Beogradska akademija poslovnih i umetničkih strukovnih studija

Projektovanje informacionih sistema

UML – Dijagram klasa

dr Rade Matić

UML – Dijagram klasa

Sa tačke gledišta njegovog korišćenja objekat se može tretirati kao crna kutija. Neophodno je opisati i njegovu unutrašnjost, njegova stanja, odnosno atribute i veze sa drugim objektima, kao i način implementacije njegovih operacija.

Primer: Studenti, profesori ...

Apstraktni mehanizmi koji su najzastupljeniji u praksi

Postupak apstrahovanja	**Postupak konkretizovanja**	**Apstrakcija**	**Konkretizacija**	**Relacija apstrakcije**
Klasifikacija	Instanciranje	Tip (klasa)	Instanca (pojavljivanje)	Je tipa
Generalizacija	Specijalizacija	Nadtip	Podtip	Generalizacija
Agregacija	Dekompozicija	Agregirani Objekat	Agregirajući Objekat	Je deo od
Projekcija	Elaboriranje	Projekcija	Elaboracija	Profinjuje
Učaurenje	Realizacija	Specifikacija	Implementacija	Realizuje

UML – Dijagram klasa

Klasa predstavlja skup objekata sličnih karakteristika. Klase opisuju različite tipove objekata koje sistem može da ima, a DK pokazuju te klase i njihove veze.

SK opisuju ponašanje sistema kao skup poslova. Klase opisuju tipove objekata koji su potrebni da bi sistem mogao da obavi te poslove. Opisivanjem (specifikacijom) klase opisuju se i svi objekti koji joj pripadaju.

Zato se može reći da je **klasa** definicija, odnosno mustra (templejt) koja omogućava da se kreira novi objekat koji joj pripada.
Klasa opisuje kako je objekat interno struktuiran, koji su njegovi atributi i veze, a ne samo koje se operacije sa objektom mogu izvesti.

UML – Dijagram klasa

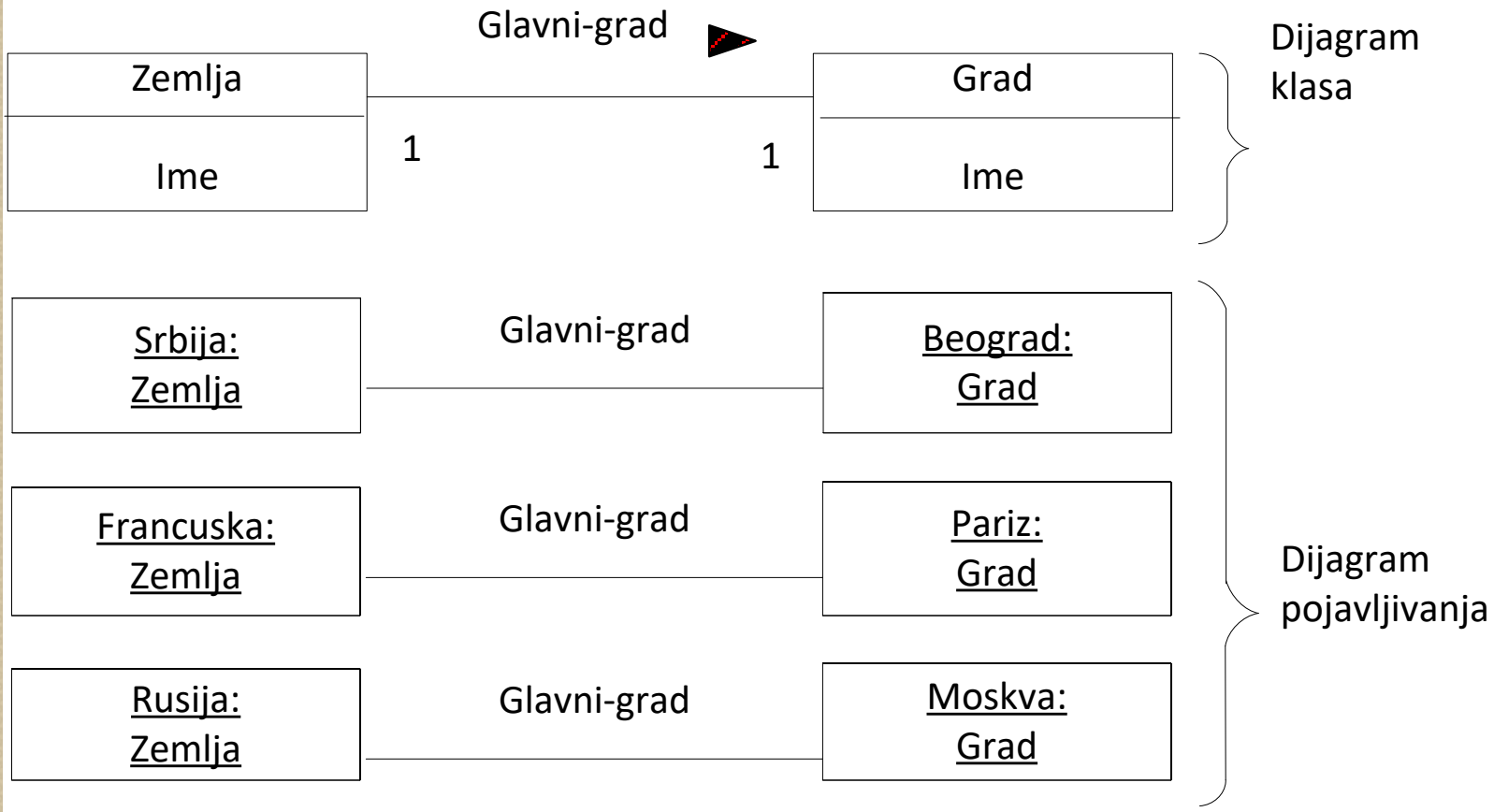
Zbog toga se uvodi pojam apstrakcije **klasifikacije** i definiše se pojam **klase objekata** kao skupa objekata sličnih karakteristika: atributa, veza sa drugim objektima i operacija (ponašanja).

Istovremeno se uvode i osnovne grafičke notacije za tzv. **objektni dijagrami**. Objektni dijagrami predstavljaju formalnu grafičku notaciju za prikazivanje objekata, klasa i njihovih međusobnih veza.

Postoje dva tipa objektnih dijagrama: **dijagrami klasa (DK)** koji prikazuje klase objekata i njihove međusobne veze i **dijagrami objekata ili pojavljivanja (DO)** koji prikazuju neki poseban skup objekata i njihove međusobne odnose.

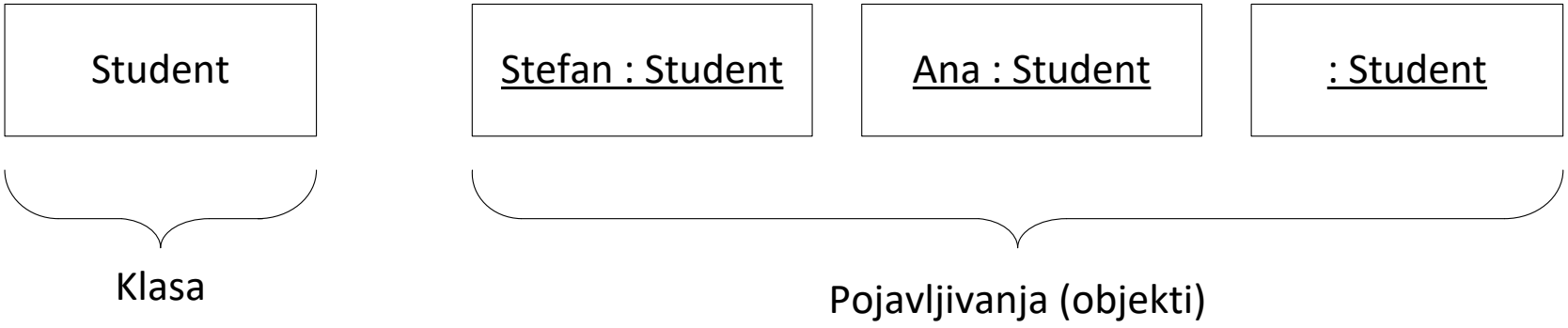
Očigledno je da se DK koriste za modelovanje sistema, dok se dijagrami objekata (pojavljivanja) pretežno koriste da prikažu primere koji objašnjavaju delove dijagrama klasa.

UML – Dijagram klasa



UML – Dijagram klasa

Ponekad se pojam klase i pojam tipa objekta izjednačavaju. Klasa se međutim se može tretirati kao jedna (od više mogućih) implementacija tipa objekta.
Objekat koji pripada nekoj klasi naziva se **pojavljivanje** klase. Ako se klasa tretira kao generator sličnih objekata, pojavljivanje je objekat koji je kreirala neka klasa. Zato su često pojmovi **pojavljivanja** i **objekta** sinonimi.
Na slici je prikazan **najednostavniji dijagram objekata i dijagram klasa** koji pokazuje odnos klase i njenih pojavljivanja (objekata).



UML – Dijagram klasa

U softveru, mnogi programski jezici neposredno podržavaju koncept klase. Svaka klasa mora imati **ime** da bi se razlikovala od ostalih klasa. Ime je slovni niz znakova. Kada ime samo stoji, zove se **jednostavno ime**.

Kada je naveden i paket u kome se klasa nalazi, u obliku prefiksa, tada se zove **ime sa putanjom**. Ime klase može sadržati bilo koji broj slova, brojeva i određenih interpunkcijskih znakova (osim dvotačke).

U praksi, ime klase je **kratka imenica ili fraza** izvedena iz rečnika sistema koji se modeluje. **Uobičajeno je da prvo slovo** svake riječi u imenu klase bude veliko, npr. Kupac ili SenzorTemperature.

UML – Dijagram klasa

Atribut predstavlja osobinu klase koja opisuje neku vrednost (stanje, informaciju) koju će sadržati svaki objekat posmatrane klase.

Navode se u okviru klase, a svaki atribut može da poseduje sledeća svojstva: **vidljivost, ime, tip, multiplicitet, podrazumevanu vrednost, kao i opis nekih dodatnih svojstava atributa** (npr. čitljivost koja označava da se atribut može samo čitati).

