Veza između **instanci** klasa
  - primjer: *Osoba* radi za najviše jednu *Kompaniju*
  - primjer: *Kompanija* ima više *Kancelarija*

- Primjer na našem dijagramu:
  - *Narudžba* (*Order*) stiže od **jednog** *Kupca* (*Customer*)
  - *Kupac* može napraviti **više** *Narudžbi*
  - Svaka *Narudžba* sadrži **više** *Linija* (*Order line*)
  - Svaka *Linija* se odnosi na **jedan** *Proizvod* (*Product*)

Asocijacija

- Asocijacija ima dva kraja
- Svaki kraj odgovara jednoj klasi
- Kraj može biti označen oznakom (labelom) ili **imenom uloge** (*role name* ili *role*)
  - Npr.: Kraj vezan za klasu *Order line* ima ime *lineitems*
  - Ako nema eksplicitnog imena, može se uvijek nazvati po klasi na čijem je kraju
    - Npr. kraj vezan za klasu *Customer* u asocijaciji *Customer-Order*, možemo nazvati *customer*

Asocijacija

- **<u>Multiplikativnost</u>**
  - Važan koncept u dijagramu klasa
  - Specificira koliko objekata jedna klase učestvuje u asocijaciji
  - Znak * znači „nula ili više“
  - Znak 1 znači „tačno jedan“
  - Znaci 0..1 znače „nula ili jedan“

Najčešće korišćene multiplikativnosti

  - Moguće precizirati i specifične multiplikativnosti:
    - 11 - „tačno 11“ za broj igrača u fudbalskom timu
    - 2..4 - „od 2 do 4“
    - 2,4 - „ili 2 ili 4“ npr. za broj vrata u automobilu
  - Primjer: *Order-Customer*

Asocijacija

- U perspektivi *specifikacije*, asocijacije predstavljaju **odgovornosti** – bez strukture klase
- Npr. iz dijagrama znamo
  - klasa *Customer* može da vrati informaciju o narudžbama
  - moguće da će postojati metod ili više njih u klasi *Customer* koji vraćaju tu informaciju
  - Isto je od klase *Order* prema klasi *Customer* i prema klasi *Order Line*

Asocijacija

- Sljedeće pitanje: koja od dvije klase je odgovorna za ažuriranje veze ?
  - *Order::Order(Customer *c)* ili
  - *Customer::addOrder(Order *o)*

  - Da li klasa *Order* sadrži pokazivač na *Customer* ili vraća informaciju tako što se obraća svim objektima klase *Customer*?

