Osnove strukturnega modeliranja

Iztok Savnik, Famnit.

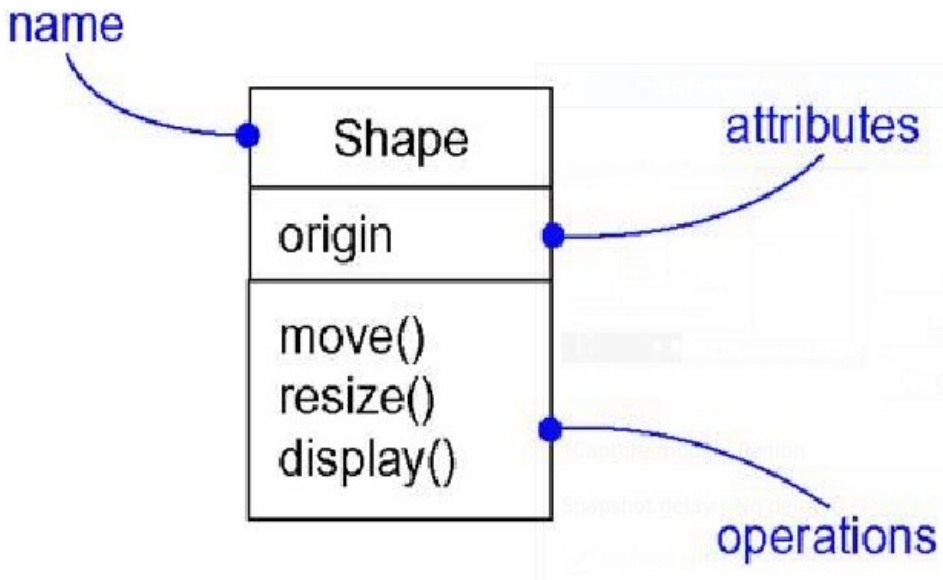
Razredi

- Razredi, atributi, operacije in odgovornosti
- Modeliranje slovarja sistema
- Modeliranje porazdelitve odgovornosti

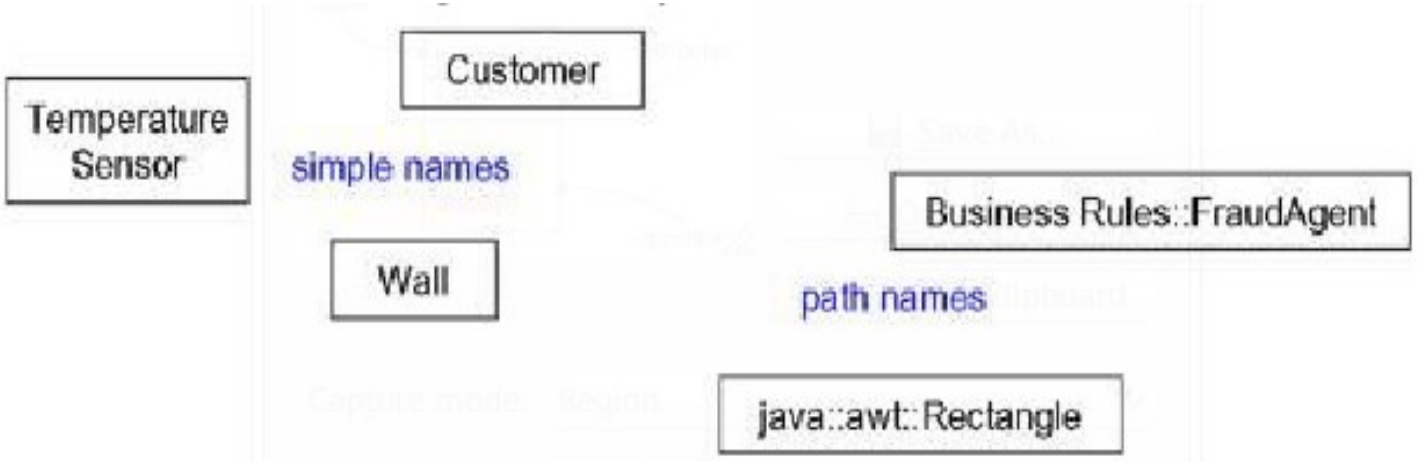
Razredi

- Razredi so najbolj pomemben gradnik objektno-usmerjenega sistema.
- Razred je opis množice objektov, ki si deli atribute, operacije, razmerja in pomen.
- Razred implementira enega ali več vmesnikov.
- Razredi predstavljajo osnovne koncepte sistema, ki ga modeliramo.

Razredi



- Grafično je razred predstavljen s pravokotnikom.
- Ime razreda mora biti unikatno znotraj lokalnega paketa.
  - Enostavno ime; ime poti kjer razred živi

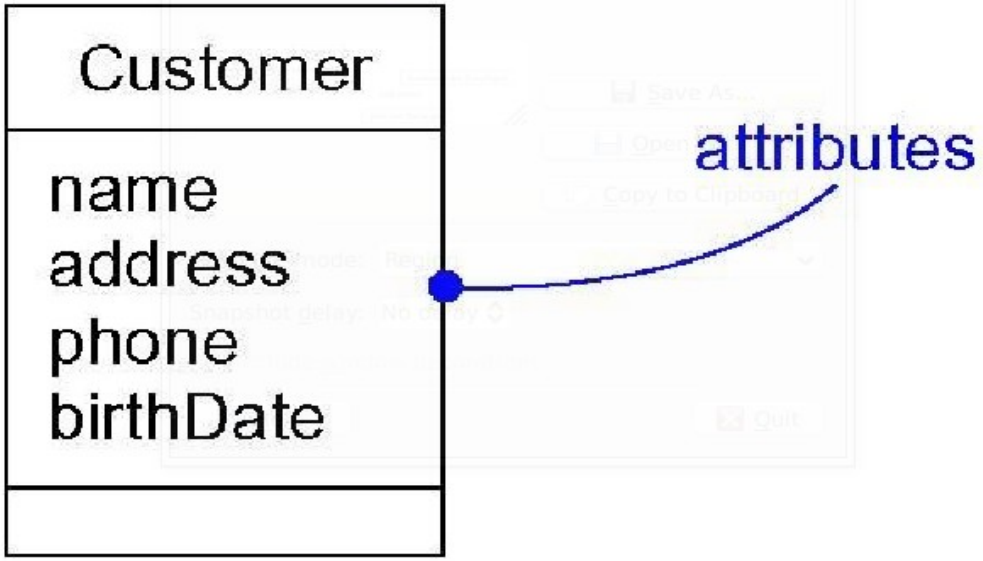


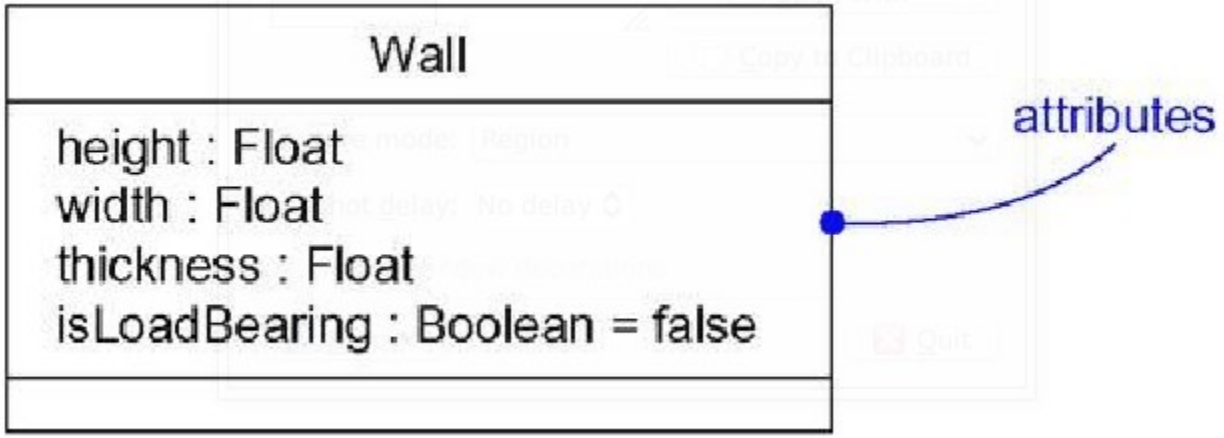
Atributi

- Atribut je imenovana lastnost razreda skupaj z opisom zaloge vrednosti lastnosti.
- Razred ima lahko več atributov ali nobenega.
- Primeri atributov
  - Lastnosti zidov: višina, dolžina, debelina, ...
  - Lastnosti stranke: ime, naslov, telefon, ...
- Atributi so abstrakcija podatkov ali stanja objektov.
- V danem trenutku ima primerek razreda definirane konkretne vrednosti atributov.

