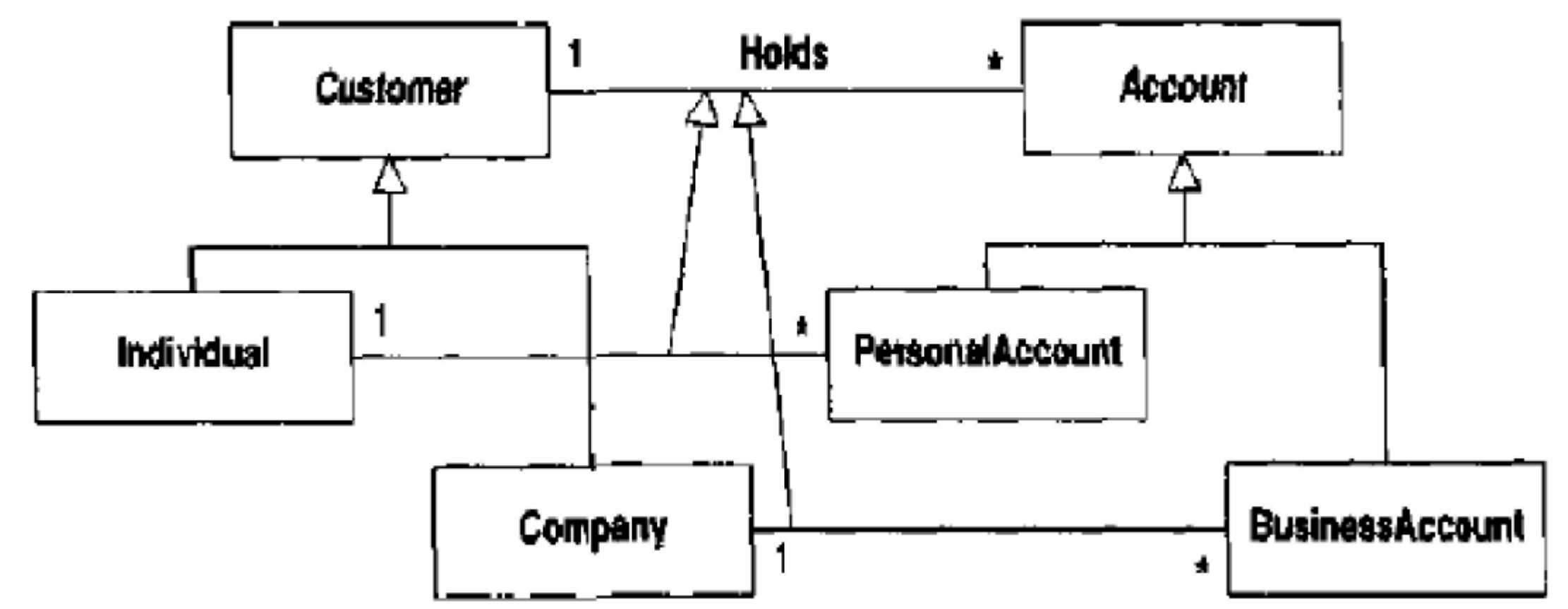
# Ograničenja

(constraints)

# Motivacija


Customer
1
Holds
*
Account
Individual
1
*
PersonalAccount
Company
1
*
BusinessAccount

# Motivacija (2)

- Prethodni dijagram uvodi generalizaciju asocijacija da bi ograničio
  - Samo fizička lica mogu da posjeduju račune za građanstvo
  - Samo pravna lica mogu da posjeduju biznis račune

- Bez generalizacije značenje dijagrama bi bilo da bilo koji tip klijenta može da posjeduje bilo koji tip računa

# Ograničenja

- Složene situacije iz domena se jednostavnije i preglednije mogu opisati tekstualnim ograničenjima