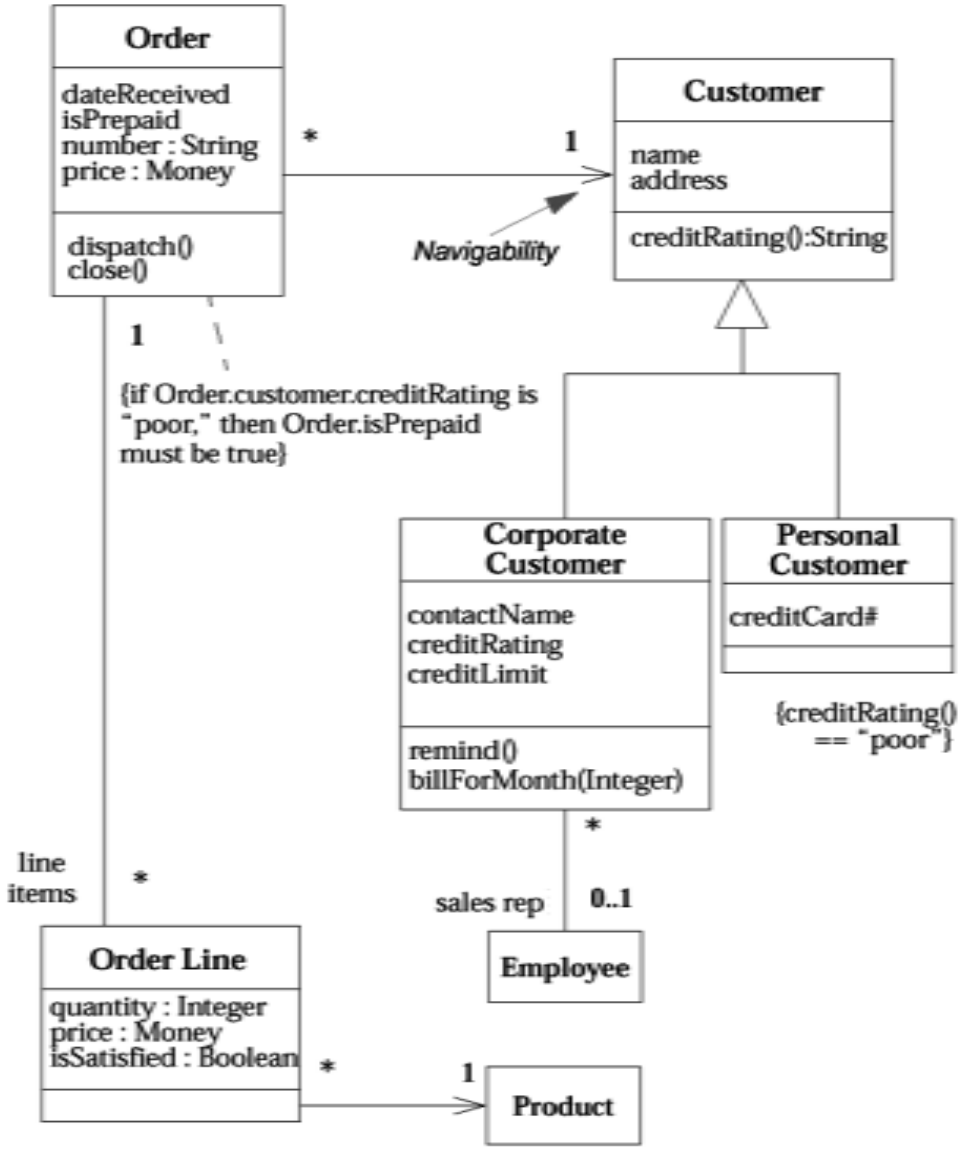
  - Za ovo služi blok za metode (operations)
  - Ušli smo u perspektivu **implementacije**

Asocijacija

- Perspektiva **implementacije**
  - Pokazivači na obje strane *Order-Customer*
  - Order sadrži kolekciju (niz/listu...) pokazivača na klasu Order Line

```
class Order {
      private Customer _customer;
      private Set _orderLines;
class Customer {
      private Set _orders;
```

Smjer asocijacije (*navigability*)



- **Jednosmjerna** (unidirectional) i **dvosmjerna** (bidirectional) asocijacija
- Strelice ukazuju da postoji jednosmjerna...
  - odgovornost - u slučaju perspektive specifikacije
  - referenca (pokazivač) – u slučaju perspektive implementacije
- Kada nema strelica, podrazumijevaju se strelice na obje strane
- Bidirekciona asocijacija proizvodi ograničenje: konzistentnost veze

Imenovanje asocijacije

- Asocijaciju možemo imenovati
  - obično glagolom (npr. *„contains“* između *Order* i *Order line*)
  - imenovati samo onda kada ime doprinosi razumijevanju (zbog čitljivosti dijagrama)

Atributi

- Primjer: *Customer* i atribut *name*
- *Konceptualna* perspektiva
  - Kupci imaju imena
- Perspektiva *specifikacije*
  - Objekat klase *Customer* nam može reći svoje ime i pruža nam način da ga setujemo
- Perspektiva *implementacije*
  - Klasa *Customer* ima polje (podatak član) koje se zove *name*