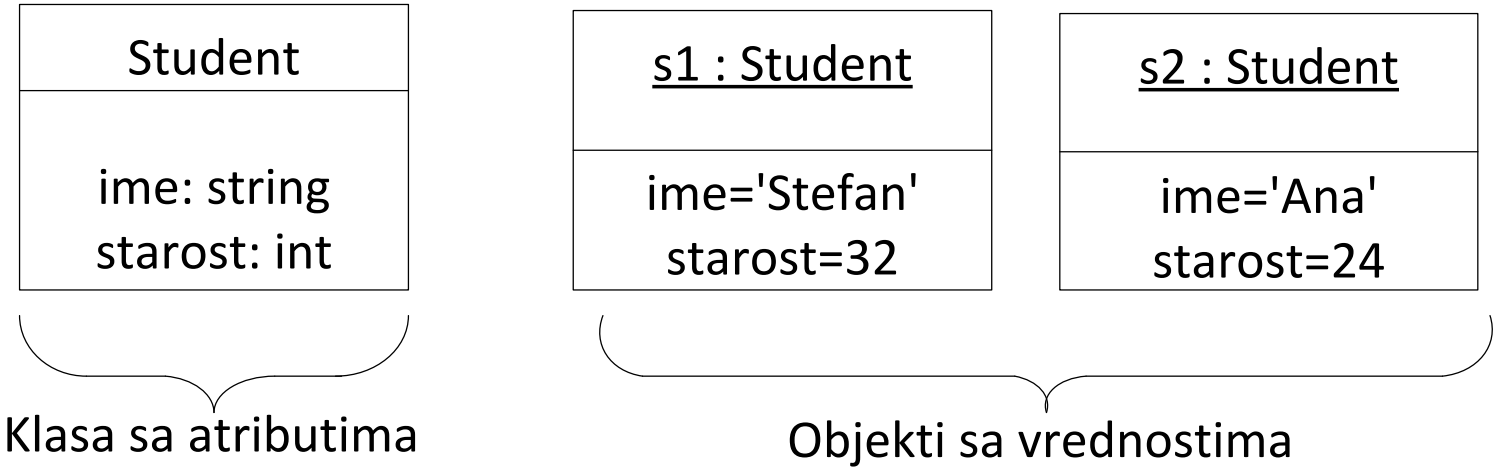
UML – Dijagram klasa

Klasa može imati nijedan ili više atributa. U datom trenutku, objekat klase ima određene vrednosti za svaki od svojih atributa klase.

Atributi se predstavljaju svojim imenom. U praksi, ime atributa je **skraćena imenica ili fraza**, koja predstavlja određenu osobinu razmatrane klase.

Uobičajeno je da se **prva reč u imenu atributa piše malim slovom**, a ostale velikim. Na primer, atributi klase *Osoba* su *ime, starost, težina, bojaOčiju* i slično.

UML – Dijagram klasa



UML – Dijagram klasa

Operacije opisuju poslove koje će objekat, kao konkretna pojava klase, znati da obavi.

Kada se od objekta zahteva da obavi neki posao, to se može zahtevati samo preko operacije koju on poseduje.

I operacije, kao i atributi poseduju odgovarajuća svojstva, i to: **vidljivost, ime, listu parametara, tip rezultata operacije** itd.

UML – Dijagram klasa

Operacija je realizacija nekog zadatka, koga može zahtevati svaki objekat klase u cilju uticaja na ponašanje.
Drugim rečima, **operacija** je apstrakcija onog što se može učiniti nad objektom i što je zajedničko za sve objekte jedne klase.

Klasa može imati nijednu, jednu ili više operacija. U praksi, **ime operacije je kratki glagol ili fraza**, koja predstavlja određeno ponašanje razmatrane klase.

Obično se prva reč u imenu operacije piše malim slovom, a ostale velikim, kao na primer *pomeri* ili *jePrazan.*

UML – Dijagram klasa

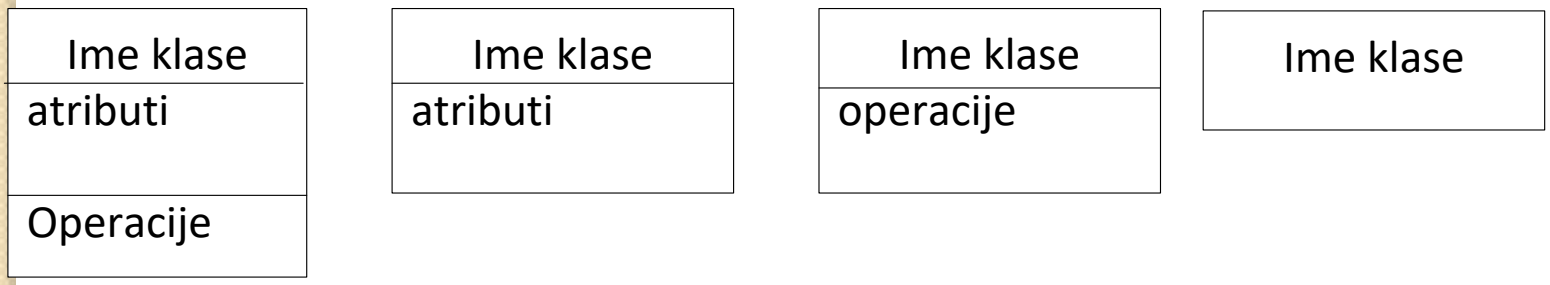
Pojam metode i pojam operacije se u OO pristupima često izjednačuju. Ovde se pretežno koristi pojam operacije, a pod metodom će se podrazumevati implementacija neke operacije u datoj klasi. Metode koje odgovaraju jednoj operaciji obavljaju isti zadatak (imaju istu specifikaciju), ali mogu imati različit kod.

Osoba
ime: string starost: int
promeniPosao() promeniAdresu()

Fajl
naziv: string velicina: int
stampaj() Kopiraj()

Objekat
boja: string polozaj: string
Pomeri (delta:Vector) Selektuj (p:Tacka):boolean

UML – Dijagram klasa

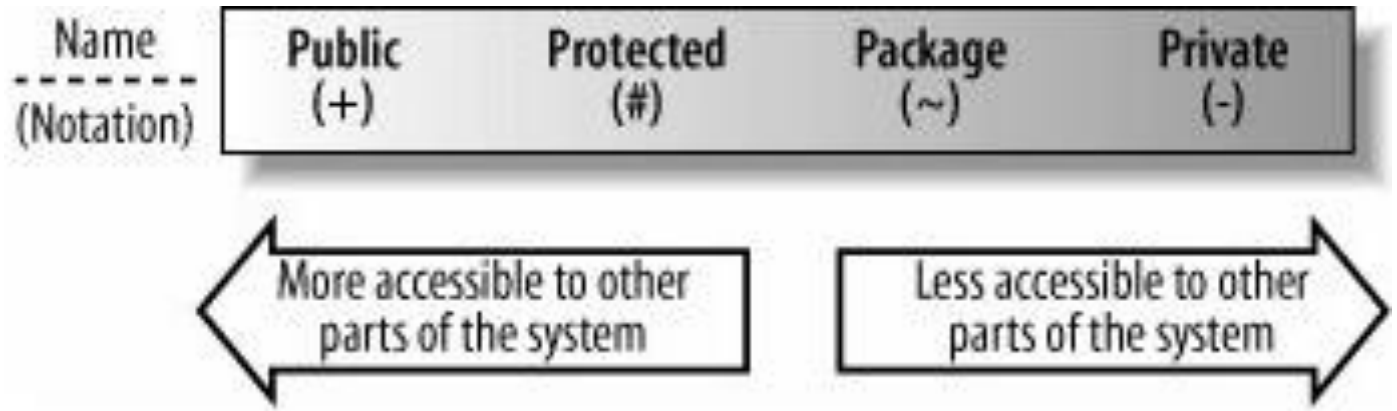


Četiri načina prikazivanja UML klase

UML – Dijagram klasa

Korišćenjem ***vidljivosti***, klasa selektivno pokazuje svoje operacije i podatke drugim klasama. Primenom karakteristika vidljivosti, možete kontrolisati pristup atributima, operacijama, pa čak i celim klasama radi efikasne primene enkapsulacije.

Postoje **četiri opšte različite vrste vidljivosti** koje se mogu primeniti na elemente UML modela.



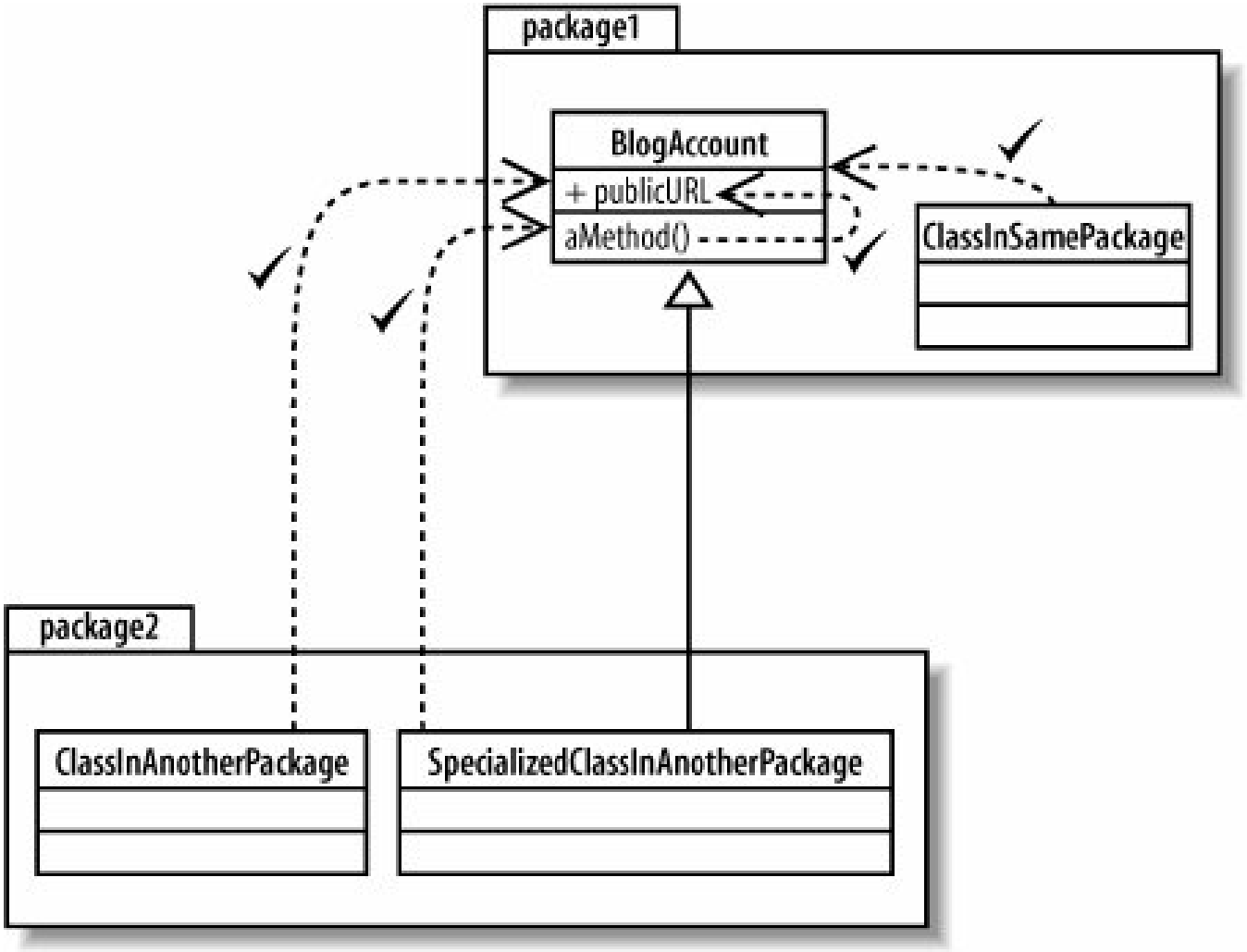
UML – Dijagram klasa

Javna vidljivost (engl. public visibility)
Počevši od karakteristike sa najmanjim ograničenjem pristupa (*more accessible to other parts of the system*), odnosno od najpristupačnije vidljivosti, ona se opisuje znakom plus (+) pre pridruženog atributa ili operacije.

