

Projektovanje softvera

Dijagrami klasa



Uvod

- Dijagram klasa prikazuje skup klasa, interfejsa, saradnji i drugih stvari strukture, povezanih relacijama
- Dijagram klasa je graf obrazovan od temena (stvari) povezanih granama (relacijama)
- Specificira logičke i statičke aspekte modela
- Elementi dijagrama klasa
 - stvari: klase, interfejsi, tipovi, izuzeci, šabloni, saradnje, paketi, ...
 - relacije: zavisnosti, generalizacije, asocijacije, realizacije
- Dijagrami klasa su najčešći u objektnom modeliranju
- Većina alata podržava generisanje skeleta koda iz dijagrama klasa

Klasifikator

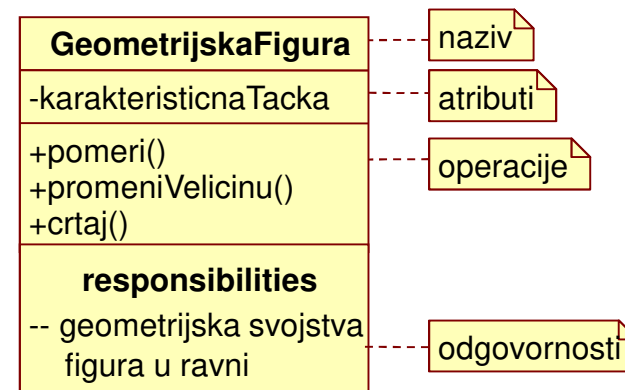
- Klasifikator je
 - karakterizacija primeraka (*instances*) sa zajedničkim karakteristikama
 - mehanizam koji opisuje strukturu i ponašanje (ima attribute i operacije)
 - apstraktna metaklasa
 - prostor imena
 - tip koji je moguća generalizacija ili specijalizacija drugog
 - potencijalno ugnežđen u drugi klasifikator
- Ne pojavljuju se svi klasifikatori na dijagramima klasa
- Konkretni klasifikatori su: klase, interfejsi, tipovi podataka, komponente, čvorovi, slučajevi korišćenja, ...
- Klasifikator ima odeljke, koji se ne moraju svi prikazati
 - odeljci mogu imati svoje nazive, da bi se izbegla dvoznačnost kada neki odeljak nedostaje

Klasa

- UML 2 definicija: klasa je opis skupa objekata koji dele istu specifikaciju karakteristika (*features*), ograničenja (*constraints*) i semantike
 - karakteristike: atributi i operacije
- Klasa implementira jedan ili više interfejsa
- Klase opisuju apstrakcije iz domena problema kao i apstrakcije iz domena rešenja
- Koriste se da reprezentuju
 - konkretne softverske stvari
 - konceptualne stvari

Simbol klase

- Simbol klase je pravougaonik podeljen horizontalnim linijama u odeljke
 - odeljak može imati i naziv odeljka
- Naziv klase
 - jednostavan:
`<naziv>`
 - sa putanjom:
`<naziv paketa>::`
`<jednostavan naziv>`
npr:
`java::awt::Rectangle`
- Odgovornosti klase
 - neformalna semantika
 - u zasebnom odeljku
 - slobodan tekst
 - svaka počinje sa --
- Svaka dobro strukturirana klasa bi trebalo da ima barem jednu i ne više od nekoliko odgovornosti



Atributi i operacije

- Atributi su imenovana svojstva klase koja opisuju opsege vrednosti koje pojave tog svojstva mogu sadržati
- Drugi nazivi: članovi podaci (C++), polja (Java)
- Atributi su strukturne karakteristike (*features*) klase
- Notacija: ime, opciono sa tipom i podrazumevanom vrednošću
- Primer: `izbor:Boolean=false`
- Operacije su servisi koji se mogu zahtevati od nekog objekta klase
- Notacija: potpis koji sadrži listu parametara sa eventualnim tipovima i podrazumevanim vrednostima, kao i tipom rezultata
- Primer:
`pomeri(novaPozicija:Pozicija=koordinatniPocetak):Pozicija`