Atributi

- Definiramo tipe zaloge vrednosti atributov

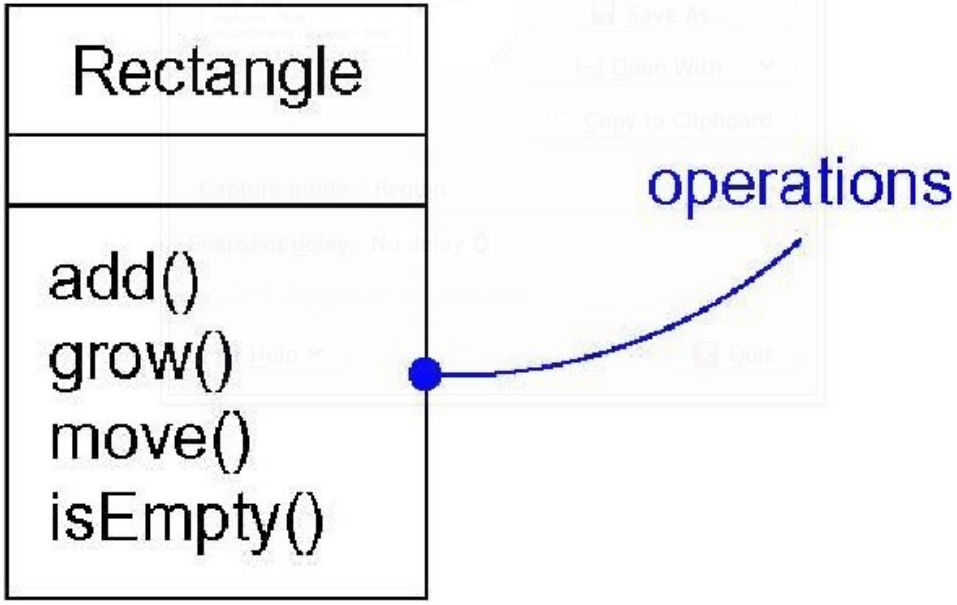




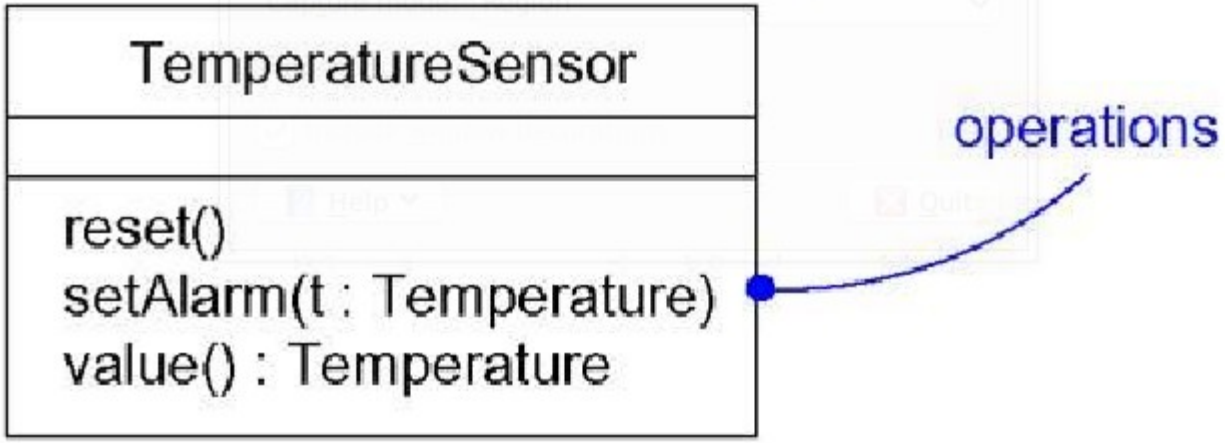
Operacije

- Operacija je implementacija servisa, ki ga nudi objekt .
  - Operacija je abstrakcija akcije, ki jo lahko izvršimo nad objektom oz. primerki danega razreda.
- Razred ima lahko več operacij ali nobene operacije.
- Primer Java paket Awt: Rectangle
  - Primerke pravokotnika lahko premikamo, spreminjamo velikost, sprašujemo glede lastnosti, ...

Operacije

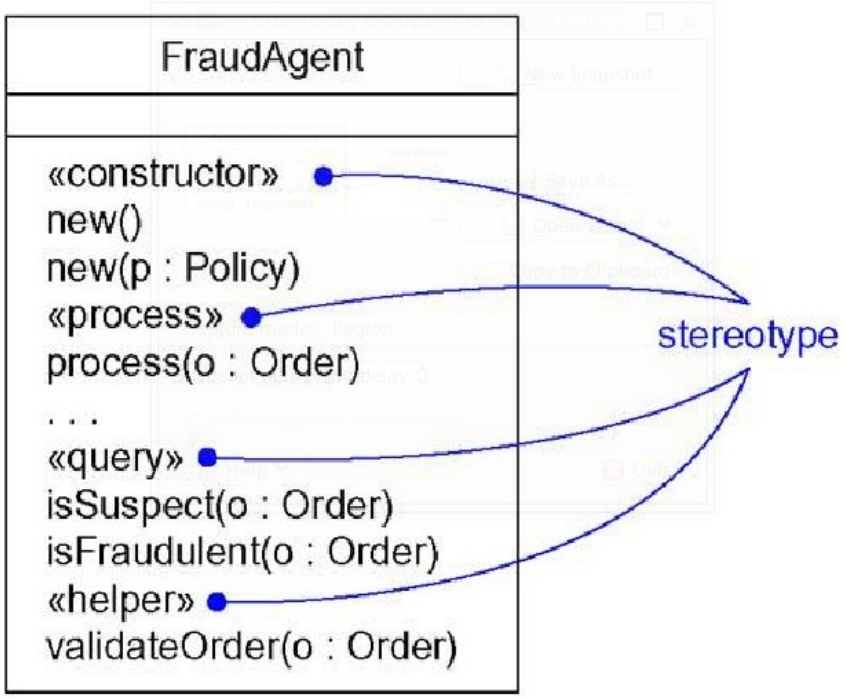


- Grafično so operacije predstavljene pod atributi.
  - Operacije so predstavljene z imenom, imeni in tipi parametrov ter (opcijsko) tipom rezultata operacije.
  - Signatura operacije: $m(p_1:T_1,...,p_k:T_k) \rightarrow T$



Organizacija atributov in operacij

- V večini primerov ne moremo narisati vseh atributov razreda.
  - Običajno jih je preveč.
  - Samo nekateri so uporabni iz danega pogleda.
- Atribute lahko organiziramo v skupine.
  - Vsaka skupina je označena s stereotipom.

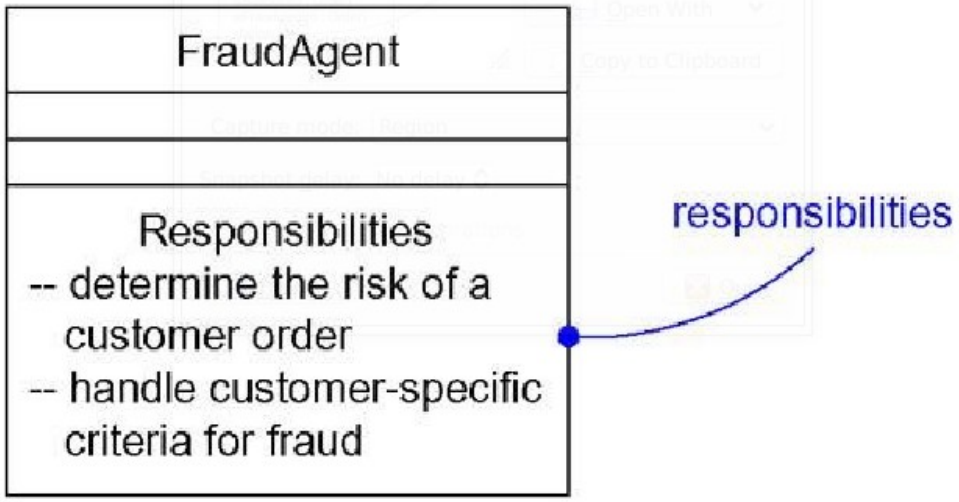


Odgovornosti

- Odgovornost je pogodba ali dolžnost razreda.
  - Ob kreaciji razreda definiramo, da imajo vsi primerki dano statično strukturo in obnašanje.
  - Splošno gledano so atributi in operacije samo del pogodbe glede strukture in obnašanja primerkov.
  - Primeri:
    - Razred Wall mora poznati višino, širino, ... stene.
    - Razred FraudAngent je odgovoren za pregled legitimnosti uporabe kreditne katice.

Odgovornosti

- Začetno lahko specificiramo odgovornosti razredov, ki tvorijo slovar.
  - Vsak razred ima vsaj eno odgovornost toda največ nekaj odgovornosti.
  - Pri modeliranju se odgovornosti prenesejo na atribute, operacije, ...



Preostali gradniki razredov

- Prilastki atributov, vidnost, polimorfizem, ...
- Vmesniki razredov
- Aktivni razredi predstavljajo procese
- Diagrami razredov

Modeliranje slovarja sistema

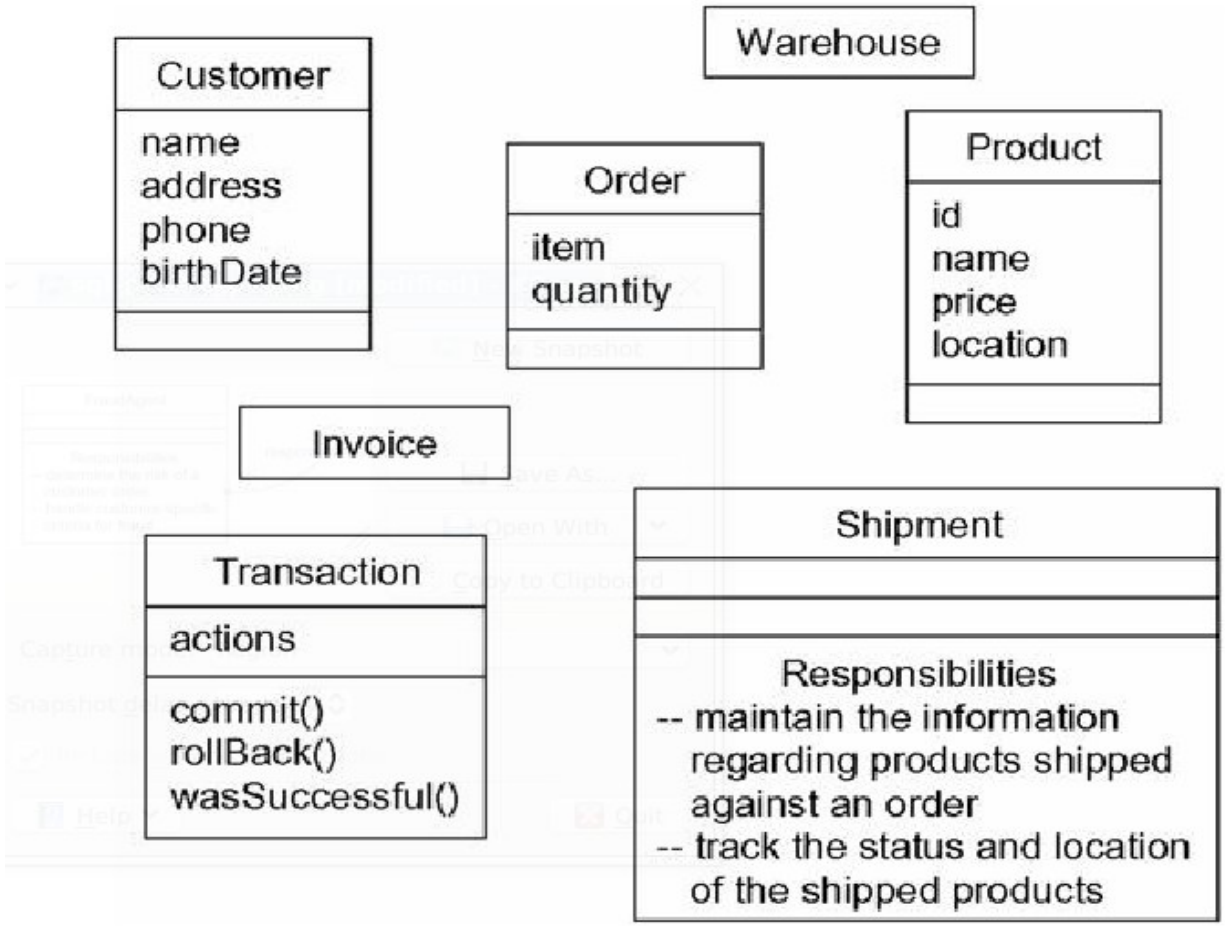
- Razredi so najbolj pogost gradnik za predstavitev abstrakcij sistema.
  - Vsaka abstrakcija je del slovarja sistema.
  - Stvari, ki so pomembne za uporabnike in programerje.
- Večina abstrakcij je dobro poznanih uporabnikom saj jih dnevno uporabljajo pri delu.
  - Primeri uporabe so lep način s katerim lahko pomagamo uporabnikom izraziti osnovne abstrakcije.