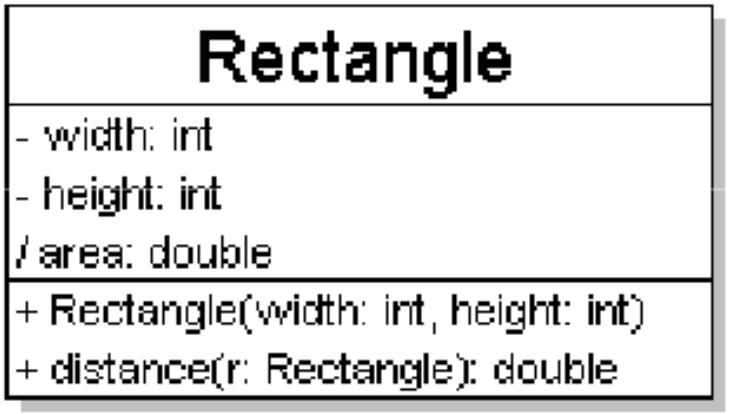
Atributi

visibility name: type = defaultValue

- Za svaki atribut možemo prikazati:
  - ime
  - tip
  - podrazumijevanu vrijednost
  - vidljivost
    - + za *public*
    - - za *private*
    - # za *protected*
    - */ izvedeni atribut*
  - underline – statički atributi klase
  - Moguće navesti i multiplikativnost:

    dateReceived[0..1] : Date

  - Primjer: *-balance : double = 0.0*



Atributi

- Iz konceptualne perspektive, nema razlike sa asocijacijom
- Razlika se javlja samo u perspektivi specifikacije i implementacije:
  - smjer samo od klase ka atributu
  - Nema referenci -> klasa sadrži atribut po vrijednosti
  - Atributi obično jednostavne klase (*string, date*) ili prosti tipovi kao *int, real*

Operacije

- Procesi koje klasa može da izvrši
- Na nivou specifikacije, prikazuju se javne metode klase
- Na nivou implementacije, prikazuju se i privatne i zaštićene (*protected*) metode

Operacije

- Primjer: *+distance(p1: Point, p2: Point): double*

- Puna UML sintaksa za operacije:

  visibility name (parameter-list) : return-type-expression {property-string}

- vidljivost:
  - + za *public*
  - - za *private*
  - # za *protected*
- name – ime operacije

Operacije

- Parameter-list su argumenti odvojeni zarezima a sintaksa je slična atributima:

direction *name: type = default value.*

- Jedini novi element je *direction* koji može biti *in*, *out* ili *inout*
  - ako je izostavljen, podrazumijeva se *in*
- *return-type-expression* su izlazni parametri operacije odvojeni zarezima
- underline: statičke metode
- Primjer:

+ balanceOn (date: Date) : Money.

Operacije

- Primjer: *+ distance(p1: Point, p2: Point): double*

- Property-string su neke osobine koje važe za operaciju
  - sa {**query**} označavamo operacije koje ne mijenjaju stanje objekta
    - mogu biti izvšavane u bilo kom redosljedu
  - operacije koje mijenjaju stanje objekta – **modifiers**
    - redosljed izvršavanja je važan
  - **getting** metode – čitaju vrijednosti atributa i ne rade ništa više
  - **setting** metode – postavljaju vrijednosti atributa i ne rade ništa više

Operacije

- Razlika između operacije i metode
  - Često se koriste kao sinonimi
  - Ali nekada ima razlike: operacija je deklaracija a metoda je implementacija
    - Posebno važno kod nasljeđivanja (jedna deklaracija i više implementacija)

Generalizacija

- Primjer u našem dijagramu: fizičko i pravno lice (*Personal Customer* i *Corporate Customer*)
  - Sve zajedničko se smješta u opštu klasu ili **super-tip** (super-klasu) ili parent-class: *Customer*
  - *Personal Customer* i *Corporate Customer* su **podtipovi**

Generalizacija

- Konceptualni nivo
  - Svaki *Personal Customer* je takođe i *Customer*
    - Svi **atributi**, sve **operacije** i sve **asocijacije** koje važe za *Customer* tip, važe i za *Personal Customer*
- Nivo specifikacije
  - Interfejs nadtipa mora biti podskup interfejsa podtipa

- Princip „zamjenjivosti“ (**substitutability**): ukoliko u bilo kom kodu koji zahtijeva objekte klase *Customer*, zamijenimo te objekte sa objektima klase *Personal Customer*, takav kod bi trebalo da radi bez problema.
  - *subtyping*