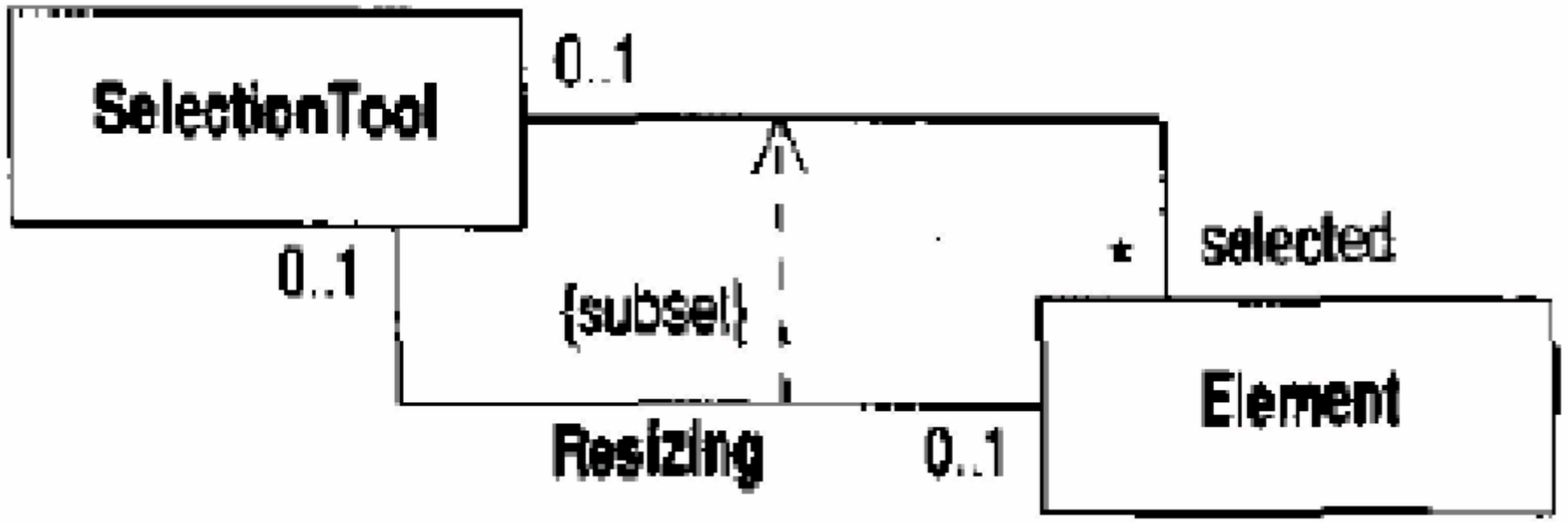
- Ograničenja se navode izmedju { i } i mogu da budu
  - Standardna ograničenja propisana u UML-u
  - Opšta ograničenja napisana u formi slobodnog teksta ili čak ciljnog programskog jezika
  - Napisana u jeziku OCL – object constraint language koji je dio UML-a

# Primjer 1


SavingsAccount
balance
deposit(amt)
withdraw(amt)
{Balance must stay within
range 0 - 250,000}