Modeliranje slovarja sistema

- Modeliranje slovarja sistema zahteva:
  1. Identifikacija stvari, ki jih uporabniki in programerji uporabljajo pri opisu problemov in rešitev.
  2. Za vsako abstrakcijo identificiraj množico odgovornosti.
     - Odgovornosti razredov morajo biti med sabo usklajene.
  3. Definiraj atribute in operacije, ki so potrebne za modeliranje odgovornosti razreda.
- Pomensko povezane razrede grupiramo v pakete.
- Večina abstrakcij definiranih preko razredov bo sodelovalo v preostalih diagramih.
  - Interakcija med razredi bo definirana s predstavitvijo obnašanja (kasneje).

Primer gradnje slovarja sistema

- Prodaja



Porazdelitev odgovornosti

- Želeli bi zagotoviti, da je odgovornost uravnoteženo porazdeljena po razredih.
- Ne želimo, da je razred prevelik ali premajhen.
- Definicija prevelikih abstrakcij vodi do rigidnega modela, ki ga je težko spremeniti in ponovno uporabiti.
- Definicija premajhnih razredov vodi do nepreglednega modela z velikim številom abstakcij.

Porazdelitev odgovornosti

- Modeliranje porazdelitve odgovornosti po razredih.
  1. Identificiraj množico razredov, ki deluje skupno za realizacijo nekega obnašanja.
  2. Identificiraj množico odgovornosti za posamezne razrede.
  3. Razcepi razrede, ki imajo preveč odgovornosti in združi tiste, ki so premajhni v enega.
  4. Upoštevaj komunikacijo med razredi in porazdeli odgovornosti v skladu s komunikacijo, tako da sodelujoči razred ne naredi preveč ali premalo.

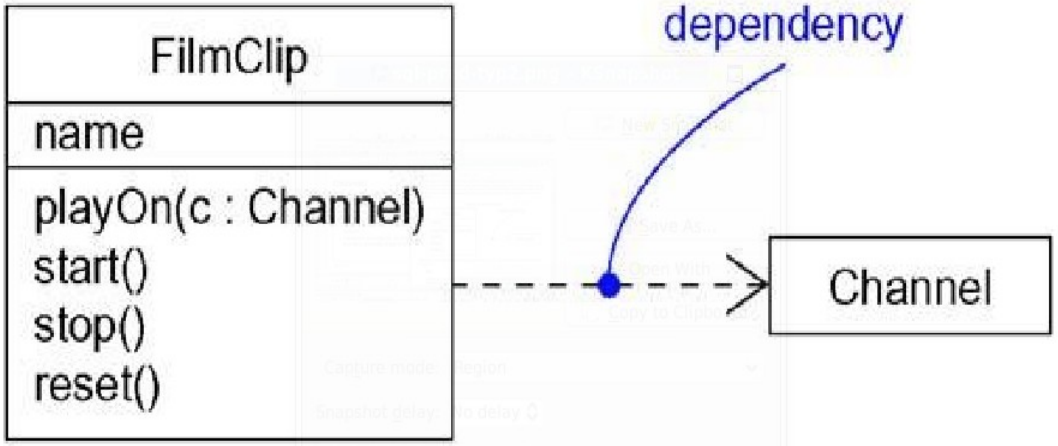
Razmerja

- Odvisnost, generalizacija in asociacija
- Modeliranje enostavnih odvisnosti
- Modeliranje dedovanja
- Modeliranje strukturnih razmerij
- Kreacija mreže razmerij

Razmerja

- Ob kreiranju abstrakcij z uporabo razredov vidimo, da skoraj noben razred ni definiran sam zase.
  - Večina razredov sodeluje z drugimi razredi na veliko načinov.
  - Pri modeliranju ne gledamo samo na stvari ampak modeliramo tudi razmerja med stvarmi.
  - Primer hiše: stene se držijo sten, vrata in okna so v stenah, luči in oprema so v sobi, itd.
- Tri vrste razmerij so posebaj pomembne.
  - Odvisnost
  - Generalizacija
  - Asociacija

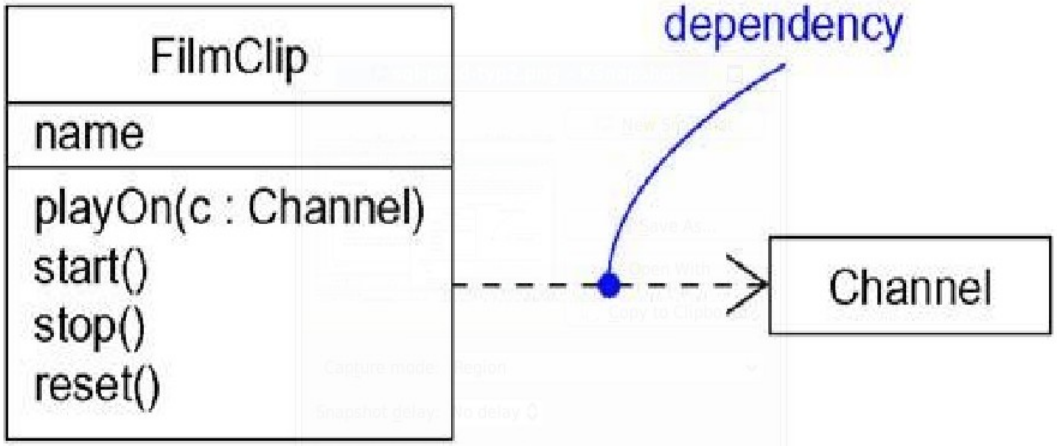
Odvisnost



- Odvisnost

  izraža razmerje *uporablja.*
  - Sprememba specifikacije ene stvari vpliva na drugo stvar, ki *uporablja* prvo stvar.
  - Odvisnost ni nujno tudi v drugo stran.
  - Grafično je odvisnost predstavljena s črtkasto puščico.
  - Odvisnost uporabljamo, ko želimo izraziti, da ena stvar uporablja drugo stvar.

Odvisnost

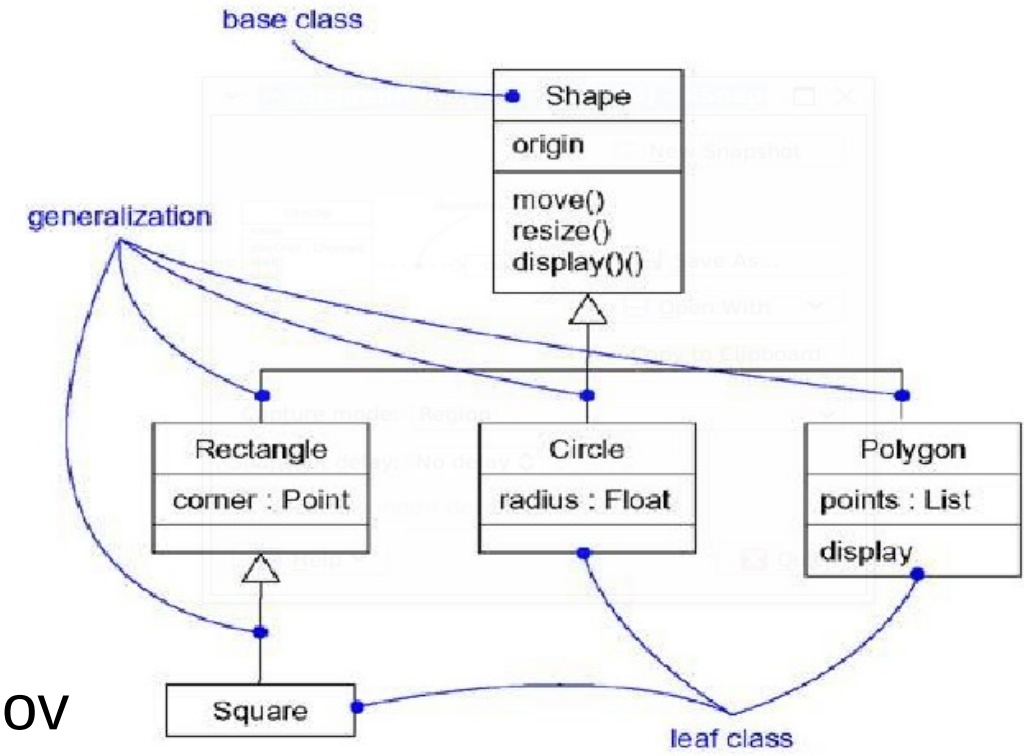


- Uporaba odvisnosti.
  - Največkrat predstavlja odvisnost med razredi preko parametrov metod.
  - To je razmerje *uporablja*, ki pove, da sprememba uporabljenega razreda vpliva na dan razred--- uporabljen razred ima lahko spremenjen vmesnik ali obnašanje.
- Odvisnosti so lahko tudi med drugimi stvarmi.
  - Med paketi, med beležkami in paketi, itd.