Generalizacija

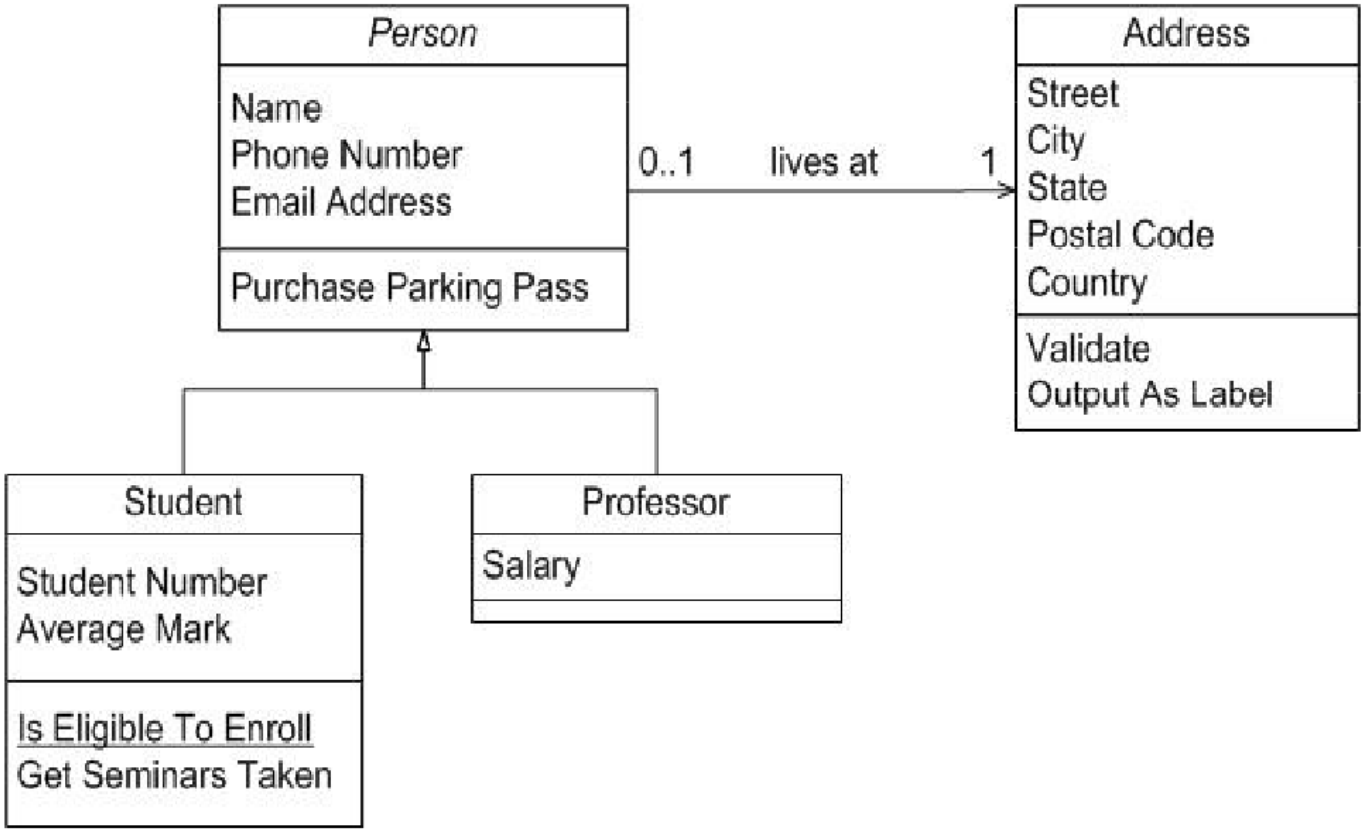
- Nivo implementacije
  - *subclassing*: podklasa nasljeđuje sve metode i podatke članove nadklase i može de re-implementira metode.