# Standardna ograničenja

- Ograničenja {new} i {destroyed}
- Ograničenje {subset}



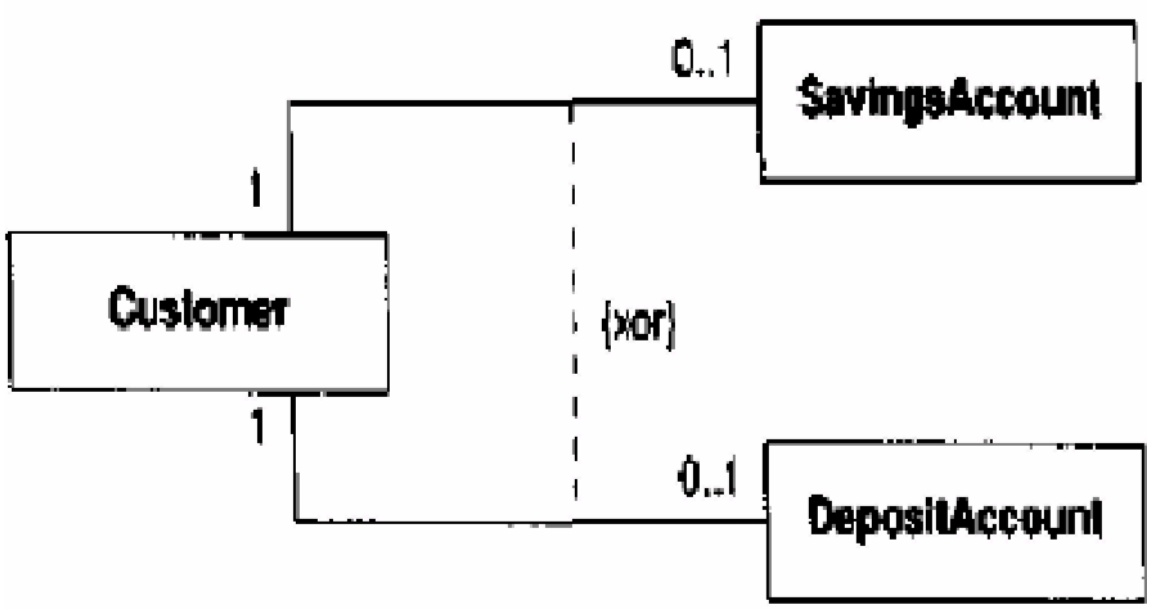
# Ograničenje subset

- Na prethodnom dijagramu dvije asocijacije su povezane relacijom zavisnosti sa ograničenjem *subset*

- Značenje je da skup instanci jedne asocijacije mora da bude podskup skupa instanci druge asocijacije, pri čemu obje asocijacije povezuju iste dvije klase
  - Kada se SelectionTool *s* koristi za resize elementa *e,* kreira se instanca asocijacije Resizing, dok ograničenje subset znači da isti taj element *e* mora da bude među selektovanim elementima od *s*