Generalizacija

- Generalizacija je razmerje med splošno stvarjo in bolj specifično stvarjo.
  - Splošno stvar imenujemo nad-razred ali starš.
  - Specifične stvari imenujemo pod-razredi ali otroci.
  - Primerki otrokov se lahko uporabljajo povsod tam kjer se lahko uporabljajo starši ne pa tudi obratno.
  - Otroci podedujejo lastnosti staršev.
  - Otrok ima lahko svoje atribute in metode, ki lahko tudi prekrijejo podedovane.
  - To imenujemo tudi polimorfizem.
  - Grafično je generalizacija predstavljena s puščico, ki ima prazno glavo.

Generalizacija



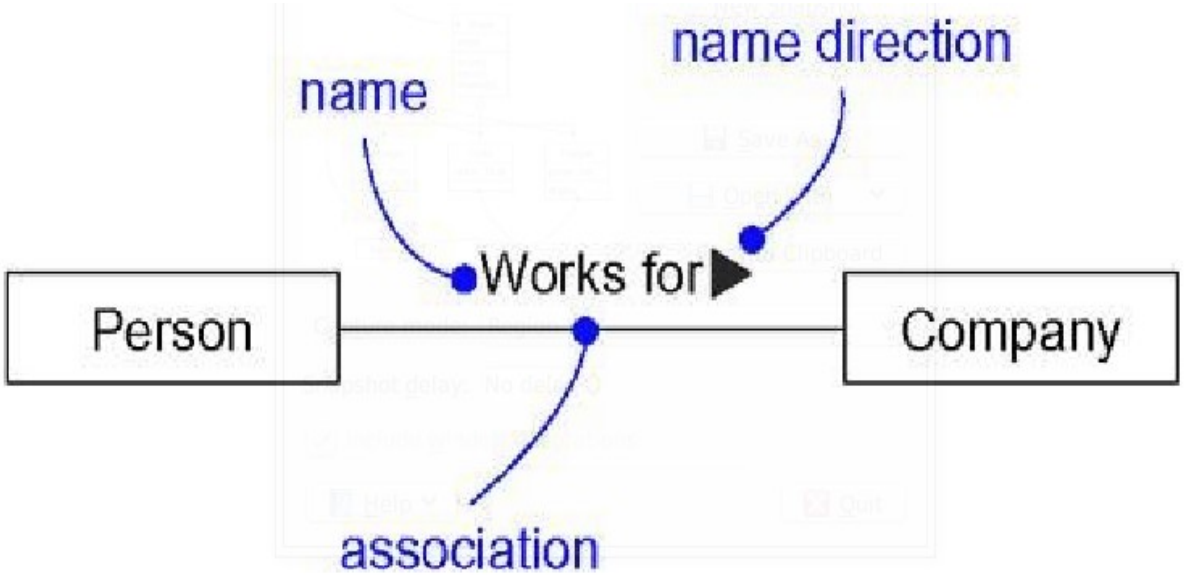
- Razred ima lahko
  0, 1 ali več
  nad-razredov.
- Razred brez nad-razredov
  in z večimi podrazredi imenujemo korenski razred.
- Razred, ki nim<a podrazredov imenujemo list.
- Velikokrat uporabljamo generalizacija za modeliranje dedovanja.
- Generalizacojo lahko uporabljamo tudi za opis razmerij med paketi.

Asociacija

- Asociativnost in odvisnost sta lahko refleksivni, generalizacija pa ni.
- Asociacija je strukturno razmerje, ki pove da so primerki neke stvari povezani s primerki druge stvari.
- Asociacija, ki povezuje dva razreda služi za navigacijo iz enega objekta na drugi.
  - Imenujemo jo tudi binarno razmerje.
- Imamo tudi asociacije, ki povezujejo več kot dva razreda.
- Grafično je asociacija predstavljena s polno črto.

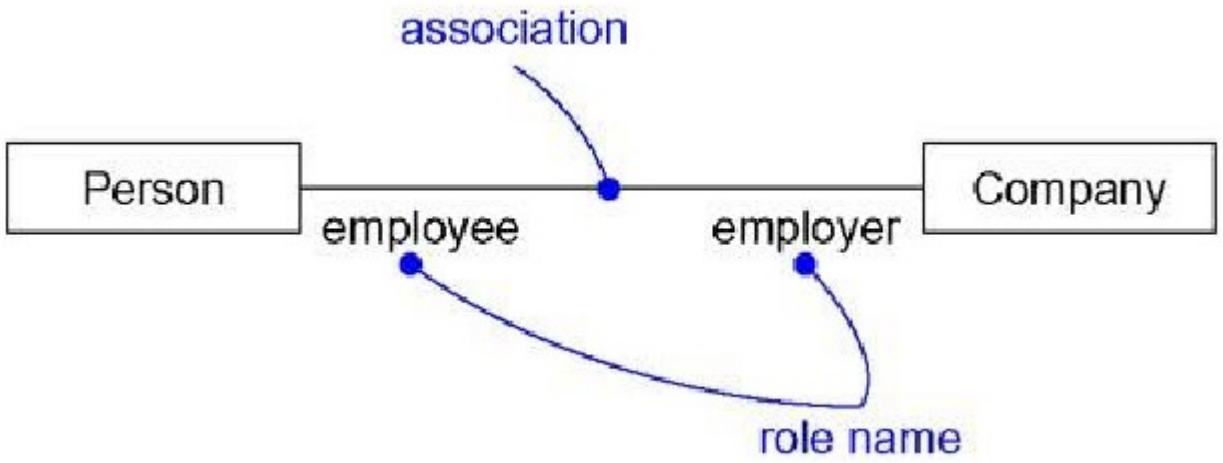
Asociacija

- Asociacija ima ime, ki je lahko označeno s smerjo asociacije.
- Velikokrat uporabljamo vloge in asociaciji ne damo ime.



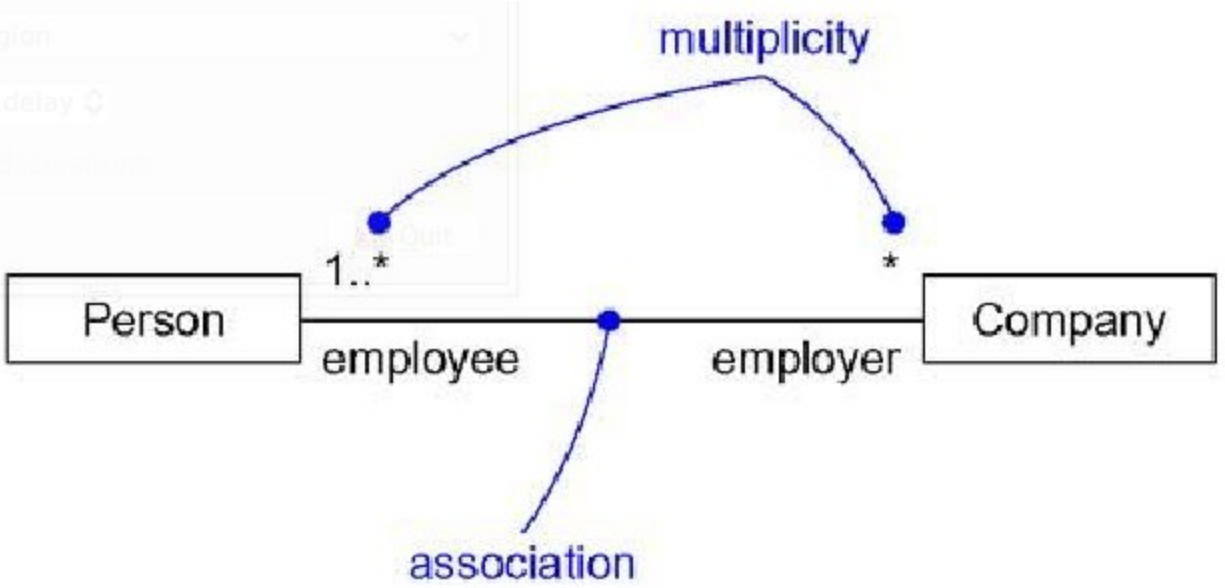
Vloga asociacije

- Razred, ki sodeluje v asociaciji ima določeno vlogo.
  - Vloga ima lahko ime, ki pove kakšno vlogo ima razred v asociaciji.
  - Oseba ima v asociaciji z razredom Firma vlogo *zaposlen* in firma ima vlogo *zaposlovalca*.



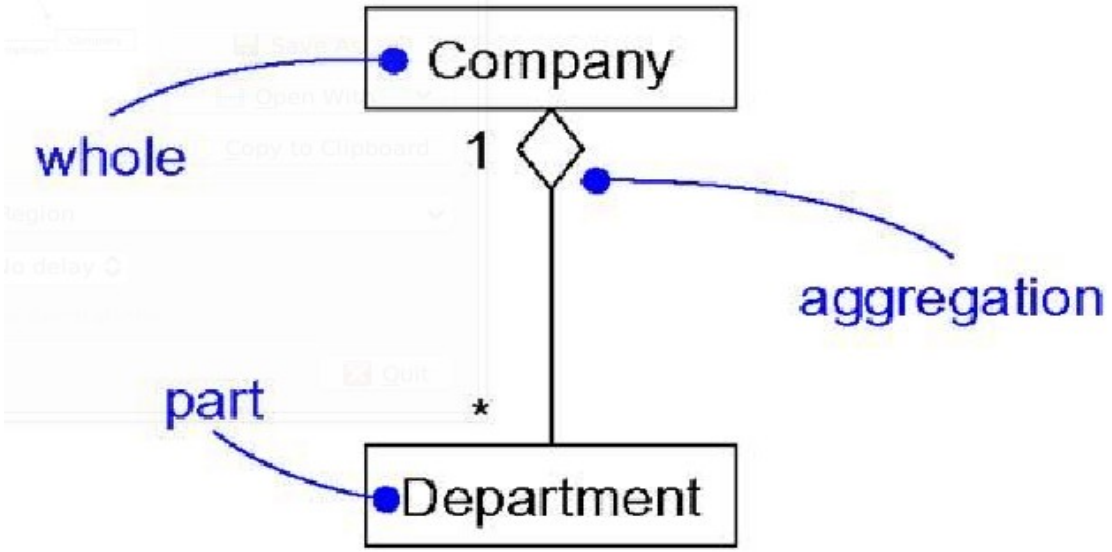
Številnost asociacije

- Asociacija predstavlja strukturno razmerje med objekti.
  - Opišemo lahko koliko objektov je v strukturni ralaciji preko asociacije.
  - Uporabljamo številnost kot v primeru modela ER.