- *subclassing* je jedan od načina da se implementira *subtyping*

- *subclassing* treba implementirati jedino ako postoji *subtyping* na konceptualnom nivou

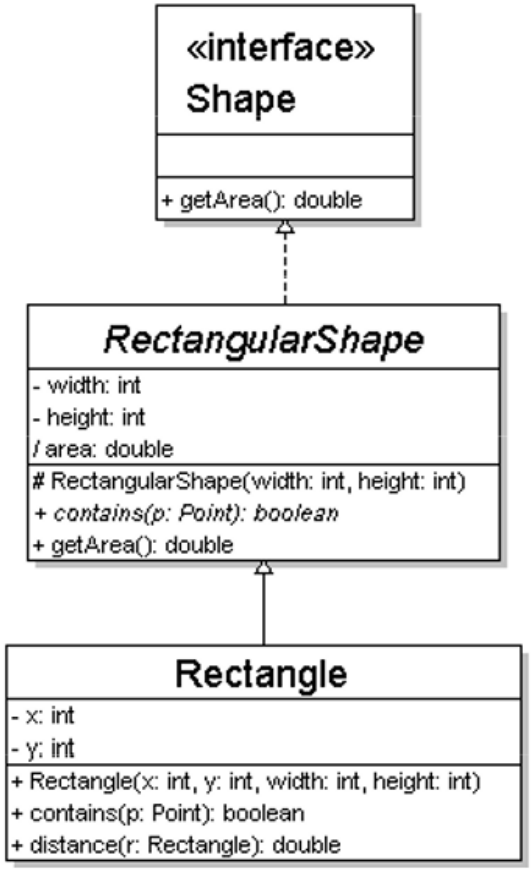
Primjer

- Koja je ovo perspektiva?



Generalizacija

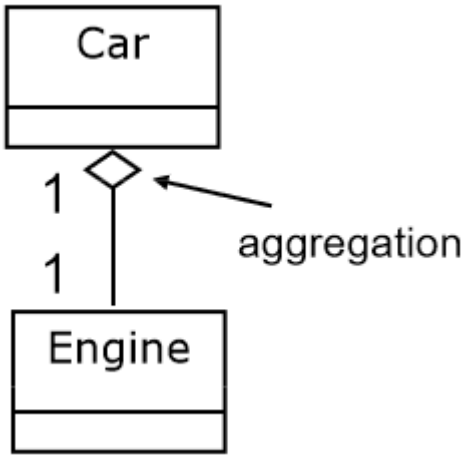
- ...može da predstavlja i implementaciju interfejsa
- Roditelj može biti:
  - klasa
  - apstraktna klasa
  - interfejs



Agregacija

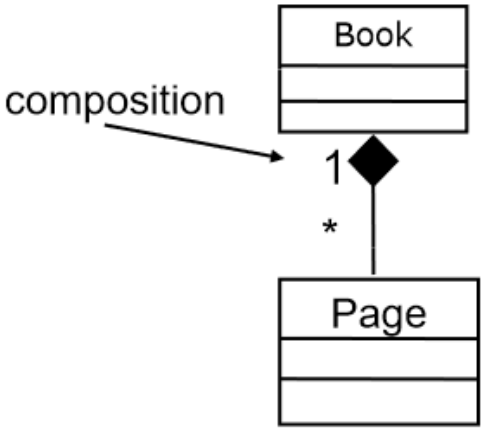
- *„part-of“* veza
  - Primjer: motor je dio automobila

- Razlika između agregacije i asocijacije je mala:
  - agregacijom samo naglašavamo *„part-of“ odnos*