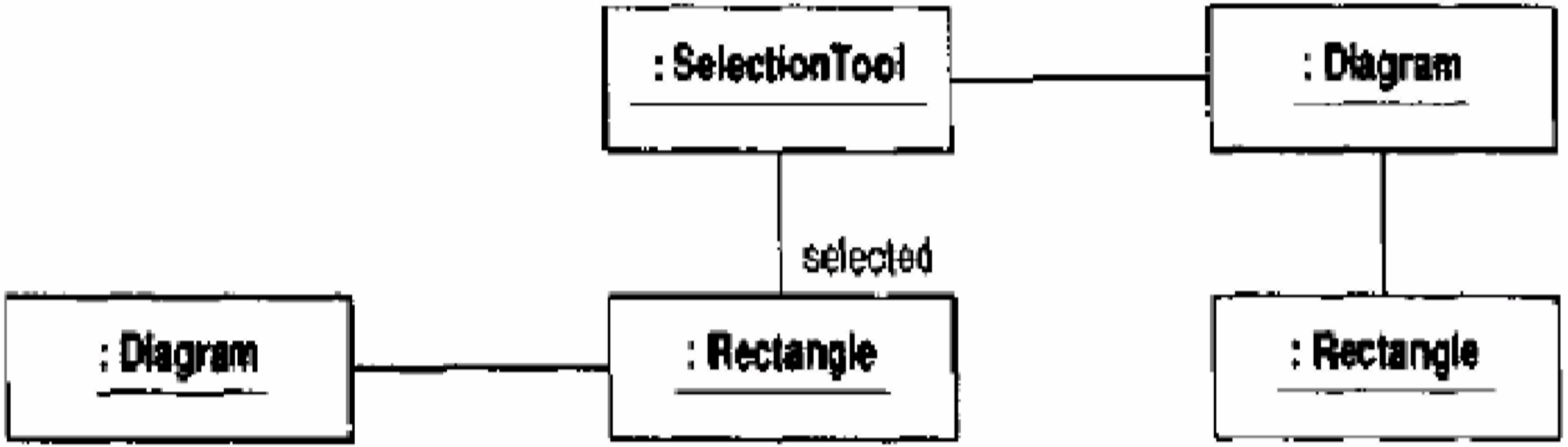
# Ograničenje xor

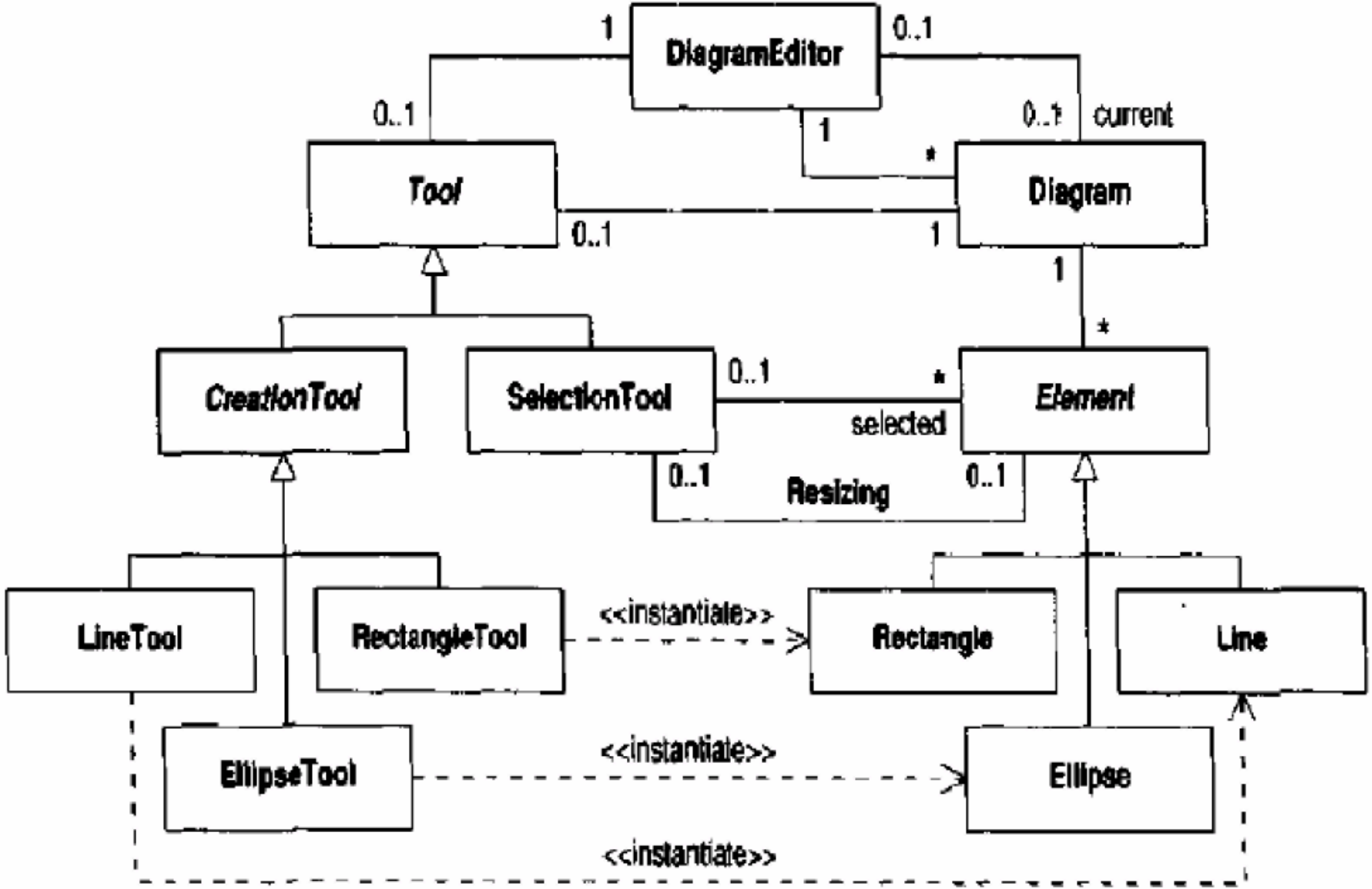
- Ograničenje xor može da se primijeni na parove asocijacija koje se „završavaju“ na istoj klasi gdje je multiplikativnost 1
- Ograničenje xor ima značenje da objekat „zajedničke“ klase ne može istovremeno da bude prisutan u obje asocijacije
  - Klijent ne može istovremeno da bude vlasnik štednog i tekućeg računa
  - Da li mora 0 da bude uključena u multiplikativnost na strani računa?