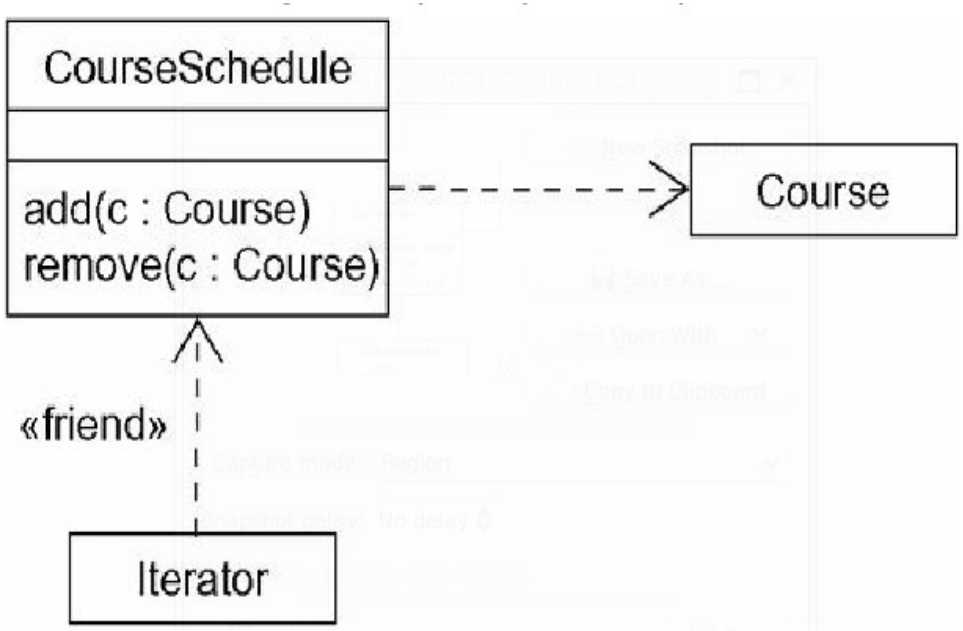
Agregacija

- Navadna asociacija predstavlja strukturno relacijo med enakovrednimi igralci.
  - Oba dela sta povezana na istem nivoju in nobena vloga ni bolj pomembna.
- Agregacija uporablja razmerje celota / del kjer en razred predstavlja večjo *celoto* in drug razred predstavlja manjši *del*.



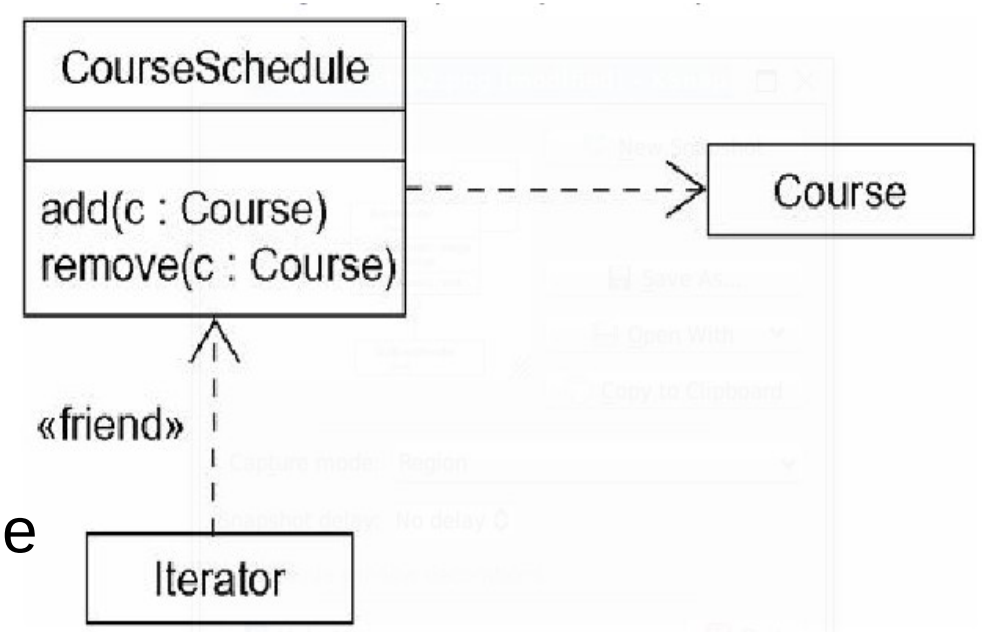
Modeliranje enostavnih odvisnosti

- Najbolj pogosto razmerje odvisnosti med razredi je v primeru, da metoda uporablja razred kot parameter.
  - Kreiraj odvisnost med razredoma A in B v primeru, da A vsebuje metodo s parametrom tipa B.
- Primer prikaže model sistma, ki omogoča prirejanje študentov in učiteljev predmetom.



Modeliranje enostavnih odvisnosti

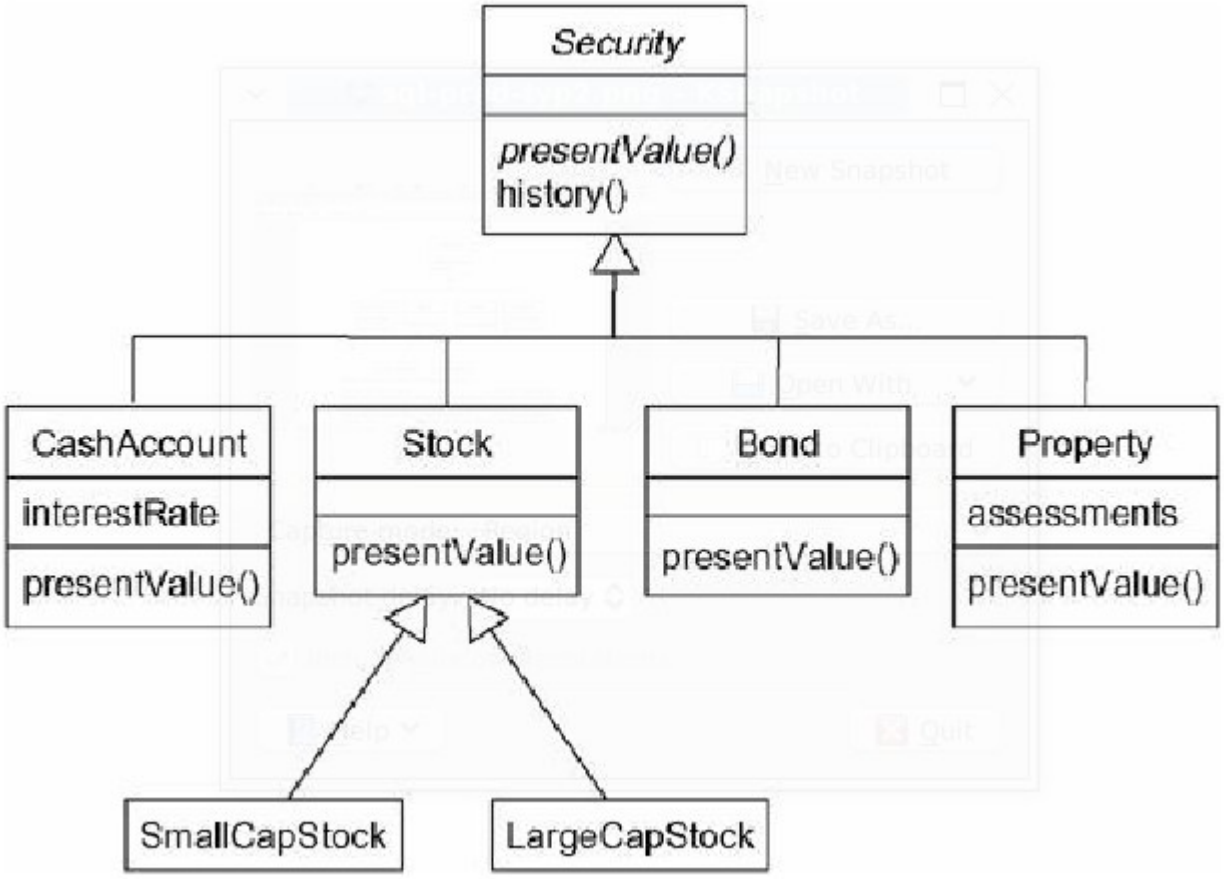
- Odvisnost med CourseSchedule in Course.
- Course je parameter add() in remove().
- Če podamo polno signaturo operacij potem ni potrebno risati odvisnosti.
- Iterator uporablja CourseSchedule, medtem ko ta razred ne ve nič o iteratorju.
- Uporabljen je stereotip za predstavitev C++ relacije friend.



Modeliranje dedovanja

- Med načrtovanjem velikokrat naletimo na situacijo kjer je potrebno modelirati več podobnih razredov.
  - Koncepte lahko modeliramo kot ločene abstrakcije.
  - Lahko uporabljamo dedovanje, da izkoristimo skupne lastnosti razredov.
- Modeliranje dedovanja:
  - Za dano množico razredov poišči skupne odgovornosti, atribute in metode.
  - Predstavi skupne lastnosti kot poseben razred. Če je potrebno kreiraj nov razred, ki vsebuje skupne lastnosti.
  - Definiraj bolj specifične razrede, ki dedujejo od skupnega nadrazreda, z uporabo razmerja generalizacija.