Ponekad se podrazumeva da je reč o javnoj vidljivosti iako nema nikakvog znaka ispred atributa ili operacije. Atribut ili operaciju treba deklarisati kao javnu samo ukoliko želimo da budu dostupni direktno bilo kojoj drugoj klasi.

UML – Dijagram klasa

Korišćenjem javne vidljivosti, svaka klasa unutar modela može pristupiti atributu *publicURL*.



UML – Dijagram klasa

Važno je to da se javni interfejs klasa menja što je manje moguće radi sprečavanja nepotrebnih promena kad god je klasa upotrebljena. Mnogi OO projektanti smatraju da je otvaranje atributa klase ka ostatku sistema isto što i „izlaganje naše kuće bilo kojoj osobi sa ulice bez provere pre nego što uđe“.

Najbolje je izbegavati javne atribute, ali uvek postoje izuzeci. Primer kod koga je prihvatljivo koristiti javni atribut, jeste kada je atribut konstanta koju može koristiti više različitih klasa.

Atributi koji se ponašaju kao konstante, tj. data im je inicijalna nepromenljiva vrednost, takođe imaju i ***readOnly*** svojstvo.
U takvoj situaciji, izlaganje takvog atributa ostatku sistema nije toliko opasno.

UML – Dijagram klasa

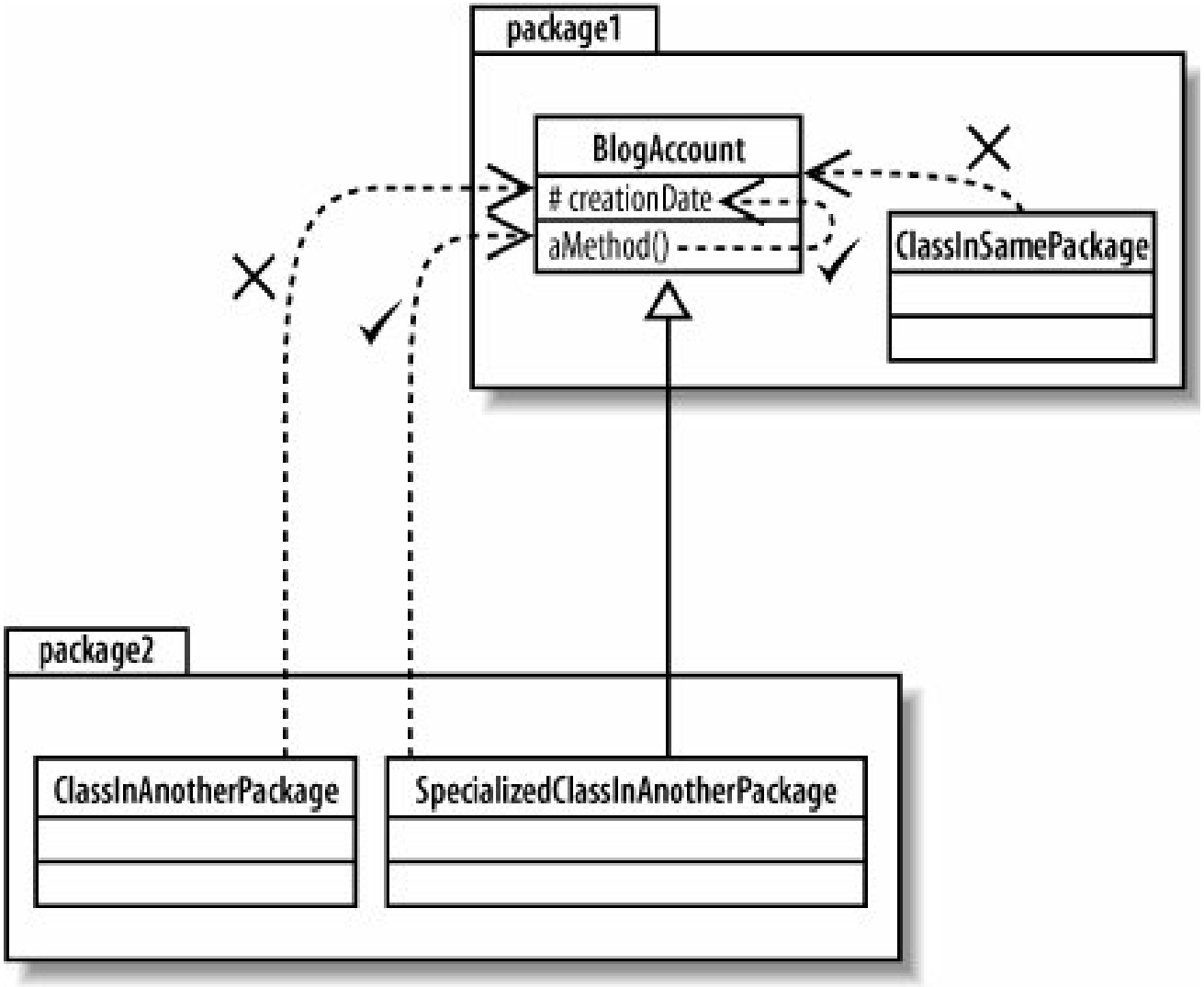
Zaštićena vidljivost (engl. protected visibility)
Zaštićeni atributi i operacije se specificiraju korišćenjem (#) simbola, i vidljiviji su ostatku sistema u odnosu na privatne atribute i operacije, ali su manje vidljivi od javnih.

Elementima u klasama koji su deklarisani sa ***protected***, se može pristupiti metodama koje su deo te iste klase i takođe metodama koje su deklarisane u svakoj klasi koje su nasleđene iz osnovne klase (klasa sa *protected* vidljivošću).

Protected elementima se ne može pristupiti klasom koja nije nasleđena iz naše klase bilo da je u istom paketu ili ne.

UML – Dijagram klasa

Bilo koja operacija u *BlogAccount* klasi ili klasama nasleđenim iz nje mogu pristupiti *protected creationDate* atributu.



UML – Dijagram klasa

Zaštićena vidljivost je značajna ukoliko želimo dozvoliti specijalizovanim klasama da pristupe nekom atributu ili operaciji u baznoj klasi bez otvaranja tog atributa ili operacije celom sistemu.

Za korišćenje *protected visibility* se može prosto reći:
„**Ovi atributi ili operacije su korisni unutar moje klase i klasama koje proširuju moju klasu (nasleđene klase, podklase), ali ih niko drugi ne može koristiti“.**

UML – Dijagram klasa

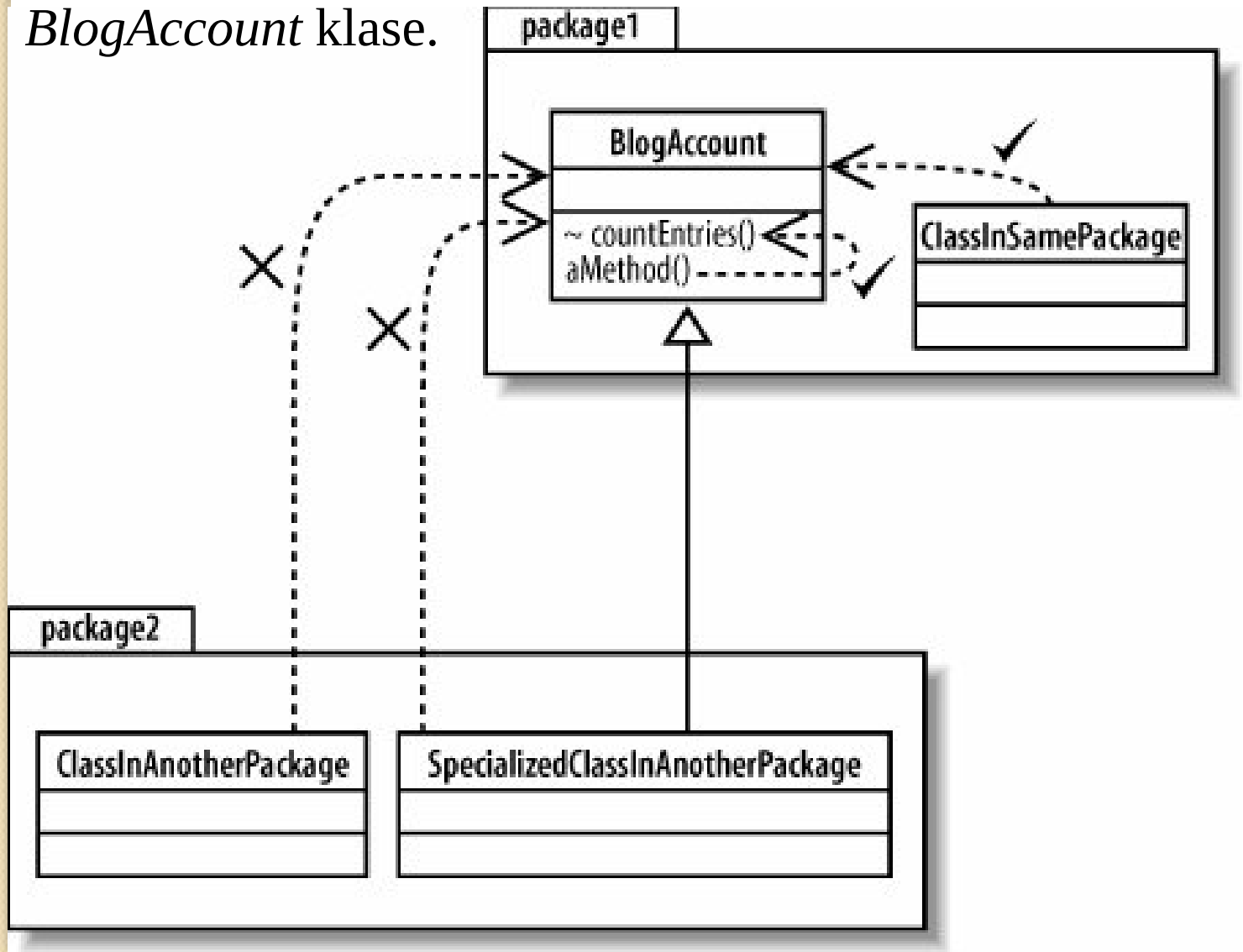
Paketna vidljivost (package visibility) se specificira znakom tilda (~). Ovde su paketi ključni faktor u određivanju koje klase mogu da vide neki atribut ili operaciju koja je deklarisana ovim znakom.