# Object constraint language - OCL

- OCL je formalni jezik koji omogućava specifikovanje opštih ograničenja za sve elemente na UML dijagramima, a naročito na klasnim dijagramima




1
DiagramEditor
0..1
0..1
current
1
*
Tool
Diagram
0..1
1
1
*
CreationTool
SelectionTool
0..1
*
Element
selected
0..1
Resizing
0..1
LineTool
RectangleTool
<<instantiate>>
Rectangle
Line
EllipseTool
<<instantiate>>
Ellipse
<<instantiate>>

# Primjer 2

- Potrebno je uvesti ograničenje da slektovani element mora pripadati istom dijagramu za koji je povezan objekat klase SelectionTool
  - Ovo ograničenje može da se implementira kao metoda klase SelectionTool. Instanca klase SelectionTool *s* povezana je sa dijagramom *d* i njom je selektovan skup elemenata *es*. Onda se za svaki element is skupa *es* provjeri da li je povezan upravo sa dijagramom *d*.
- Za formalno specifikovanje opšteg ograničenja u jeziku OCL potrebno je da postoji mogućnost za specifikovanje:
  - Konteksta ograničenja
  - Izraza za navigaciju kroz model
  - Logičkih uslova koji se odnose na relacije između konteksta i povezanih objekata