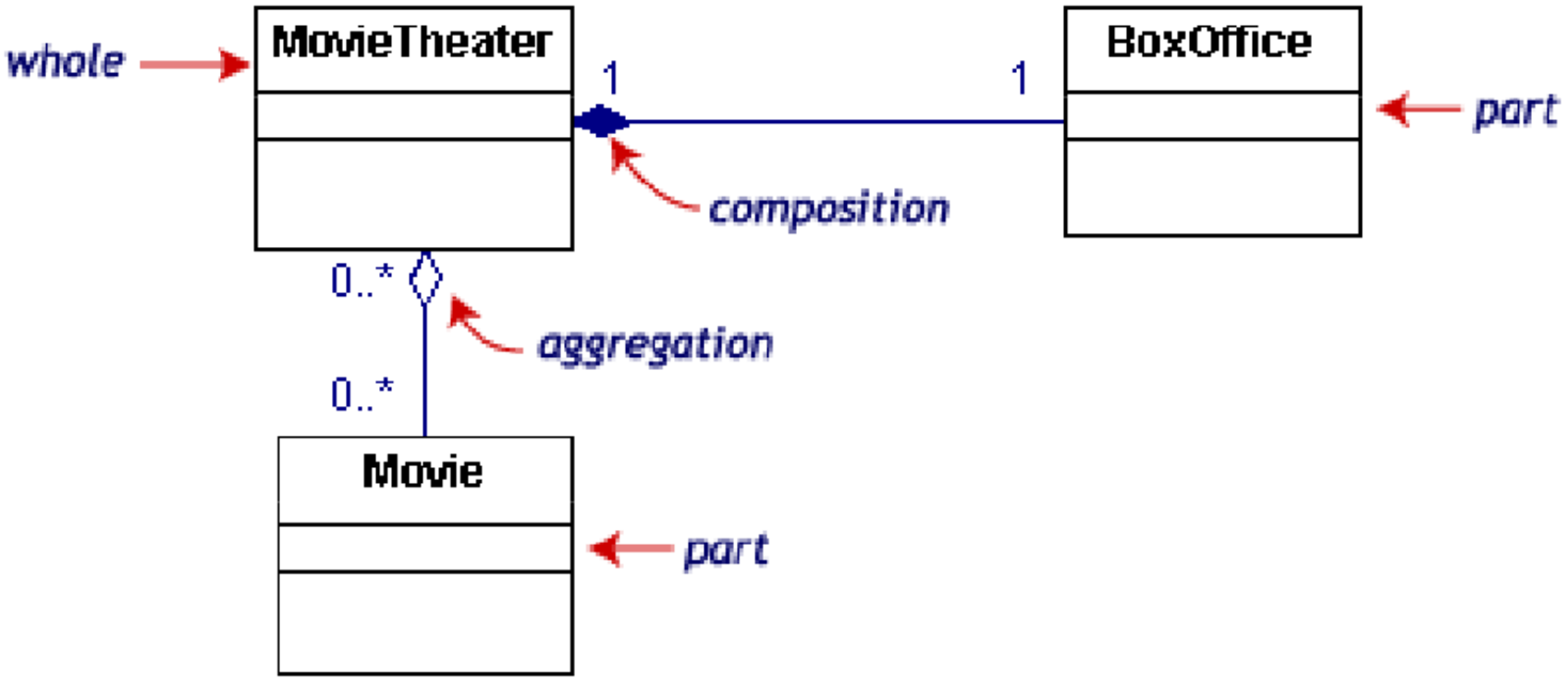


Kompozicija

- Jača od agregacije
  - „is entirely made of“
- Dio može pripadati samo jednoj cjelini
- Životni vijek dijela prati životni vijek cjeline
  - Kada se uništava/briše objekat cjeline -> kaskadno se uništavaju/brišu objekti djelova