Pravilo je prilično jednostavno: **ako dodamo neki atribut ili operaciju deklarisanu kao *package visibility* nekoj klasi, onda bilo koja klasa iz istog paketa može direktno pristupiti tom atributu ili operaciji**. Klase izvan tog paketa ne mogu pristupiti zaštićenim atributima ili operacijama, čak i ako nasleđuju tu klasu.

Praktično, paketna vidljivost je najkorisnija kada želimo da deklarišemo kolekciju operacija i atributa kroz naše klase koje **mogu biti korišćene samo unutar našeg paketa**. Na primer, ako dizajniramo paket korisnih klasa i želimo ponovo da upotrebimo ponašanje između tih klasa, ali ne želimo da izlažemo ostatku sistema to ponašanje, deklarisaćemo paketnu vidljivost interno za te pojedine operacije u paketu.

UML – Dijagram klasa

Operaciju *countEntries* može pozvati bilo koja klasa iz istog paketa kao i *BlogAccount* klasa pomoću metoda unutar same *BlogAccount* klase.



UML – Dijagram klasa

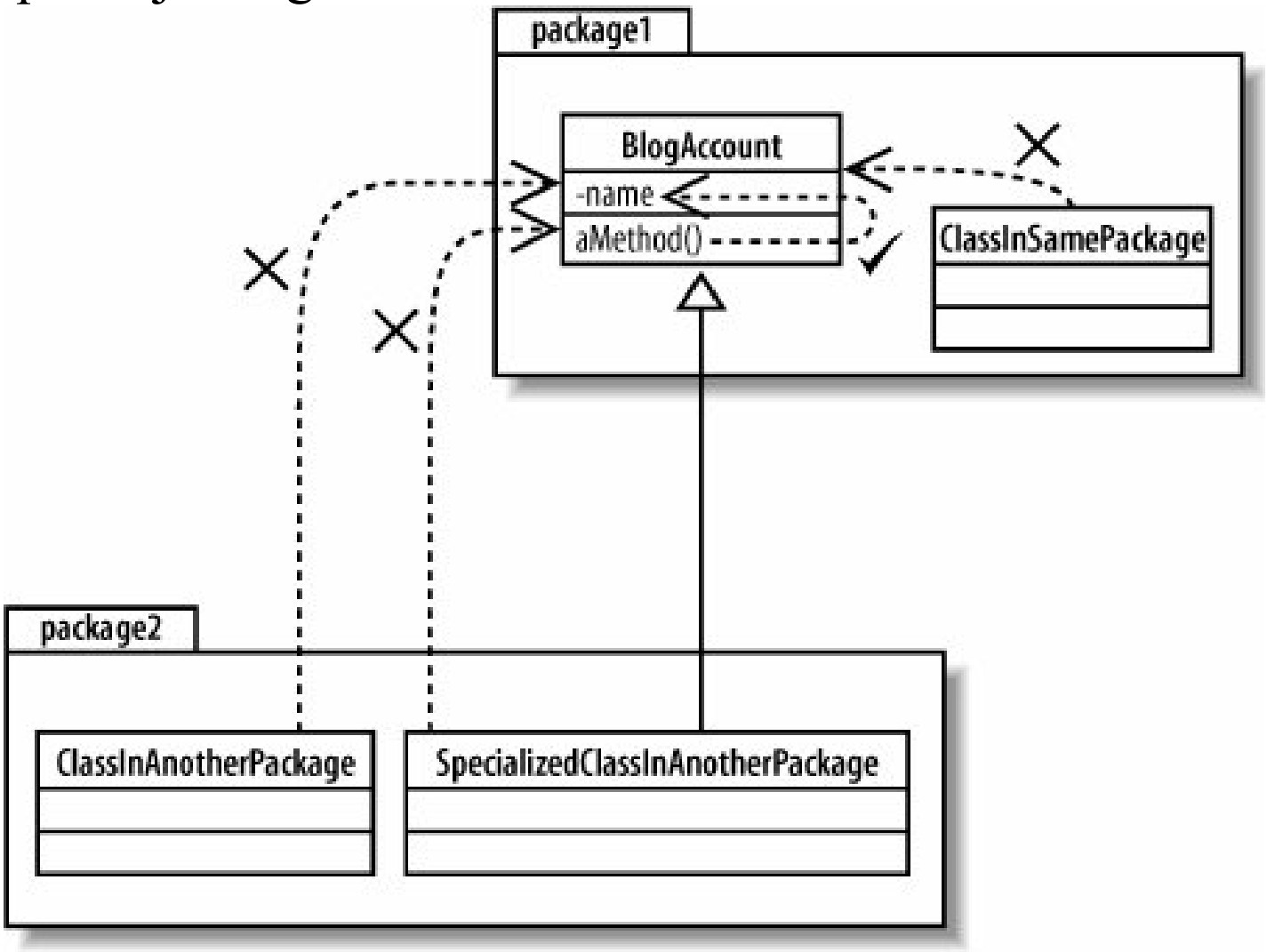
Privatna vidljivost (engl. private visibility)

Poslednja u nizu UML skale vidljivosti je privatna vidljivost.
Ova vidljivost je najkrući ograničen tip vidljivosti i prikazuje se dodavanjem znaka minus (-) ispred atributa ili operacije. **Samo klasa koja sadrži *private* element može da vidi ili da radi sa podatkom smeštenim u privatnom atributu, ili da pozove privatnu operaciju**. Privatna vidljivost je najkorisnija ukoliko imate atribut ili operaciju od koje ne želimo da zavisi ostatak sistema. To može biti slučaj kada nameravamo kasnije da promenimo neki atribut ili operaciju, ali ne želimo drugim klasama da dozvolimo promenu tog elementa.

Prihvaćeno je pravilo da bi atributi uvek trebali da budu *private*, i samo u ekstremnim slučajevima otvoreni za direktan pristup korišćenjem nešto veće vidljivosti. Izuzetak od ovog pravila je kada treba da delimo atribute naše klase sa klasama nasleđenim iz nje. U tom slučaju, uobičajeno se koristi *protected*.

UML – Dijagram klasa

Operacija *aMethod* je deo klase *BlogAccount*, tako da ova operacija može pristupiti privatnom atributu *name* i nijedna operacija druge klase ne može videti atribut *name*.



UML – Dijagram klasa – Oblast vidljivosti i vidljivost

Java ima pakete. Omogućava da elemenat bude vidljiv samo unutar istog prostora imena (*namespace*).
C# ne daje pravu podršku vidljivosto *namespace* kao u Java, ali koristi *namespace* za organizovanje previše klasa obezbeđujući nivoe razdvajanja sors koda. Mogu se smatrati kontejnerom koji se sastoji od drugih *namespace*, klasa itd.
Namespace može imati sledeće tipove kao svoje članove: *namespace* (ugnežđeni *namespace*), Klase, Interfejse, Strukture i Delegate.
Oblast vidljivosti i vidljivost u C#
Oblast vidljivosti definiše deo aplikacije gde je promenljiva dostupna, nazvan opseg promenljive. Promenljive se mogu definisati u klasama, metodama, petljama i strukturama.
Postoje tri glavna opsega za promenljivu:

- Nivo klase
- Nivo metode
- Nivo bloka (ugnežđeni opseg)

UML – Dijagram klasa – Oblast vidljivosti i vidljivost

Pet tipova vidljivosti u C#:

1. Public
2. Protected
3. Internal
4. Protected internal
5. Private

Public

Protected

Članovima se može pristupiti samo iz iste klase ili unutar klase.

Internal

Članovima se može pristupiti samo iz istog projekta.

Protected Internal

Članovima se može pristupiti samo iz istog projekta, ali klasi mogu pristupiti i one klase koje nasleđuju ovu klasu, čak i iz drugog projekta.

Private

Članovi navedeni na ovaj način mogu biti dostupni samo drugim članovima iste klase. Podrazumevano, klase i strukture su podešene na ovaj nivo vidljivosti, a ovo je najrestriktivnije.