# Kontekst ograničenja

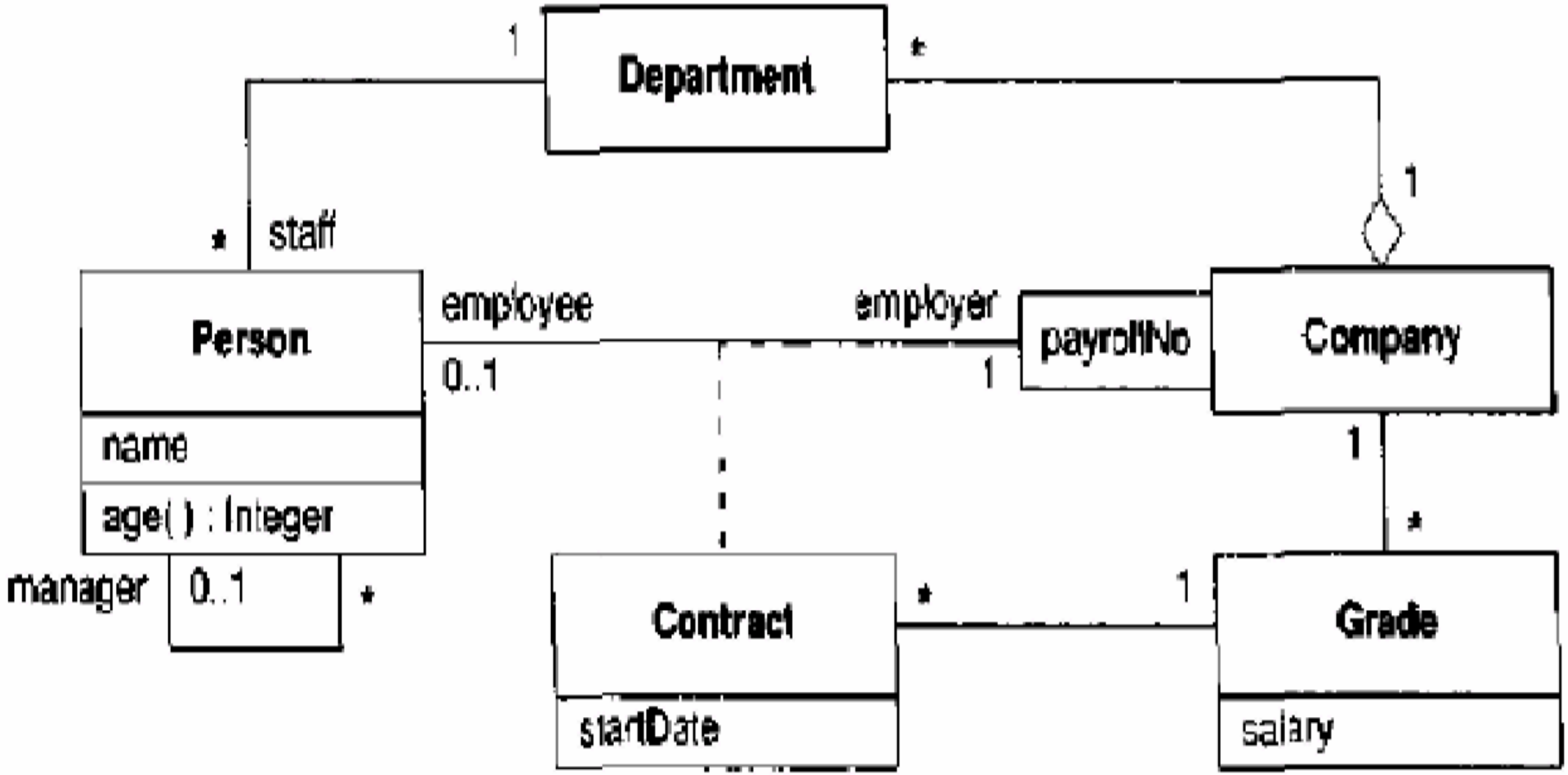
- Kontekst je u osnovi element modela za koji se definiše uslov, na primjer klasa
  - Navedeno je ograničenje da iznos na računu mora da bude između 0 i 250000, ograničenje se odnosi na sve instance klase SavingsAccount
  - Isto ograničenje može da se zada u tekstualnoj formi i da se ne prikazuje na dijagramu (self ukazuje na tekući objekat kao pokazivać this u C++ i omogućava definisanje uslova na nivou podatka člana)


```
SavingsAccount
self.balance > 0 and self.balance < 250000
```

# Izrazi za navigaciju

- Potrebno je specifikovati ograničenja na veze između elemenata modela
- OCL sa izrazima za navigaciju implementira mogućnost da se referenciraju objekti koji su u vezi sa kontekstim objektom
- Izraz za navigaciju locira željeni objekat tako što prati linkove između objekata počevši od kontekstnog objekta

# Primjer 3


Department
1
*
1
*
staff
Person
name
age( ) : Integer
employee
0..1
employer
1
payrollNo
Company
1
*
manager
0..1
*
Contract
startDate
*
1
Grade
salary

# Primjer 3 (2)

- Prethodni dijagram: kompanija sadrži skup departmana, zaposleni u kompaniju raspoređeni su u departmane, zaposleni mogu da imaju menadžere, kvalifikator za asocijaciju ukazuje na to da se svaki zaposleni identifikuje sa jedinstvenim payrollNo brojem u okviru kompanije, asocijativna klasa za prestavljanje ugovora između zaposlenog i kompanije, svaki ugovor vezan je za određenu poziciju (platu) u kompaniji