Dodatne mogućnosti

- Simbol klase može sadržati prazne odeljke, a može biti i bez odeljaka
 - prazan odeljak atributa/operacija ne znači da ih klasa nema, već da nisu relevantni za dati pogled (dijagram)
 - mogu se koristiti i tri tačke (. . .) da naznače postojanje dodatnih atributa/operacija
- Atributi/operacije se mogu grupisati, a svakoj grupi može da prethodi opisni prefiks
- Apstraktna klasa i apstraktna operacija
 - naziv se piše *italikom* ili uz ograničenje `{abstract}`
- Zajednički članovi (atributi i operacije) klase - podvučeno
- Pravo pristupa članu (vidljivost, *visibility*) znak se piše ispred atributa/operacije:
 - javni član: + (podrazumevano)
 - zaštićeni član: #
 - paketski član: ~
 - privatni član: -

Potrosac

Potrosac

Tipovi podataka

- Tipovi podataka su tipovi čije vrednosti (podaci) nemaju identitet
- Vrste tipova podataka:
 - apstraktni tipovi podataka (`<<dataType>>`)
 - primerci imaju samo vrednost, bez identiteta (npr. `Datum`)
 - tipovi nabiranja (`<<enumeration>>`)
 - uzimaju vrednost iz definisanog skupa simboličkih imena (npr. `boolean`)
 - primitivni tipovi (`<<primitive>>`)
 - postojeći prosti tipovi podataka u jeziku implementacije (npr. `int`)

`<<dataType>>`
Datum

`<<enumeration>>`
boolean
true
false

`<<primitive>>`
int
{vrednosti u opsegu:
od $-2^{31}-1$ do 2^{31} }

Osobine klasa

- Multiplikativnost – osobina koja ograničava broj primeraka klase
- Multiplikativnost se navodi u gornjem desnom uglu
- Specifične vrednosti multiplikativnosti:
 - 0 – uslužna klasa, sadrži samo zajedničke (statičke) attribute i operacije
 - 1 – unikat (*singleton*) – klasa koja ograničava broj primeraka na 1
- Podrazumevani slučaj je proizvoljan broj primeraka
- Koren hijerarhije klasa (*root*) je klasa koja nema roditelje
- List hijerarhije klasa (*leaf*) je klasa koja nema potomke, tj. ona iz koje se ne može izvoditi
- Osobine `{root}` i `{leaf}` se pišu u odeljku naziva klase

Sintaksa atributa

- Sintaksa atributa:
`[pravo_pristupa] [/] ime [:tip] [multiplikativnost]
[=inicijalna_vrednost][{osobine}]`
- Simbol / označava da je atribut "izveden" (može se "izračunati" na osnovu drugih)
- Multiplikativnost atributa (podrazumevano 1), navodi se u zagradama []
 - specificira se kao interval celih brojeva, gornja granica može biti neograničena (*)
 - primer: `consolePort:Port[2..*]`
- Osobine atributa se navode razdvojene zarezima; neke od njih:
 - `readOnly` – vrednost se ne može menjati, dodavati ni uklanjati nakon inicijalizovanja
 - `ordered` – vrednosti elemenata su uređene
 - `unique` – vrednosti elemenata (u jednom objektu zbirke) su jedinstvene (skup)
 - `bag` – isto kao `unordered nonunique` (nije skup, nije uređena zbirka)
 - `seq` ili `sequence` – isto kao `ordered nonunique` (uređen niz elemenata sa ponavljanjem)
 - `composite` – atribut složene strukture
 - `redefines ime` – redefiniše nasleđen atribut `ime`
 - `subsets ime` – podskup skupa vrednosti atributa `ime`

Sintaksa operacije

- Sintaksa operacije:
`[pravo_pristupa] ime ([lista_parametara])
[: tip_rezultata] [{osobina}]`
- Sintaksa parametra u listi:
`[smer] ime : tip [multiplikativnost]
[= podrazumevana_vrednost] [{osobina}]`
- Smer može biti:
 - `in` – ulazni parametar, ne sme biti modifikovan (podrazumevano)
 - `out` – izlazni parametar, mora se inicijalizovati u metodu
 - `inout` – ulazno-izlazni parametar, inicijalizovan pre poziva, može se modifikovati
- Osobine operacije:
 - `query` – izvršenje ne menja stanje objekta, operacija je čista funkcija bez bočnih efekata
 - `exception lista` – operacija može da baca izuzetke iz liste
 - `leaf` – operacija nije polimorfna i ne može se redefinisati u izvedenoj klasi
 - `concurrency = vrednost` – šta garantuje operacija pri izvršenju u konkurentnoj sredini
 - `sequential` – pozivaoci moraju obezbediti da je samo jedna nit u jednom trenutku poziva
 - `guarded` – operacija garantuje sekvencijalizaciju svih poziva svih štićenih operacija
 - `concurrent` – operacija se izvršava kao atomska