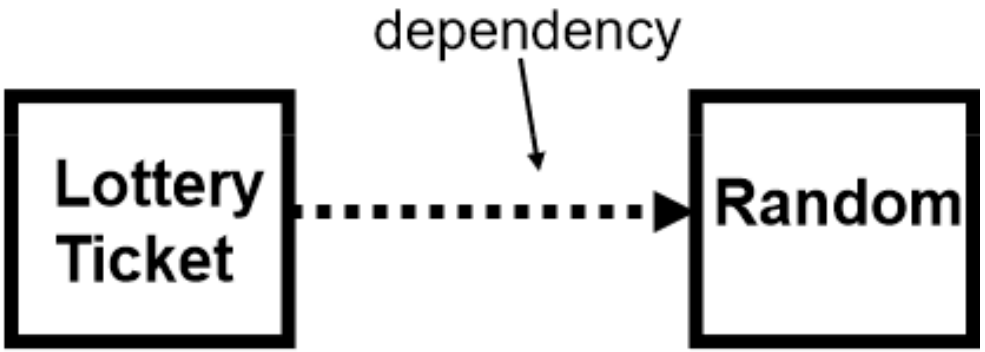


Agregacija vs. Kompozicija

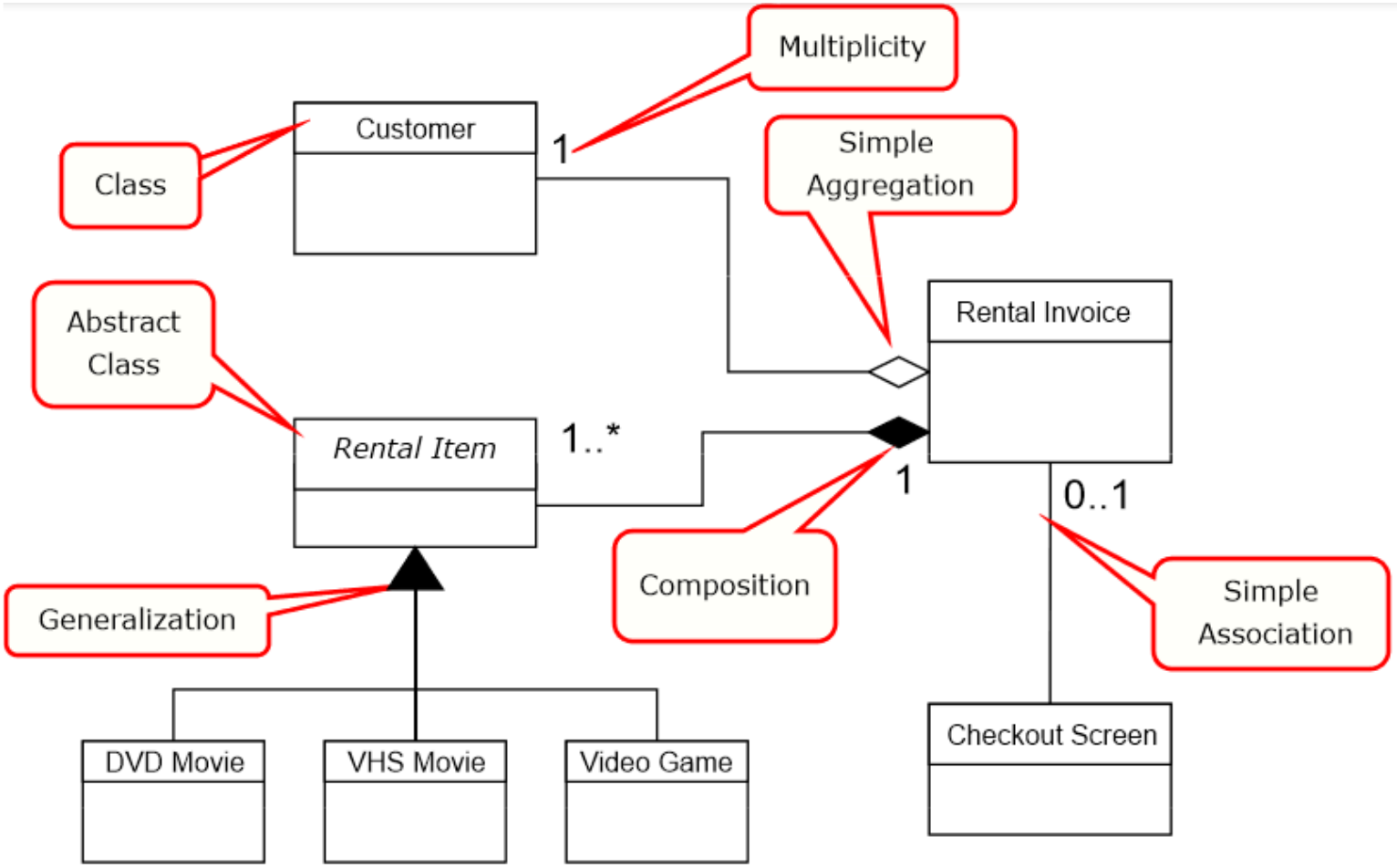


Zavisnost

- Najslabija veza
- „uses temporarily“
- često predstavlja detalje implementacije



Primjer


Multiplicity
Customer
Class
1
Simple
Aggregation
Abstract
Class
Rental Invoice
Rental Item
1..*
1
0..1
Composition
Simple
Association
Generalization
DVD Movie
VHS Movie
Video Game
Checkout Screen

Vježbe

Vježba 3

- Koristeći jedan od UML alata, nacrtati dijagram klasa za program od prošlog časa.

- Uvesti pojam boje za crtanje svake od figura.