# Linkovi

- Primjer, skup zaposlenih u jednom departmanu dobija se sljedećim izrazom

  Department

  self.staff

- Primjer, skup deprtamana u kompaniju

  Company

  self.department

- Kada postoje bar dvije asocijacije između istih klasa moraju se imenovati uloge za asocijacije

# Kolekcije

- Kada OCL izraz može da vrati više od jednog objekta kaže se da izraz vraća kolekciju, primjer su prethodna dva izraza, u ostalim slučajevima kaže se da izraz vraća jedan objekat
- Primjer, departman u kome radi zaposleni

  Person

  self.departman
- Primjer, šef od zaposlenog, ovaj izraz može da ne vrati nijedan objekat

  Person

  self.manager

# Iteracije

- Primjer, svi zaposleni jedne kompanije

  Company

  self.department.staff

- Prethodni izraz računa se „postupno“, prvo se generiše spisak departmana unutar kompanije, a onda se za svaki departman generiše spisak ljudi koji tamo rade, da bi konačan rezultat bila unija skupova zaposlenih po departmanima

# Kvalifikovane asocijacije

- Primjer, kvalifikovana asocijacija omogućava selekciju određenog zaposlenog na osnovu jedinstvenog identifikatora 314159

  Company

  self.employee[314159]

- Primjer, menadžer od zaposlenog 314159

  Company

  self.employee[314159].manager

# Asocijativne klase

- Dozvoljeno je pristupati instanci objekata preko asocijativne klase

Grade

self.contract.employee

Person

self.contract.grade

# Tipovi u OCL-u

- Osnovni tipovi
  - Boolean
  - Real
  - Integer
  - String
- Tipovi iz modela su definisani klasama
  - Imaju atribute i metode
- Objekti vs. kolekcije

# Operacije nad objektima

- Primjer, godine i plata zaposlenog

  Person

  self.age

  self.contract.grade.salary

- Primjer, spisak imena zaposlenih u departmanu

  Department

  self.staff.name

# Tipovi kolekcija

- Primjer, pozicija (radno mjesto) zaposlenih u departmanu

  Department

  self.staff.conctract.grade
- Prethodna kolekcija je multiskup, jer je očekivano da više zaposlenih imaju jednu istu poziciju
- OCL uvodi pretpostavku da kada se izrazom obuhvata više od jedne asocijacije sa multiplikativnosti većom od 1, kolekcija koja se dobija u rezultatu je multiskup

# Operacija nad kolekcijama

- Primjer, operacija sum(), masa zarada na nivou departmana

  Department

  self.staff.contract.grade.salary->sum()

- Primjer, operacija size(), operacija asSet(), broj različitih pozicija u departmanu

  Department

  self.staff.contract.grade->asSet()->size()

# Operacije nad kolekcijama (2)

- Primjer, operacija select(), zaposleni sa zaradom većom od 50000

  Company

  self.employee->select(p:Person|p.contract.grade.salary>50000)
- Lokalna promjenljiva p:Person, često nije potrebno da se eksplicitno deklariše
- Primjer, menadžeri zaposlenih sa platama većim od 50000

  Company

  self. employee->select(contract.grade.salary>50000).manager

# Operacije nad kolekcijama (3)

- Primjer, operacija collect(), godine starosti zaposlenih u departmanu

  Department

  self.staff->collect(p:Person|p.age)

- Primjer, ukupna zarada svih zaposlenih uvećana za 10%

  Company

  self.contract.grade->collect(salary*1.1)->sum()

# Ograničenja

- Ograničenja su u OCL-u izrazi + logički operatori
- Osnovna ograničenja, najjednostavnija ograničenja iskazana kao poređenje na jednakost i nejednakost objekata i kolekcija, uz standardne operatore poređenja numeričkih vrijednosti
- Primjer, departman u kome zaposleni radi mora da pripada kompaniji za koju zaposleni radi