Aktivne klase

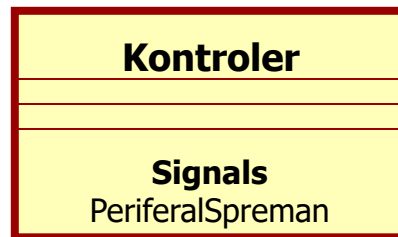
- Aktivni objekat je onaj koji poseduje vlastiti tok kontrole koji se odvija simultano sa drugim tokovima kontrole
- Simultano – konkurento (isti resurs) ili paralelno (posebni resursi)
- Aktivna klasa je klasa čiji su primerci aktivni objekti
- Aktivni objekti su “koreni” pojedinih tokova kontrole
 - oni pozivaju svoje ili metode drugih objekata
- Tok kontrole se pokreće/zaustavlja:
 - kada se aktivni objekat stvara/uništava, ili
 - mogu postojati i posebne operacije za pokretanje i zaustavljanje
- Mnogi jezici podržavaju aktivne objekte (Ada, Java, Smalltalk)
- Aktivni objekti i klase - mogu se pojaviti u dijagramima gde i pasivni

Procesi i niti

- Proces (*process, task*) je "teški" (*heavyweight*) tok kontrole
 - ima vlastiti adresni prostor
- Nit (*thread*) je "laki" (*lightweight*) tok kontrole
 - deli zajednički adresni prostor sa drugim nitima
- Jedan proces može sadržati više niti
- Proces je jedinica konkurentnosti u operativnim sistemima:
 - procesi konkurišu za resurse
- Primeren mehanizam za komunikaciju:
 - procesi - razmena poruka (*message passing*)
 - niti - deoba zajedničkih podataka (*data sharing*)

Aktivne klase - notacija

- Grafički simbol aktivne klase:
- UML 1:

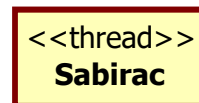


UML2:

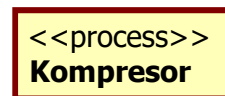


- poseban odeljak se predviđa za signale koje klasa može primati

- Nit (thread)



- Proces (process)



UML 1

UML 2

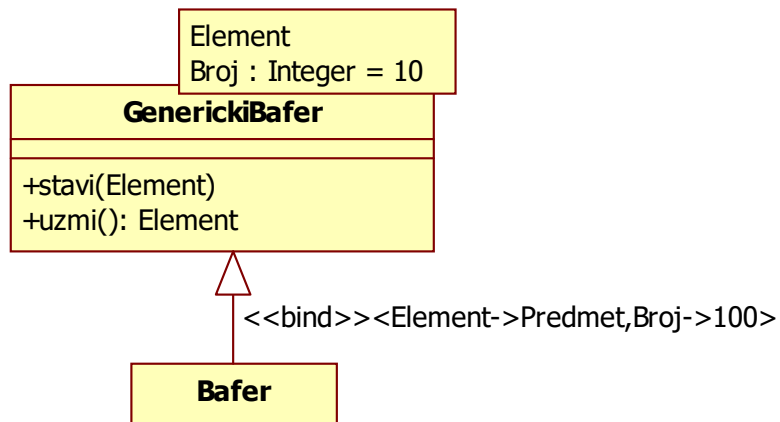
Šabloni

- Šablon (*template*) je parametrizovani element
 - C++, Ada, Java podržavaju šablone (generike)
- Primer na C++:

```
template <class Element, int Broj> class GenerickiBafer{
public:
    virtual void stavi(const Element&);
    virtual Element& uzmi();
    ...
};
typedef GenerickiBafer<Predmet,100> Bafer; // eksplicitno generisanje
GenerickiBafer<Predmet,100> bafer;      // implicitno generisanje
```

Šabloni - notacija

- Primer generičkog bafera:
 - definicija šablona i eksplicitno generisanje iz njega:

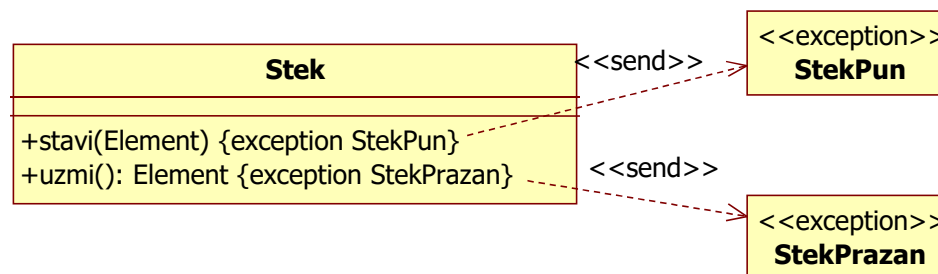


implicitno generisanje:

```
GenerickiBafer<Element->Predmet, Broj->100>
```


Izuzeci

- Izuzeci su vrste signala koji se modeliraju stereotipom klase `<<exception>>`
- Signal šalje operacija koja izaziva izuzetak
 - modelira se stereotipom `<<send>>` relacije zavisnosti
 - samo vizuelna informacija, jer operacija već navodi kao osobinu `exception`
- Primer:



Relacije

- Na klasnim dijagramima se pojavljuju sve četiri (prve tri su češće) vrste relacija:
 - zavisnost (*dependency*)
 - asocijacija (*association*)
 - generalizacija (*generalization*)
 - realizacija (*realization*)

Zavisnost

- Povezuje stvari kod kojih izmena nezavisne stvari utiče na ponašanje zavisne stvari
- Zavisna stvar koristi nezavisnu stvar
- Grafička notacija: klasa A zavisi od klase B
- Često se koristi kad je jedna klasa (B) tip parametra operacije druge klase (A)



Generalizacija

- Povezuje opštije sa specijalizovanim stvarima
 - opštija stvar – superklasa, natklasa, osnovna klasa ili roditelj
 - specijalizovana stvar – subklasa, potklasa, izvedena klasa ili dete
- Grafička notacija: klasa A je dete, B je roditelj
- Drugi nazivi relacije:
 - vrsta (*is-a-kind-of*),
 - izvođenje (A se izvodi iz B),
 - nasleđivanje,
 - proširivanje
- Generalizacija znači:
 - objekti dece se mogu pojaviti gde god se očekuje objekat roditelja
- Deca nasleđuju osobine svojih roditelja, attribute (strukturu) i operacije (ugovor)
- Operacija deteta koja ima isti potpis kao operacija roditelja redefiniše operaciju roditelja
- Redefinicija operacije omogućava njeno polimorfno ponašanje
- Klasa koja ima samo jednog roditelja koristi jednostruko nasleđivanje
- Klasa koja ima više roditelja koristi višestruko nasleđivanje



Asocijacija

- Specificira da li su primerci jedne stvari povezani sa primercima druge stvari
- Asocijacija je strukturna relacija
- Asocijacija može biti unidirekcionalna ili bidirekcionalna
 - jednosmerna, odnosno dvosmerna, navigabilnost
- Dozvoljena je i asocijacija sa samim sobom
 - postoje veze između objekata iste klase
- Uobičajeno je da asocijacija povezuje dve klase (binarna asoc.)
- Moguće je da asocijacija povezuje i više klasa (n-arna asocijacija)
- Grafička notacija: klasa A je u asocijaciji sa klasom B

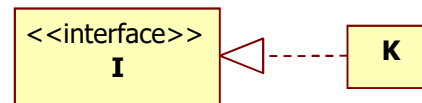


Realizacija

- Semantička relacija između klasifikatora od kojih jedan specificira ugovor a drugi garantuje njegovu implementaciju

- Grafička notacija:

- kanonička forma:

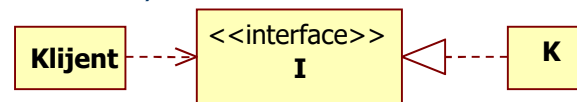


- skraćena forma:



- Korišćenje interfejsa (relacija zavisnosti)