Modeliranje dedovanja



Modeliranje dedovanja

- Imena *Security* in *presentValue* določajo, da je predstavljen razred abstrakten.
  - Pri gradnji hierarhije dedovanja večkrat naletimo na primer, ko je neko notranje vozlišče hierarhije abstraktno.
- Lahko kreiramo razrede, ki imajo več kot en nadrazred.
  - Dobimo večkratno dedovanje, kar ne omogočajo vsi programski jeziki.

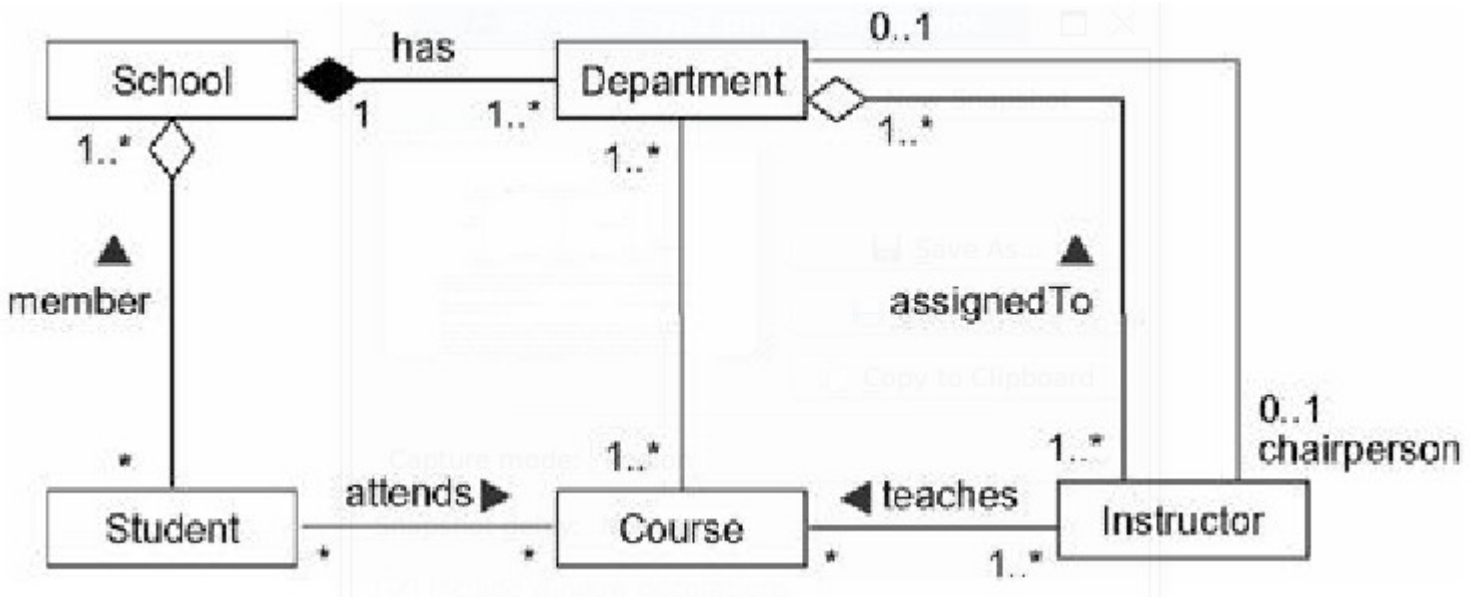
Modeliranje strukturnih razmerij

- Razmerji odvisnost in generalizacija sta enosmerni.
  - Eden izmed razredov ne ve nič o razmerju.
- Asociacija definira zvezo med razredi v obe smeri.
  - Oba razreda sta v zvezi.
  - Asociacijo lahko uporabljamo za navigacijo v obe smeri.
  - Definirana je strukturna relacija, ki definira pot med objekti razredov v zvezi.

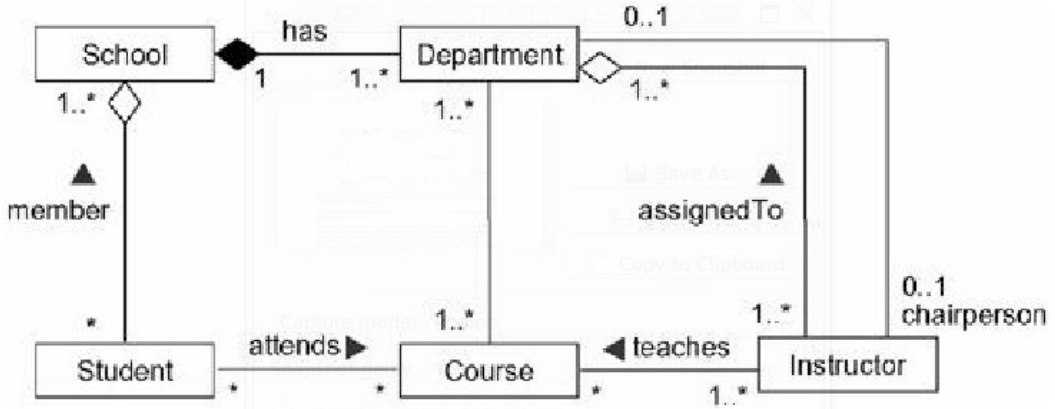
Modeliranje strukturnih razmerij

1. Za vsak par razredov, kjer je potrebna navigacija iz enega razreda do drugega in obratno, definiraj asociacijo med razredoma.
   - To je podatkovni pogled na asociacijo.
2. Za vsak par razredov, če primerki razreda potrebujejo interakcijo s primerki drugega razreda, potem definiraj asociacijo med njima (ne samo v primeru parametrov).
   - To je pogled na asociacijo preko obnašanja.
3. Za vsak razred v asociaciji definiraj številnost kot tudi imena vlog.
4. Če asociacija definira razmerje celota/komponenta, potem uporabi bodisi poln ali votel diamant na strani celote.

Modeliranje strukturnih razmerij



Modeliranje strukturnih razmerij



- Poglejmo si asociacije primera bolj podrobno.
  - Asociacija med Student in Course definira, da študenti obiskujejo predmete.
  - Podobno, asociacija med Course in Instructor pravi, da učitelji učijo predmete.
  - Asociacija med School in Department definira agregacijo. Šola je sestavljena iz oddelkov.
  - Prav tako asociacija med Student in School definira agregacijo. Šolo sestavljajo študenti.
  - Imamo dve asociaciji med Department in Instructor. Učitelj je lahko v enem ali večih oddelkih in vodja oddelka je lahko učitelj, ni pa nujno da obstaja iz obeh strani.
  - Asociacija med Course in Department pravi, da imajo oddelki enega ali več predmetov ter predmeti spadajo v en ali več oddelek.

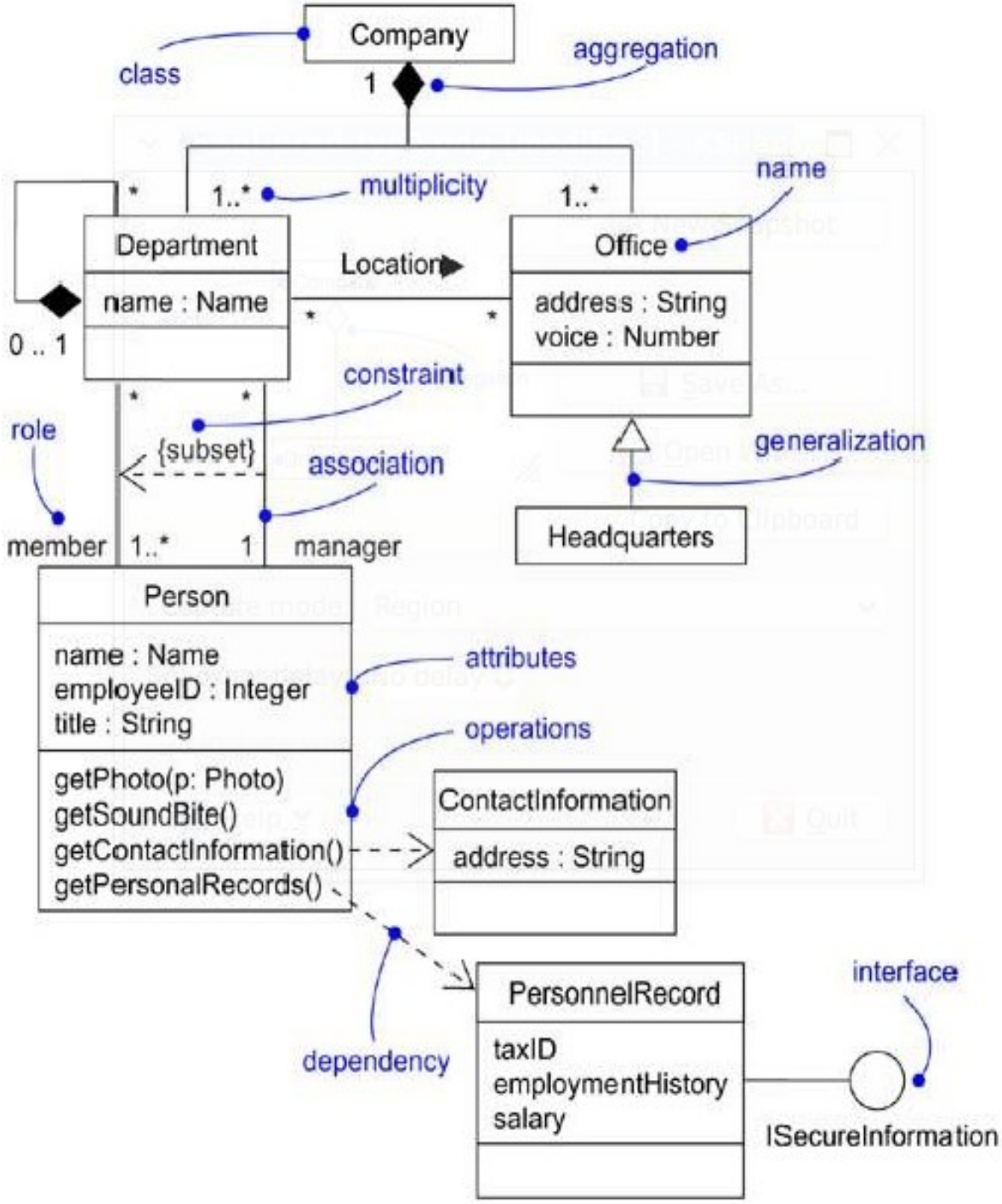
Diagrami razredov

- Uporaba diagramov razredov
- Modeliranje enostavnih sodelovanj
- Modeliranje sheme podatkovne baze
- Konstrukcijsko in vzvratno inženirstvo

Diagrami razredov

- Najbolj pogosti diagrami za modeliranje objektno-usmerjenega sistema.
  - Modeliranje slovarja sistema
  - Modeliranje enostavnega sodelovanja
  - Modeliranje logične sheme podatkovne baze
- Modeliranje statičnega načrtovalskega pogleda na sistem z diagrami razredov.
  - Osnova za diagrame komponent in diagrame postavitve.