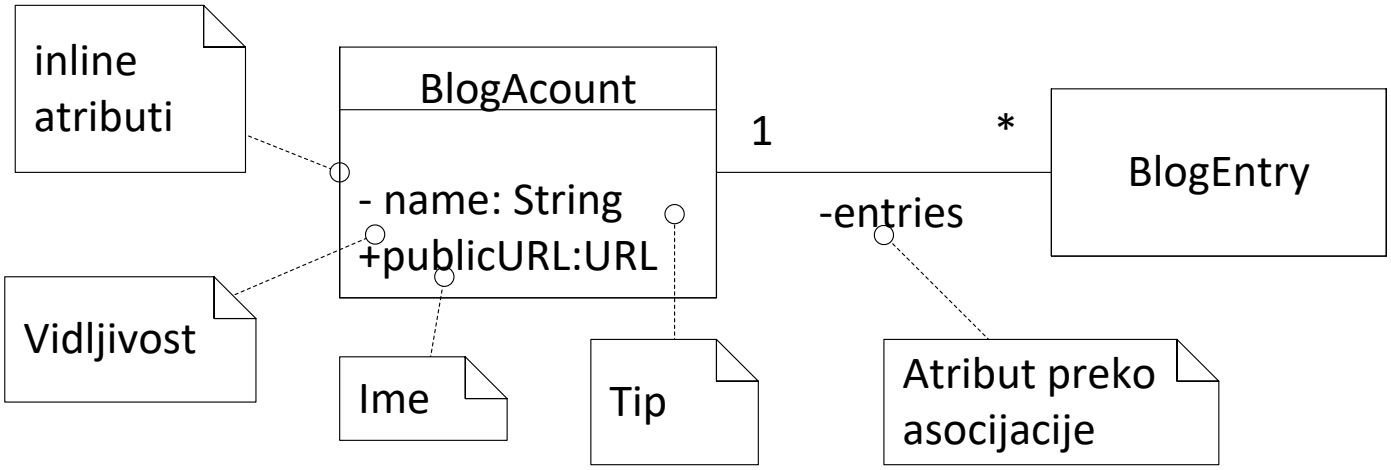
UML – Dijagram klasa

Stanje klase: Atributi

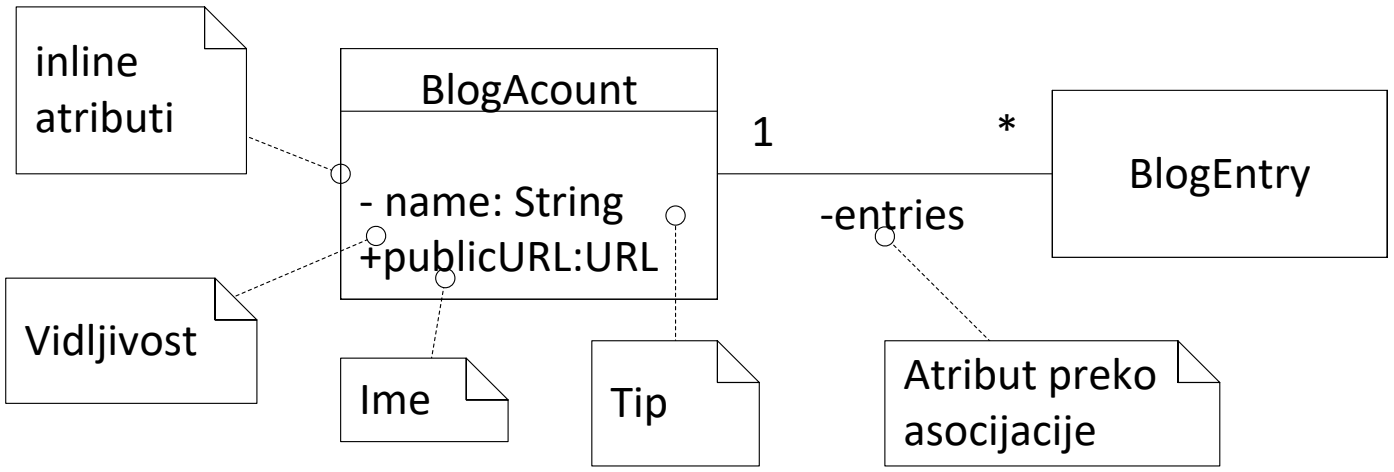
Atributi klase su delovi informacija koji predstavljaju stanje objekta.

Klasa *BlogAcount* sadrži dva *inline* atributa (*name* i *publicURL*), i jedan atribut koji predstavlja asocijaciju *BlogAccount* i *BlogEntry* klasu.

Nebitno je da li smo deklarisali atribut sa *inline* ili preko asocijacije jer ako pogledamo kod za ovu klasu videćemo suštinu stvari. Međutim, zbog preglednosti dijagrama preporučuje se pisanje *inline* atributa.



UML – Dijagram klasa




```
public class BlogAccount
{
// Dva inline atributa
   private String name;
   public URL  publicURL;
// Jedan atribut preko asocijacije, čije je ime 'entries'
// Ovaj atribut predstavlja niz klasa BlogEntries-a
   BlogEntry[] entries;
// ...
}
```

UML – Dijagram klasa

Kardinalnost definiše koliko se pojavljivanja jedne klase može pridružiti jednom pojavljivanju druge klase u posmatranoj asocijaciji.
Kardinalnost je obično interval koji definiše najmanji, odnosno najveći broj pojavljivanja (ili sve moguće brojeve pojavljivanja) koji se mogu pridružiti jednom pojavljivanju u posmatranoj asocijaciji.

Kardinalnost može biti “1” (tačno 1), “1..*”, jedan ili više, “3..5”, (tri do pet, uključujući tri i pet), “2,3,18” (dva, tri ili 18). Postoje i grafičke oznake za određenu vrstu multipliciteta. Sama zvezdica (*) označava nula, jedan ili više.

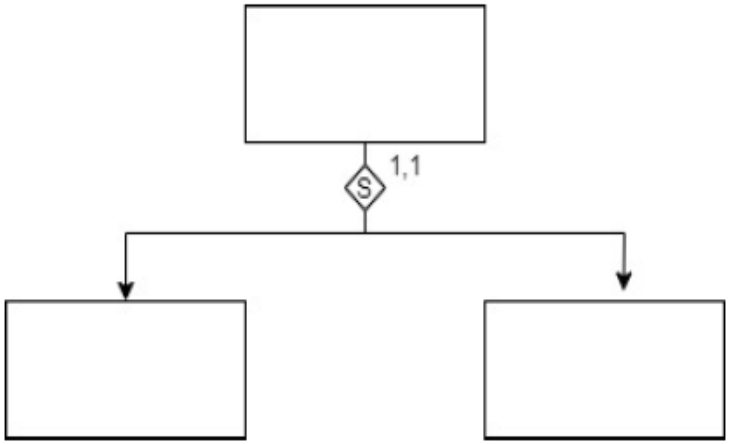
UML – Dijagram klasa

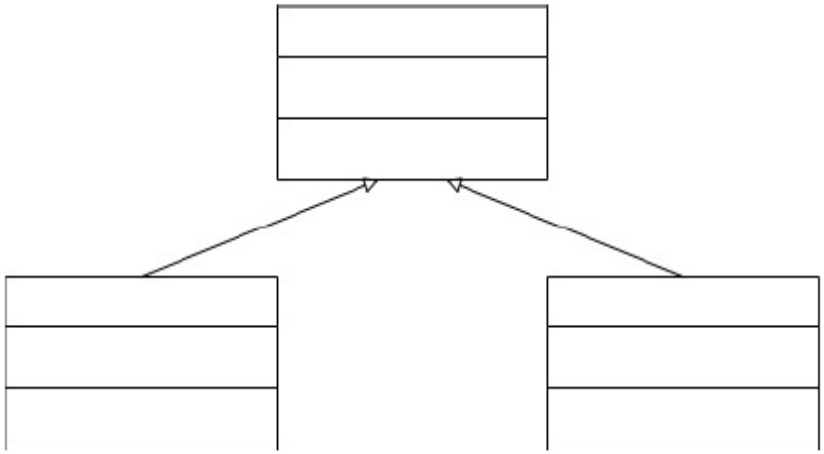
Kardinalnosti

PMOV	CLASS DIAGRAM
1,1	1
0,1	0..1
0,M	0..* (ili *)
1,M	1..*
2,8	2..8

UML – Dijagram klasa

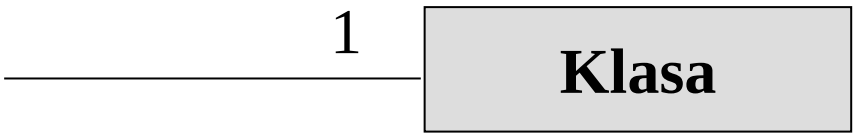
Kardinalnosti



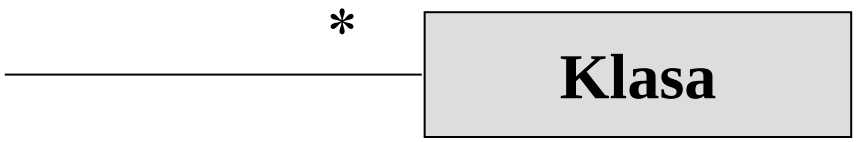


UML – Dijagram klasa

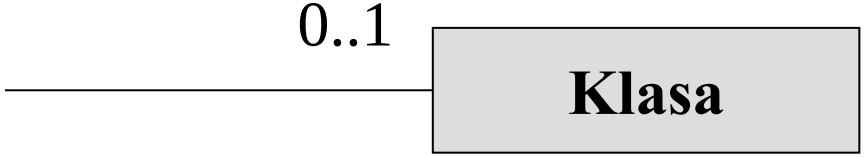
Kardinalnosti



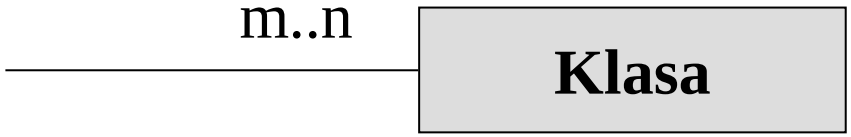
tačno jedan



više
(nula ili više)

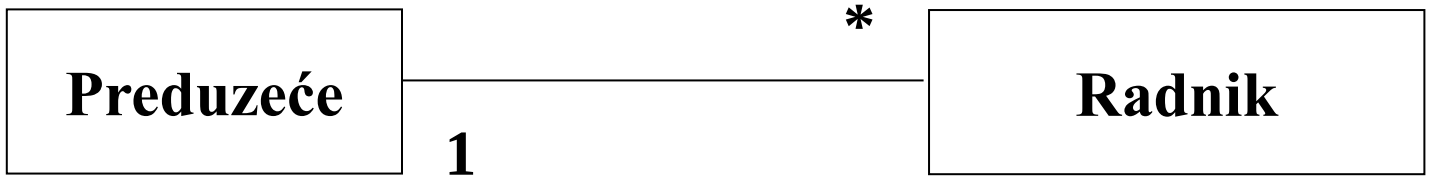


opciono
(nula ili jedan)



tačno definisan
brojni opseg

Primer:

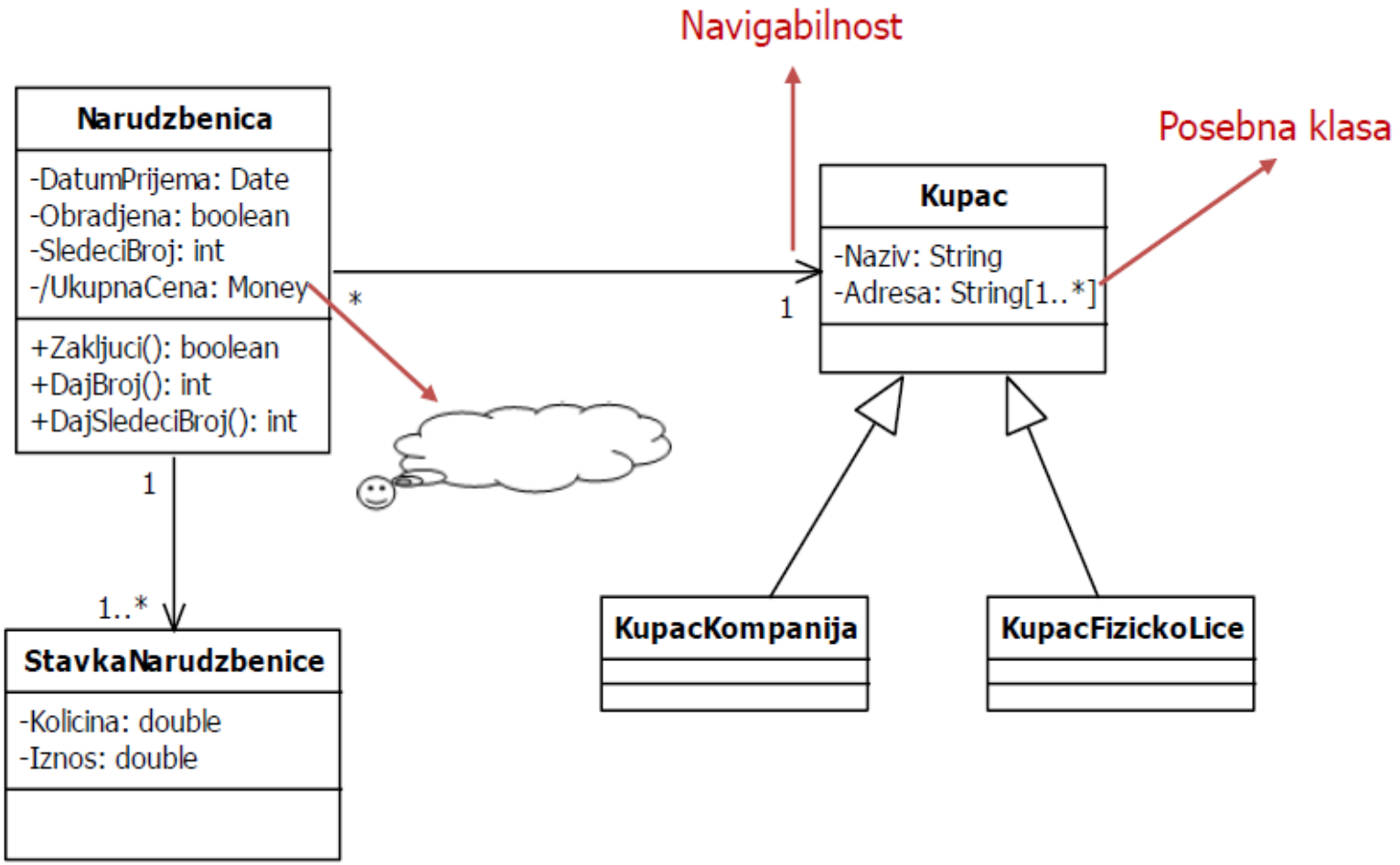


UML – Dijagram klasa

Kardinalnost	Opis kardinalnosti
Jedna instanca (Exacly One) _____	Ova veza definište tačno jednu kardinalnost. Na primer, svaka zemlja ima samo jedan glavni grad.
Više (Many) * _____	Ova veza definiše "više" kardinalnosti. Na primer, jedno odeljenje može da angažuje više osoba.
Opciono (Optional) 0..1	Ova veza definiše žopcionuž kardinalnost. Na primer, jedna osoba može da bude neras-poređena (0) ili raspoređena na jedno radno mesto.
Jedna ili više (One or more) 1..* _____	Ova veza definiše "jednu ili više" kardinalnosti. Na primer, dokument može da sadrži jedan paragraf ili više paragrafa (ali mora da sadrži najmanje jedan).
Nula ili više (Zero or one) 0..* _____	Ova veza definiše "nula ili više" kardinalnosti. Na primer, osoba može da ima nijednu ili više isplata.

Prikaz tipova kardinalnosti

UML – Dijagram klasa: primer



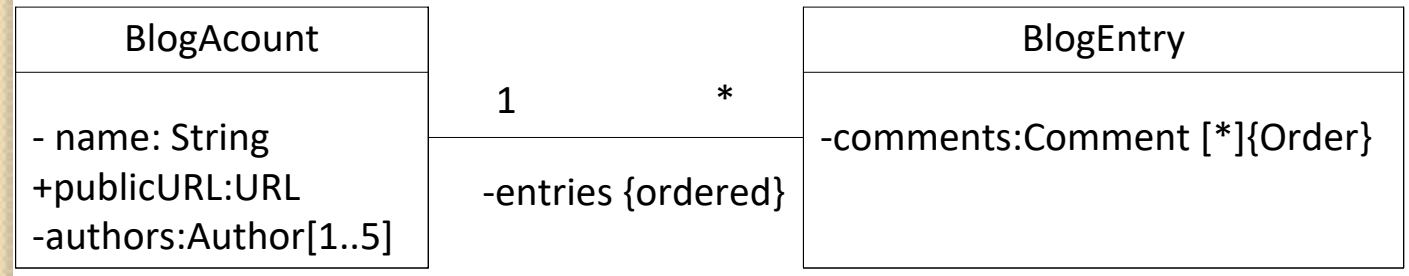
UML – Dijagram klasa

Linija bez ikakvog simbola za kardinalnost označava vezu “jedan prema jedan”, iako bi po pravilu trebalo staviti 1 sa obe strane.

Kardinalnost se upisuje uz klasu na koju se odnosi, odnosno uz klasu čija se pojavljivanja pridružuju jednom pojavljivanju druge klase.

UML – Dijagram klasa

Atributi *comments* i *authors* predstavljaju kolekciju objekata, *comments* predstavlja niz više klasa tipa *Comment*.
Zvezdica na kraju atributa *comments* specificira da može da sadrži više objekata *Comment* klase.
Atribut *authors* je ograničen sa skupom između 1 i 5. Atribut *entries* predstavlja asocijaciju gde * specificira da se bilo koji broj *BlogEntry* objekata može smestiti u atribut *entries* klase *BlogAccount*.
S druge strane 1 označava da jedan *BlogEntry* objekat u *entries* atributu može da bude u jednom i samo jednom *BlogAccount* objekatu.
Osobina ***ordered*** govori da je svaki *Comment* objekat smešten u nekom redosledu.
Osobina ***unique*** u slučaju *authors* atributa, predstavlja niz od pet objekata klase *Author*, govori da *BlogAccount* može da ima najviše pet različitih autora tj. ne sme da postoji dva ista od pet autora koji rade na blogu.



UML – Dijagram klasa

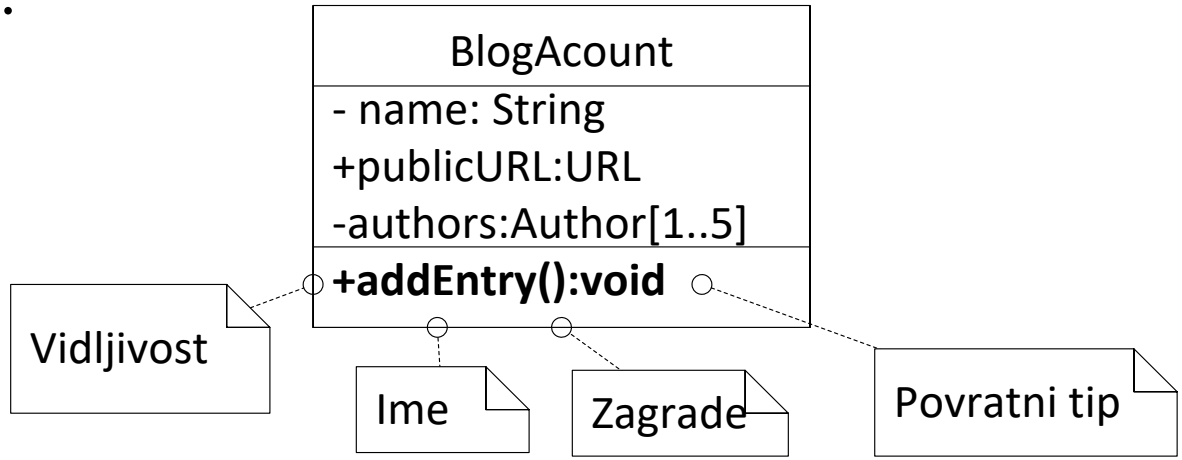
Ponašanje klase: Operacije

Operacije opisuju šta klasa može da uradi ali ne i kako.

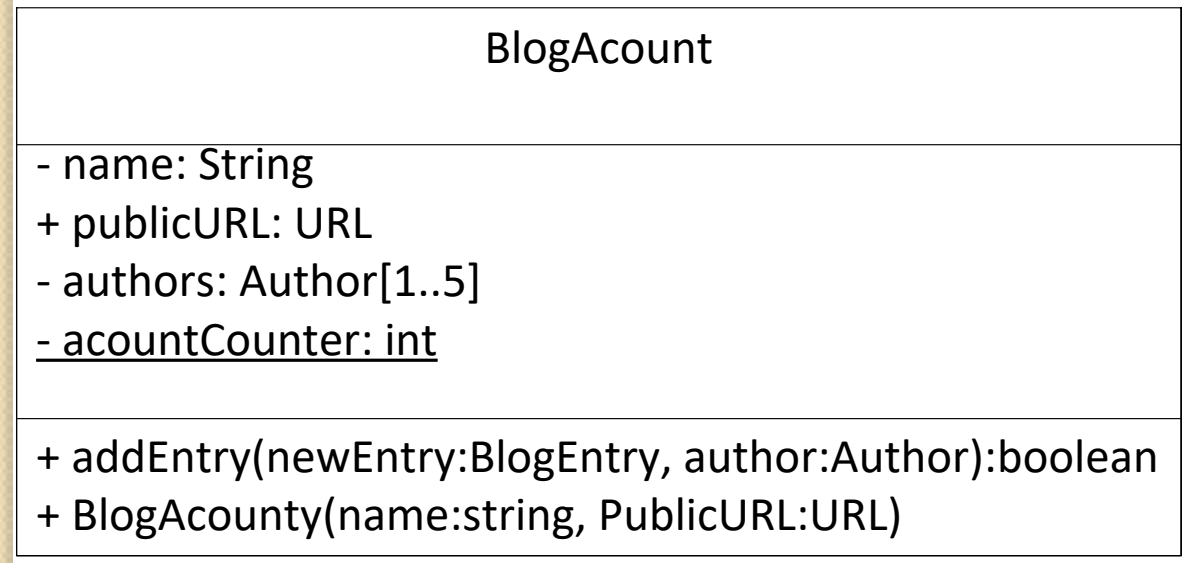
Operacija je kao obećanje ili minimalni ugovor koji deklariše da će klasa da sadrži ponašanje koje operacija kaže da će da uradi.