  Person

  self.employer = self.department.company

# Osnovna ograničenja

- Primjer, isEmpty(), zaposleni u kompaniji moraju biti stariji od 18 godina

  Company

  self.employee->select(age<18)->isEmpty()

ili

  self.employee->select(age<18)->size = 0

- Primjer, includes(), pozicija zaposlenog mora da postoji u skupu pozicija za tu kompaniju

  Person

  self.employer.grade->includes(contract.grade)

# Osnovna ograničenja (2)

- Primjer, includesAll, zaposleni u departmanu moraju da budu u skupu zaposlenih u kompaniji

  Departman

  self.company.employee->includesAll(self.staff)

# Složena ograničenja

- Logički operatori AND, OR, XOR, NOT, IMPLIES
- Primjer, svi zaposleni strariji od 50 godina moraju da imaju zaradu veću od 25000

Person

self.age>50 IMPLIES self.contract.grade.salary > 25000

# Iterativna ograničenja

- Ograničenja koja su definisana nad kolekcijama i testiraju se nad svim članovima kolekcije
- Primjer, forAll, za svaku poziciju u kompaniji mora da postoji radnik

  Company

  self.grade->forAll(g|NOT g.contract->isEmpty())
- Primjer, exists, vratiće TRUE ako je bar jedan član kolekcije zadovoljio ograničenje, svaki departman mora da ima menadžera

  Department

  self.staff->exists(e|e.manager->isEmpty())

# Iterativna ograničenja (2)

- Primjer, allInstances(), plate zaposlenih na nekoj poziciji su veće od 20000

  Grade

  Grade.allInstances->forAll(g|g.salary>20000)

- Primjer, ne postoje dvije različite pozicije sa istom platom

  Grade.allInstances->forAll(g:Grade|

  g<>self IMPLIES g.salary<>self.salary)

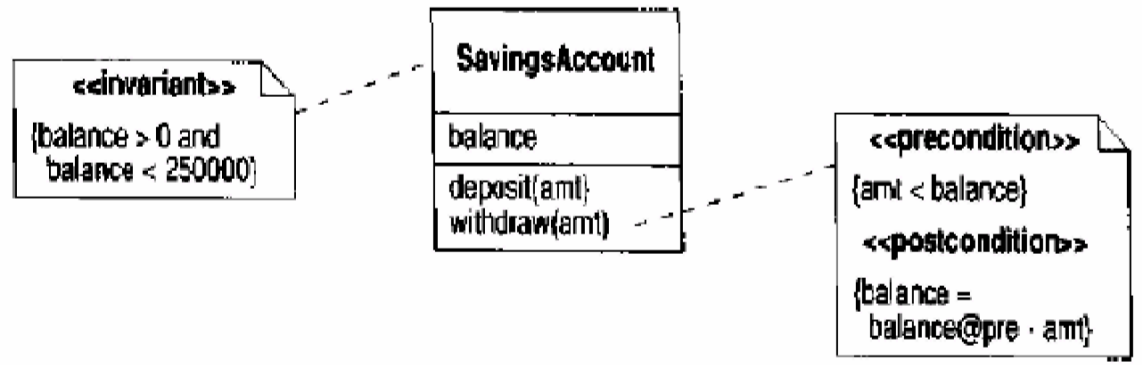
# Invarijanta klase

- Ograničenje koje zadovoljavaju sve instance klase vezano za vrijednosti podataka članova
- Primjer, iznos na štednom računu mora da bude između 0 i 25000

SavingsAccount

self.balance > 0

AND self.balance < 25000



# Pre/Post conditions

- Definiše se za metode klase, eksplicitno zahtijevaju čuvanje invarijante klase

SavingsAccount:withdraw(amt)

pre: amt < balance

post: balance = balance@pre - amt

# Generalizacija

Individual

self.account->forAll(a|a.oclType=PersonalAccount)