Zgled


Company
class
aggregation
1
multiplicity
1..*
1..*
name
Department
Location
Office
name : Name
address : String
voice : Number
0 .. 1
constraint
role
{subset}
association
generalization
member
1..*
1
manager
Headquarters
Person
name : Name
employeeID : Integer
title : String
attributes
operations
getPhoto(p: Photo)
getSoundBite()
getContactInformation()
getPersonalRecords()
ContactInformation
address : String
PersonnelRecord
interface
taxID
employmentHistory
salary
dependency
ISecureInformation

Enostavna sodelovanja

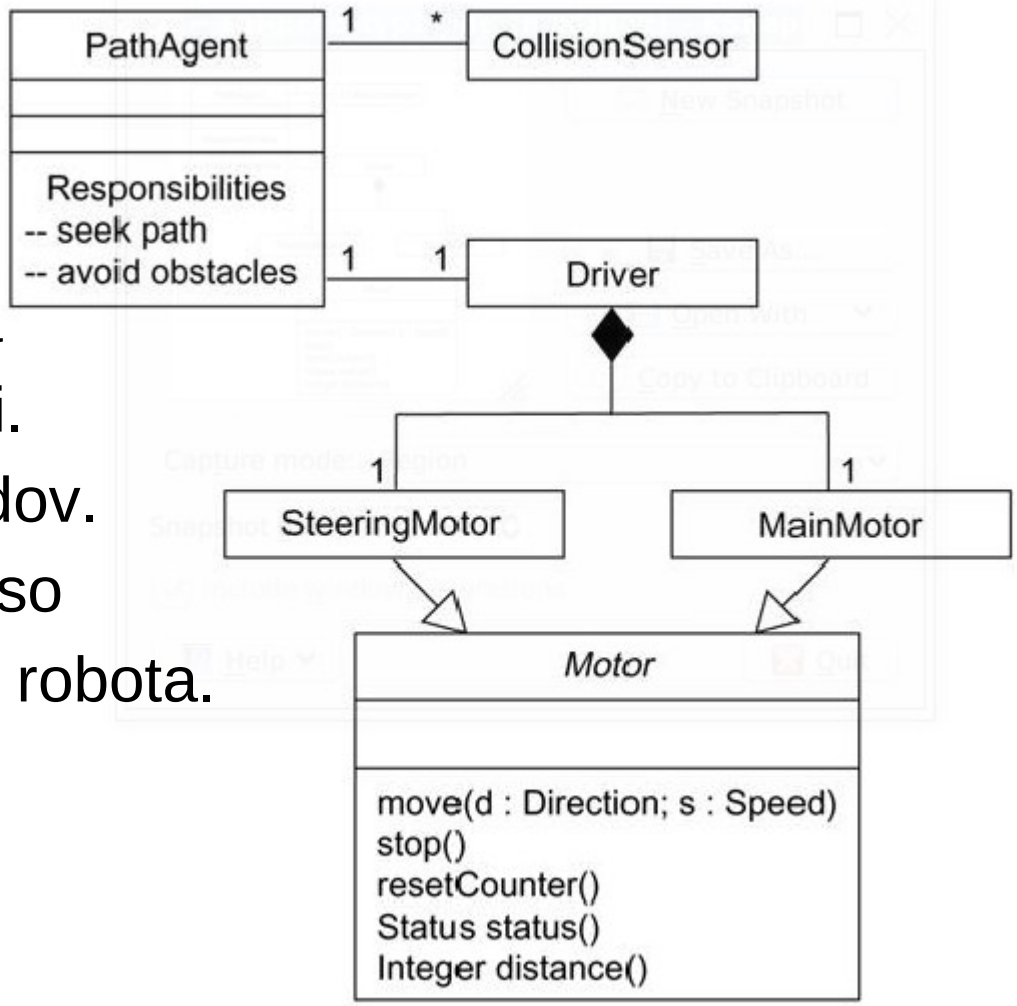
- Nobeden razred ni samostojen—vsak deluje v sodelovanju z drugimi za izvedbo semantike, ki je večja od vsote posameznih razredov.
  - Vizualizacija, specifikacija, konstrukcija in dokumentacija načinov sodelovanja elementov slovarja.
  - Diagrame razredov uporabimo za predstavitev takšnih sodelovanj.
- Diagram razredov prikaže samo del stvari in razmerij, ki sestavljajo načrtovalski pogled sistema.
  - Vsak diagram se mora osredotočiti na eno samo sodelovanje.

Enostavna sodelovanja

- Modeliranje sodelovanja:
  1. Identificiraj mehanizme, ki bi jih želeli modelirati. Mehanizem predstavlja neko funkcijo ali obnašanje dela sistema, ki je rezultat interakcije združbe razredov, vmesnikov in drugih stvari.
  2. Za vsak mehanizem določi razrede, vmesnike in druga sodelovanja, ki so del tega sodelovanja. Identificiraj tudi zveze med temi stvarmi.
  3. Uporabi scenarije za opis združbe razredov. Tukaj odkrijemo lahko dele, ki manjkajo ali so narobe modelirani.
  4. Vnesi podatke o modeliranih stvareh. Poišči odgovornosti za vsak posamezen razred (ravnotežje). Sčasoma se odgovornosti pretvorijo v atribute in metode.

Enostavna sodelovanja

- Implementacija avtonomnega robota.
- Shema fokusira na razrede, ki so potrebni za premikanje robota po poti.
- Obstaja precej več razredov. Prikažemo samo tiste, ki so pomembni za premikanje robota.



Enostavna sodelovanja

- En razred lahko sodeluje v večih shemah.
- En diagram fokusira na eno sodelovanje.
- S fokusiranjem na različna sodelovanja preko posebnih diagramov kreiramo razumljiv pogled iz večih zornih kotov.

Shema podatkovne baze

- Sistemi, ki jih načrtujemo imajo velikokrat persistentne objekte.
  - Običajno uporabljamo relacijski SUPB, objektni ali relacijsko-objektni sistem SUPB.
- UML diagrami razredov vsebujejo gradnike podatkovnega modela ER.
  - Diagrami razredov imajo poleg gradnikov za modeliranje podatkov tudi gradnike za opis obnašanja.
  - SUPB predstavi obnašanje kot prožilce ali shranjene procedure.

Shema podatkovne baze

1.Identificiraj razrede katerih primerki katerih stanje mora preživeti izvajanje aplikacije.

2.Kreiraj diagrame razredov, ki vsebujejo persistentne razrede. Označi razred kot persistenten z označenimi vrednostmi.

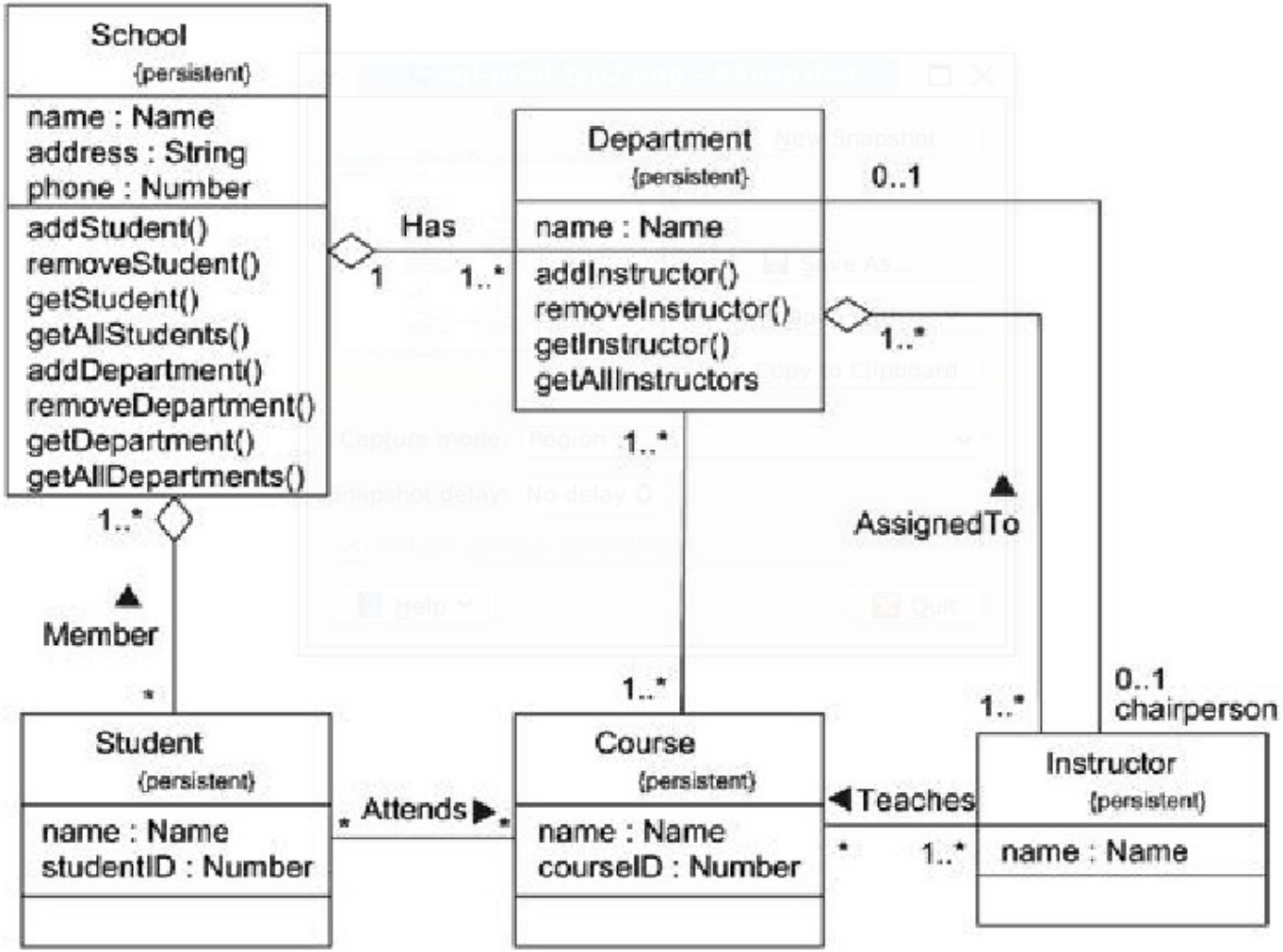
3.Razširi strukturne podrobnosti: določi podrobnosti o atributih, asociacijah in številnost vlog asociacij.