Operacija najmanje treba da ima vidljivost, ime, parametre u zagradama koji su neophodni za njegovo izvršenje i tip povratne vrednosti.

Operacija *addEntry* je javna, ne zahteva ni jedan parametar koji treba da se prosledi kad je neko pozove i ne vraća nikakavu vrednost (*void*).



UML – Dijagram klasa



Ovde se prosleđujuju parametri *newEntry* i *author* operaciji *addEntry* koja kao rezultat vraća povratnu vrednost tipa *boolean*. Postoji **jedan izuzetak kad se ne mora specificirati povratni tip**, a to je kad je u pitanju deklarisanje **konstruktora klase**. Konstruktor klase kreira i vraća novu instancu klase koja se specificira u tom konstruktoru i zato se ne deklariše povratna vrednost. Operacija *BlogAccount* je pravi primer za to.

UML – Dijagram klasa

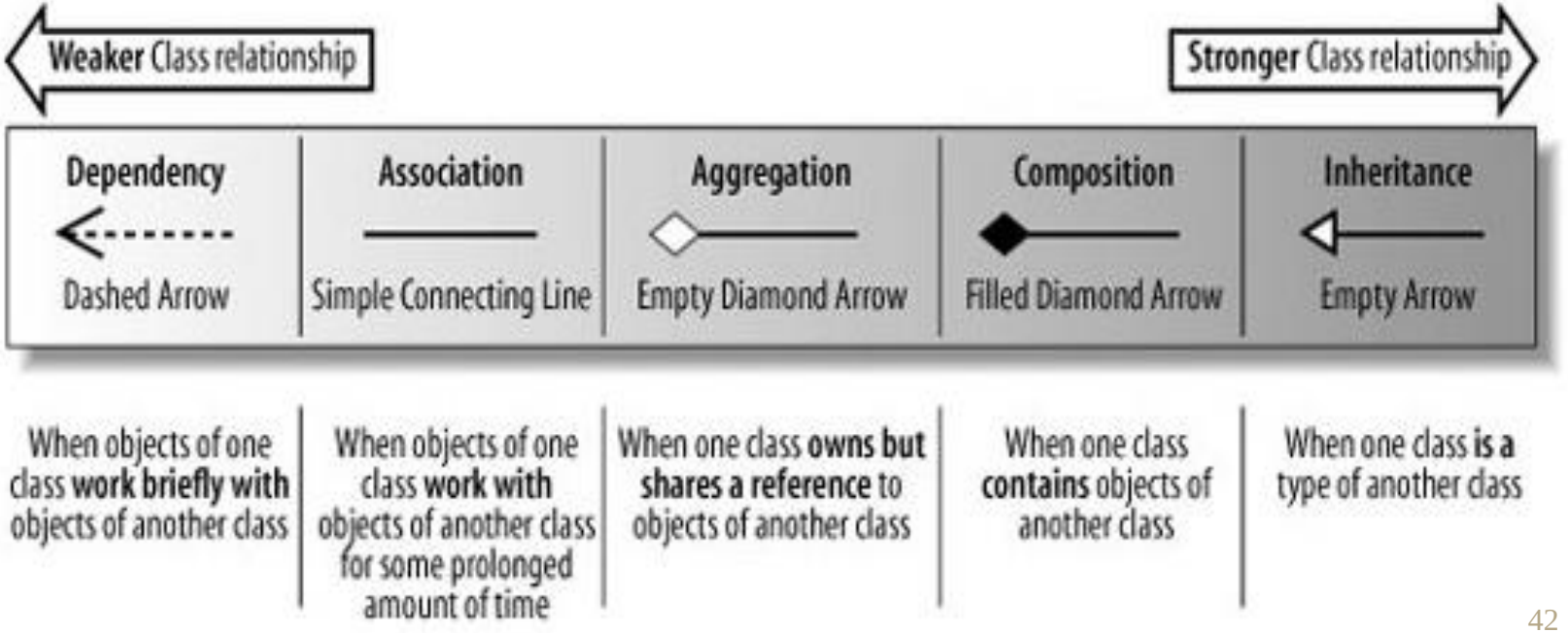
BlogAcount
- name: String + publicURL: URL - authors: Author[1..5] - acountCounter: int
+ addEntry(newEntry:BlogEntry, author:Author):boolean + BlogAcounty(name:string, PublicURL:URL)

Do sada je svaki objekat (instanca, pojavljivanje) *BlogAccount* klase dobijao svoju kopiju atributa i operacija. Ako želimo da određena klasa deli istu kopiju atributa i operacija onda nam je potrebna osobina **static**. Na primer, ako želimo da čuvamo broj objekata *BlogAccount*-a u sistemu, onda je ovaj brojač, nazvan *accountCounter,* dobar kandidat da bude **statični atribut** ove klase. Tako da svaki put kad se instancira (kreira) klasa *BlogAccount* ovaj se brojač povećava za jedan. Isto tako može da se smanji za jedan. Statički atribut se podvlači donjom crtom. Dakle, *static* osobina znači da se kreira za sve samo jedna kopija sa kojom rade svi ostali.

UML – Dijagram klasa

Veze (relacije) između klasa

Odnosi između klasa su različite jačine. **Jačina odnosa između klasa se zasniva na tome koliko su klase zavisne jedna od druge**. Za dve klase koje su snažno zavisne jedna od druge kaže se da su usko povezane. Promene u jednoj klasi će skoro uvek uticati i na drugu klasu. Povezanost je obično, ali ne i uvek loša stvar. I sve što je jači odnos moramo biti pažljiviji.



UML – Dijagram klasa

Zavisnost (*Dependency*)
Zavisnost između dve klase znači da jedna klasa mora znati za drugu klasu da bi koristila objekte te klase. U dijagramu klasa, dve klase objekata su zavisne jedna od druge, da bi se osigurale da rade zajedno.

Zavisnost govori da samo objekti klase mogu raditi zajedno, međutim to se smatra najslabijim direktnim odnosom koji može postojati između dve klase.

Zavisnost se često koristi kada imamo klasu koja obezbeđuje korisne operacije, kao što su npr. Javini opšti izrazi (java.util.regex) i matematički paketi (java.math). Klase zavise od java.util.regex i java.math klasa da bi koristile usluge koje te klase nude.

UML – Dijagram klasa

Asocijacija (*Association*)

Pod pojmom **Asocijacija** (link) podrazumeva se fizička ili konceptualna veza pojavljivanja objekata. Npr. *Milan Radi u Prodaji*.

Veza se formalno definiše kao n-torka, odnosno uređena lista pojavljivanja.

Asocijacija predstavlja grupu veza slične strukture i jedinstvene semantike.

Veze su pojavljivanja asocijacije. Sve veze u jednoj asocijaciji povezuje objekte iz istih klasa.

Asocijacija opisuje skup potencijalnih veza, na isti način kao što klasa predstavlja skup potencijalnih objekata.

UML – Dijagram klasa

Asocijacija (*Association*)

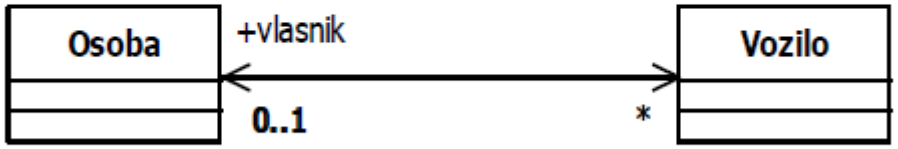
Primer asocijacije je *Radi* koja predstavlja veze objekte iz klase Radnik i klase Organizacija.

Asocijacije predstavljaju dvosmerne veze. Po pravilu, asocijaciji se daje samo jedno ime koje izražava vezu u jednom smeru, inverzno ime se podrazumeva, a mogu se koristiti oba smera asocijacije.

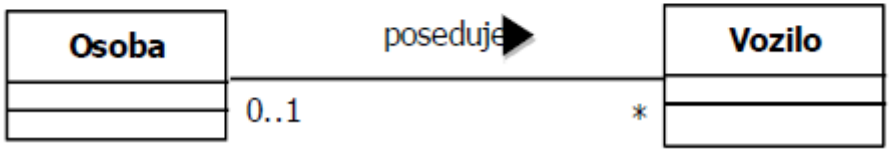
Ne moraju asocijacije, odnosno veze imati naziv ako je ta asocijacija jedina asocijacija posmatranih klasa i ako je naziv asocijacije očigledan.