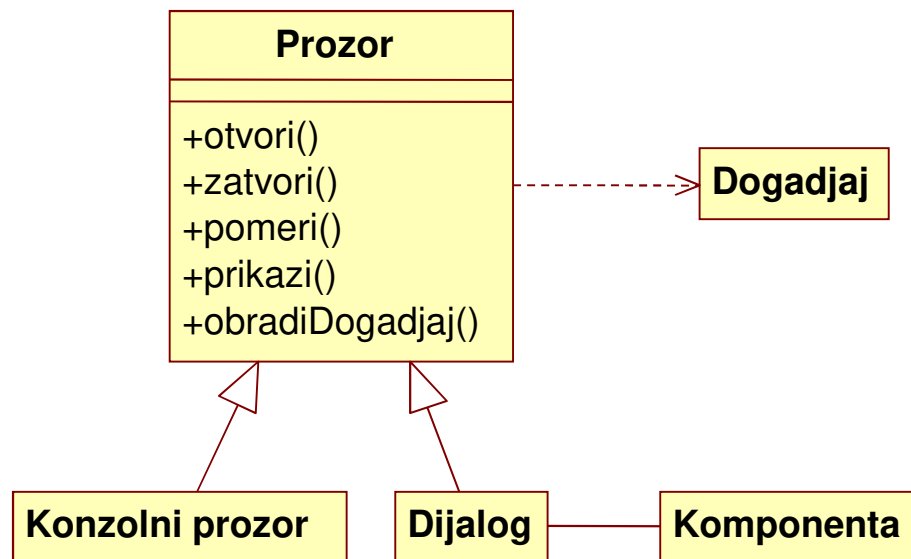
- kanonička forma:



- skraćena forma (*ball & socket*):



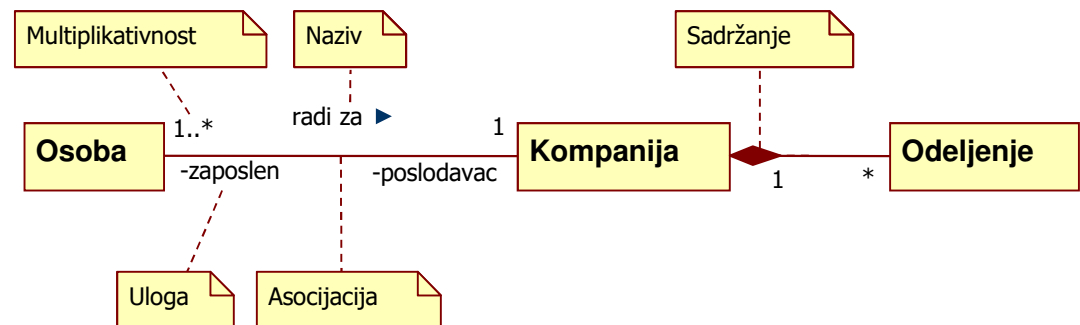
Primer osnovnih relacija



Ukrasi asocijacije

- Na asocijaciji se mogu pojaviti sledeći ukrasi:


- naziv
- smer čitanja
- uloge
- navigabilnost
- multiplikativnost
- sadržanje (agregacija ili kompozicija)
- pravo pristupa (vidljivost) kraju preko asocijacije



Multiplikativnost

- Na jednoj strani asocijacije označava se broj objekata sa te strane koji su u vezi sa tačno jednim objektom sa druge strane relacije
- Može biti:
 - tačno n , gde n može biti proizvoljan pozitivan broj (npr. 1)
 - proizvoljno mnogo, uključujući nijedan (*),
 - opseg (npr. 0..1 ili 2..*)
- UML 2.0 više ne dozvoljava izraze sa diskontinuitetom
 - npr. 0..1, 3..4, 6..* – proizvoljan broj osim 2 i 5
- Ograničenja uz multiplikativnost:
 - {ordered}, {unique} – kao kod atributa