4.Poišči vzorce, ki komplicirajo načrt podatkovne baze: ciklične asociacije, relacije tipa ena-ena in n-terne relacije. Dodaj poenostavitve, ki razrešijo probleme.

5.Obravnavaj obnašanje razreda, ki je vezano na dostop do podatkov in integriteto podatkov. Poslovna pravila so običajno kreirana en nivo višje.

6.Kjer je mogoče uporabi orodja za prevod sheme v fizični načrt.

Shema podatkovne baze



Shema podatkovne baze

- Primer prikaže informacijski sistem šole.
  - Shema predstavlja razvit logičen načrt pripravljen na prevod v fizično shemo.
  - Vseh šest razredov je označeno kot persistentni.
  - Vsi atributi imajo primitiven tip—zveze med razredi so definirane z relacijami.
  - Dva od razredov (šola in oddelek) vsebujeta operacije, ki delajo z integritetnimi omejitvami.
  - Operacije, ki preverijo pogoje za prijavo predmetov sodijo na višji nivo.

Konstrukcijsko in vzvratno inženirstvo

- Osnovni rezultat razvojnega teama je programska oprema in ne diagrami.
- Razlog za kreacijo modelov je predvidljiva izdelava prave programske opreme v pravem času.
- Kreirani modeli in implementacija morajo biti usklajeni.
  - Cena pretvarjanja med modelom in implementacijo naj bo minimalna.

Konstrukcijsko in vzvratno inženirstvo

- Nekateri modeli izdelani v UML ne bodo nikoli pretvorjeni v kodo.
  - V primeru, da bomo modelirali poslovno dejavnost z diagramam aktivnosti potem bo veliko aktivnosti vključevalo ljudi in ne računalnike.
  - V drugih primerih bomo modelirali sisteme katerih deli so strojna oprema, ki vsebuje programsko opremo.
  - V večini primerov bomo seveda pretvorili načrte v kodo.
  - UML ne vsebuje konkretno preslikavo v kodo vendar je bil zasnovan tako, da je preslikava v objektno usmerjen jezik možna.
  - Diagrami razredov so posebaj zasnovani tako, da omogočajo prevod v oo jezike kot so npr. C++, Java, itd.

Konstrukcijsko inženirstvo

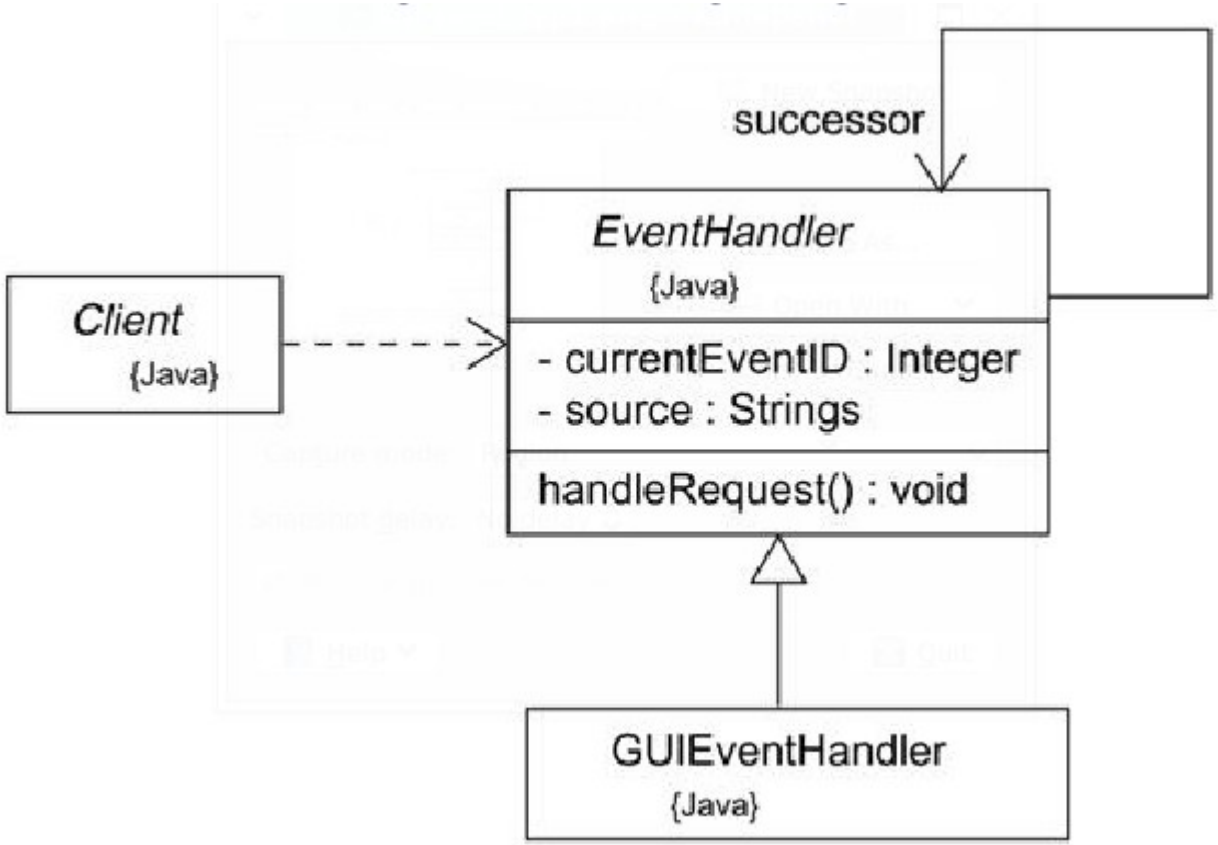
- Konstrukcijsko inženirstvo je proces pretvorbe modelov v implementacijski jezik.
  - Konstrukcijsko inženirstvo vedno povzroči izgubo informacij ker so UML modeli semantično bogatejši kot programski jeziki.
  - To je tudi eden od glavnih razlogov zakaj bomo potrebovali tudi modele in ne samo kodo.
  - Strukturne lastnosti kot so sodelovanja in lastnosti obnašanja kot npr. interakcija lahko vizualiziramo v UML vendar ne tako jasno iz kode.

Konstrukcijsko inženirstvo

- Konstrukcijsko inženirstvo diagrama razreda:
  1. Identificiraj pravila za prevajanje razredov v implementacijski jezik na nivoju celotnega projekta ali organizacije.
  2. V odvisnosti od izrazne moči programskega jezika je potrebno omejiti uporabo nekateri UML gradnikov npr. uporabo večkratnega dedovanja, če imamo na razpolago samo enostavno dedovanje od enega razreda.
     - Bodisi prepovemo uporabo večkratnega dedovanja ali definiramo (kompleksna) pravila za pretvarjanje večkratnega dedovanja v enostavno dedovanje.
  3. Uporabi označene vrednosti za specifikacijo ciljnega jezika.
     - To lahko naredimo na nivoju kolaboracij ali na nivoju posameznega razreda.
  4. Uporabljaj orodja za konstrukcijsko inženirstvo modelov.

Konstrukcijsko inženirstvo

- Model klasičnega upravljalnika dogodkov.
- Dva abstraktna razreda.
- Konkreten razred GUIEventHandler.



Konstrukcijsko inženirstvo

- Vsi razredi se prevedejo v Javo.
- Konstrukcijsko inženirstvo razreda EventHandler vodi do naslednje kode:

```
public abstract class EventHandler {
    EventHandler successor;
    private Integer currentEventID;
    private String source;

    EventHandler() {}
    public void handleRequest() {}
}
```

Vzvratno inženirstvo

- Vzvratno inženirstvo je pretvorba kode v model.
  - Rezultat vzvratnega inženirstva je tok podatkov od katerih so nekateri preveč podrobni za gradnjo modela.
- Rezultat vzvratnega inženirstva ni popolen.
  - Omenili smo izgubo informacij pri konstrukcijskem inženirstvu.
  - Model ne more biti kompletno restavriran iz kode razen v primeru, da imamo pomen opisan kot nek komentar kode.

Vzvratno inženirstvo

- Identificiraj pravila za preslikavo implementacijskega jezika v model.
  - To običajno naredimo na nivoju organizacije ali projekta.
- Identificiraj kodo, ki jo želimo pretvoriti v model in uporabi orodje za vzvratno inženirstvo.
  - Lahko uporabljamo prejšnji model, ki smo ga pretvorili v kodo s konstrukcijskim inženirstvom.
- Interaktivno popravljaj model z uporabo poizvedb nad modelom.
  - Po potrebi dodajaj in odvzemaj razrede.
  - Razširi diagram s sledenjem razmerju, ki vodi do drugega razreda.

Literatura

- Grady Booch, James Rumbaugh, Ivar Jacobson, The Unified Modeling Language User Guide, 2000, Addison Wesley