Agregacija

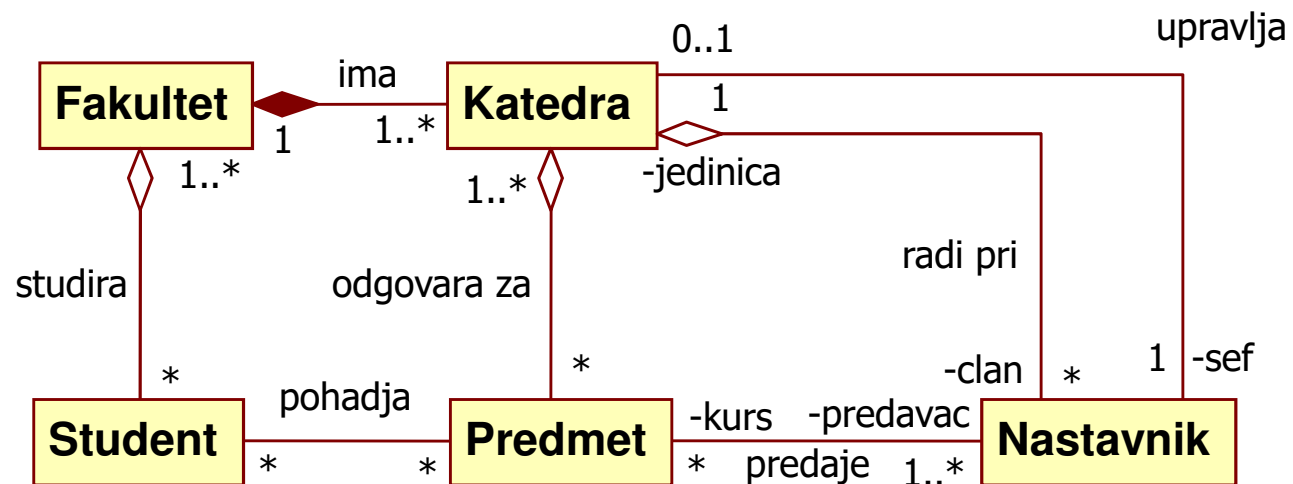
- Vrsta asocijacije kod koje jedna strana igra ulogu celine a druga ulogu dela (*whole/part*)
- Celina sadrži delove ("*has-a*" relacija)
- Agregacija najčešće označava grupisanje delova
- Agregacija ne govori ništa o uzajamnom odnosu životnih vekova celine i dela
- Deo u agregaciji može biti zajednički deo više celina
- Grafička notacija: 

Kompozicija

- Asocijacija kod koje postoji odnos celina/deo, ali je celina odgovorna za životni vek dela
- Sadržanje sa strogim vlasništvom
- Koincidentni životni vek dela u odnosu na celinu
 - deo može nastati u toku života celine i može biti uništen pre uništenja celine
- Deo u kompoziciji može biti deo samo jedne celine
- Grafički notacija:

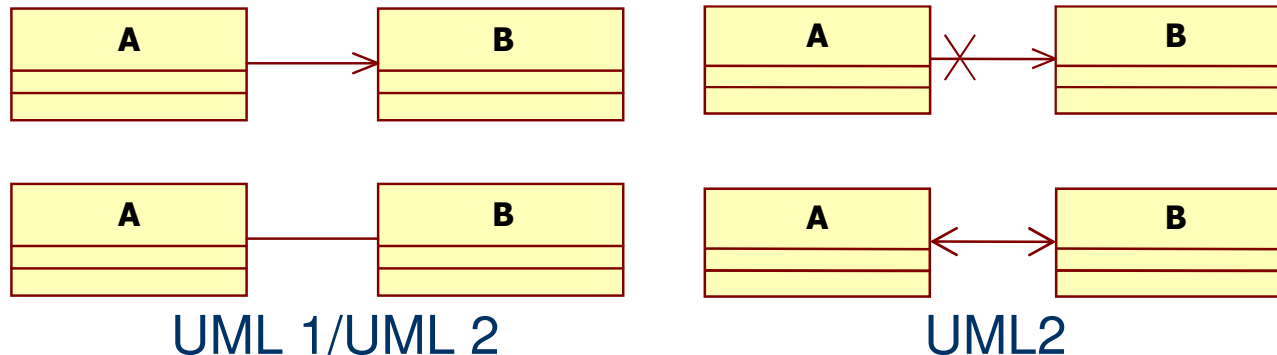


Primer asocijacija



Navigabilnost

- Strelica označava navigabilnost u naznačenom pravcu
- Krstić označava da nema navigabilnosti prema označenoj strani
- Za asocijaciju bez ukrasa navigabilnosti se smatra da je navigabilnost neodređena
- Grafička notacija:



Pravo pristupa preko asocijacije

- Ograničava vidljivost (*visibility*) objekata klasa u asocijaciji za spoljašnji svet
- Označava se sa +, #, -, ~ ispred imena uloge odgovarajuće strane relacije
- + znači da objektima sa te strane mogu pristupati svi objekti preko objekta sa druge strane
- - znači da objektima klase sa te strane mogu pristupati samo objekti klase sa druge strane
- # znači da i objekti klase izvedenih iz klase sa drugog kraja asocijacije imaju pristup
- ~ znači da i objekti klase iz istog paketa kao klasa sa drugog kraja asocijacije imaju pristup
- Primer:



- Podrazumevan je javni vizibilitet uloge u asocijaciji

Ugneždivanje

- Označava pojam kada je klasa B deklarisana u prostoru imena klase A



- Relacija ne implicira relaciju između objekata odgovarajućih klasa