UML – Dijagram klasa

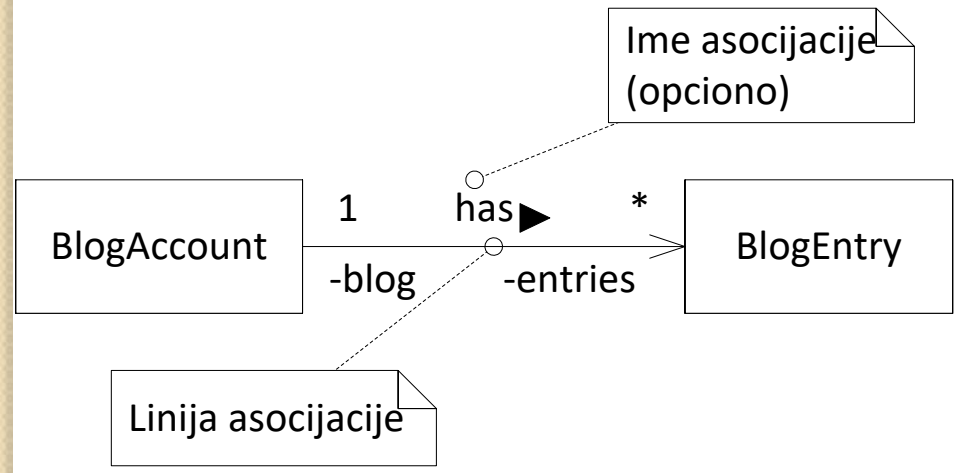
Dvosmerno referenciranje



Korišćenje naziva asocijacije



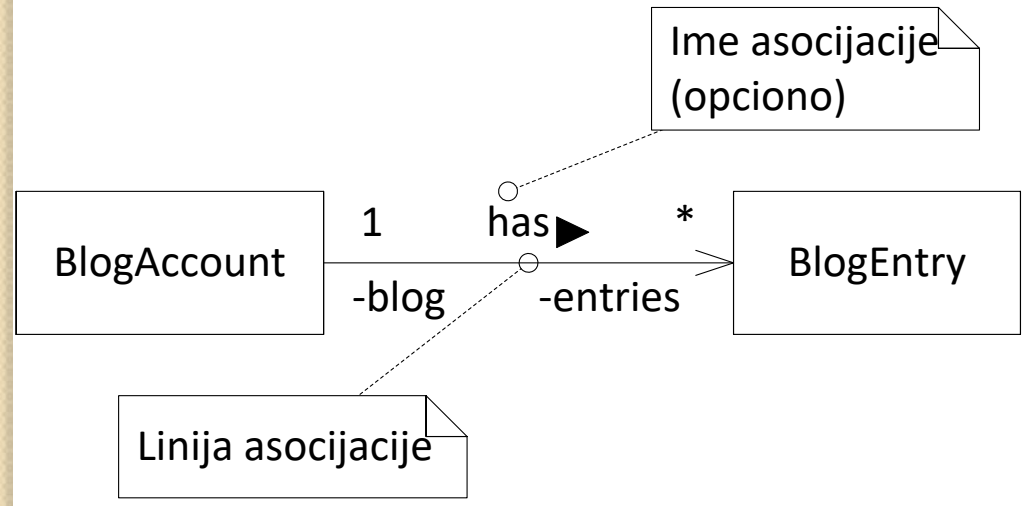
UML – Dijagram klasa



Navigabilnost (asocijacija sa strelicom) asocijacije je opciona i ona govori koja klasa sadrži atribut koji podržava ovu vezu. Bez dodatnih informacija o klasama *BlogAccount* i *BlogEntry*, nemoguće je odlučiti koja klasa treba da sadrži asocijaciju kao atribut. **U ovom slučaju samo klasi *BlogAccount* je dodat atribut**.

U ovom sistemu, logičnije je da se nalog bloga (*BlogAccount*) pita koje unose sadrži, nego da pitamo unos (*BlogEntry*) kom nalogu bloga pripada. U ovom slučaju se koristi navigabilnost da osigura da klasa *BlogAccount* dobije asocijaciju kao atribut.

UML – Dijagram klasa

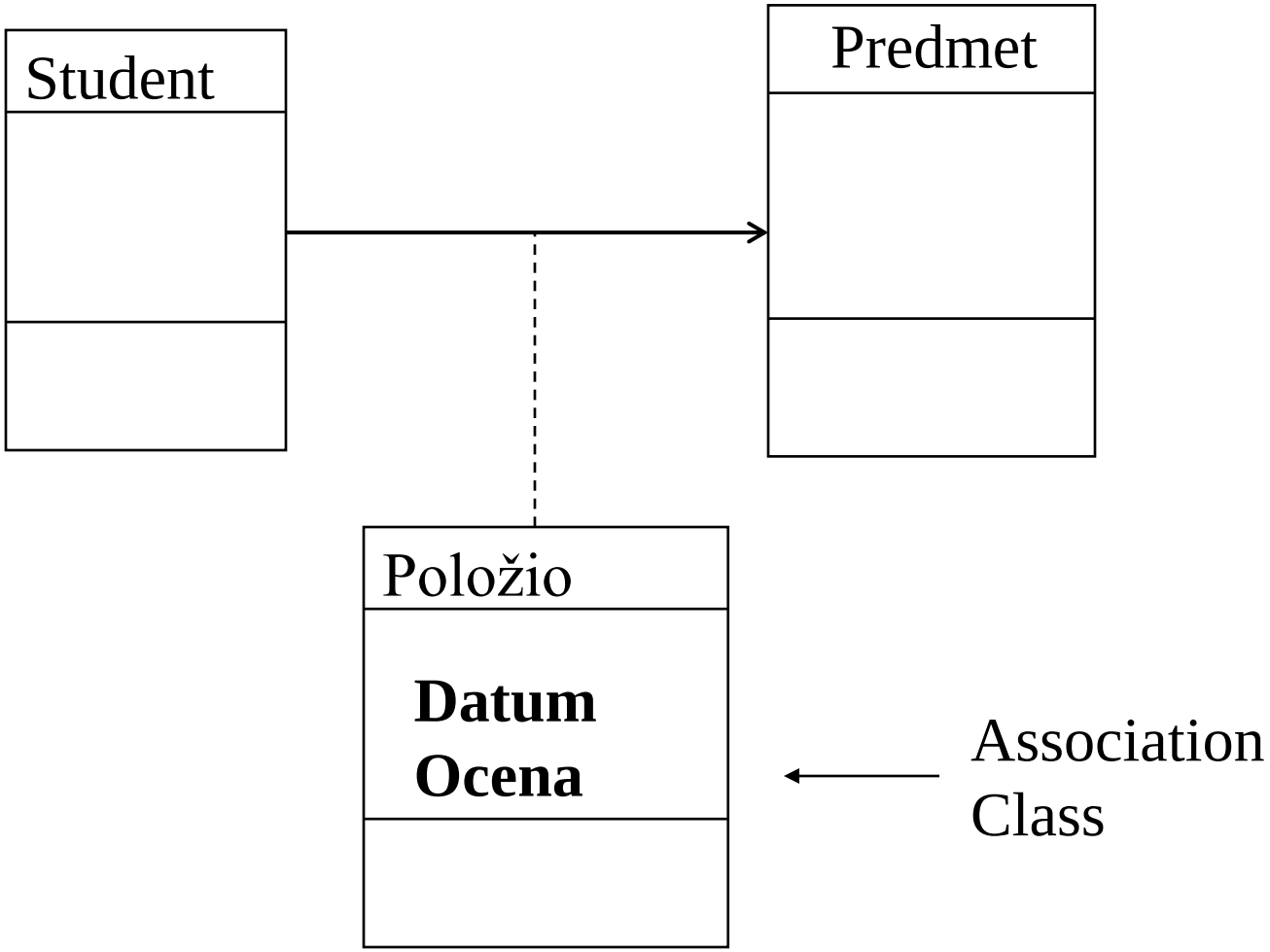


Sofverski kod za ovu vezu bi mogao da izgleda ovako:

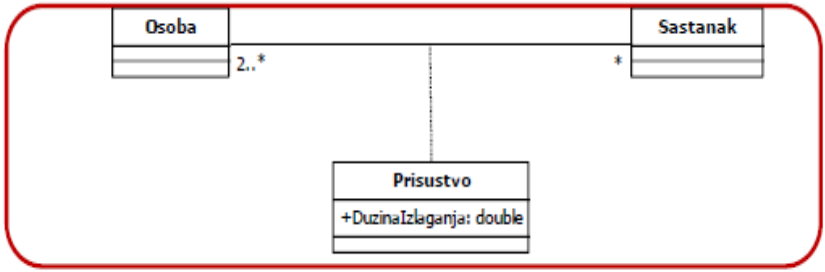
```
public class BlogAccount
{
// Ovde se nalazi atribut entries zahvaljući asocijaciji sa klasom ---
//BlogEntry
private BlogEntry[] entries;
// ... Druge metode i operacije
}
```

UML – Dijagram klasa

Asocijacija kao klasa



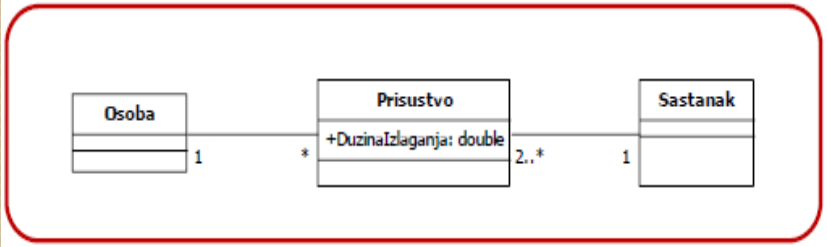
UML – Dijagram klasa - Klasa asocijacije



I način

PMOV

II način



UML – Dijagram klasa - Agregacija

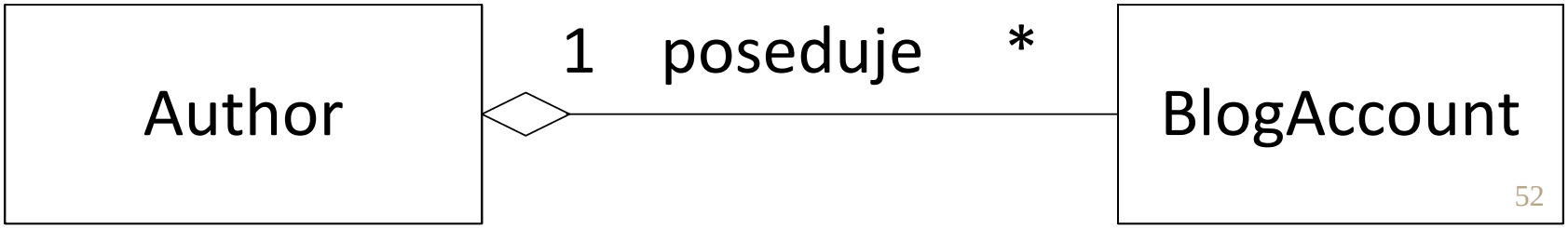
Agregacija (*Aggregation*)

Pod agregacijom se podrazumeva tzv. deo-od (*part-of*), odnosno deo-celina (*part-whole*) veza između objekata.

- Dva objekta koja se nalaze u vezi tipa agregacije imaju uloge **sklopa i dela**.
- Objekti koji učestvuju u agregaciji nazivaju **agregirajući**, a **objekat na višem nivou apstrakcije** se naziva **agregirani objekat** ili prosto agregacija.
- **U zavisnosti da li agregirajući objekat može biti deo samo jednog agregiranog objekta ili deljen između njih više**, UML razlikuje dve forme agregacije: **deljiva agregacija i nedeljiva ili kompozicija**.
- **Inverzan** postupak agregaciji je **dekompozicija**, kada se do tada posmatran jedinstven objekat dekomponuje na više drugih objekata.

UML – Dijagram klasa - Agregacija

Agregacija (*Aggregation*)
Agregacija je jača verzija asocijacije i koristi se da pokaže da klasa u **stvari *poseduje ali i deli* objekte neke druge klase**.
Agregacija se prikazuje korišćenjem **praznog romba kod klase u čijem je vlasništvu**.
Agregacijom se može pokazati da autor poseduje kolekciju blog naloga.
Veza između autora i njegovog bloga, kao što je pokazano na slici je mnogo **jača od same asocijacije**. To je zato što autor poseduje svoj blog, **čak iako bi on možda podelio sa drugim autorima, na kraju bi blog i dalje ostao njegov**. I kada bi on odlučio da ukloni taj blog on bi mogao to da uradi bez poteškoća. Softverski kod je sličan kao u asocijaciji.

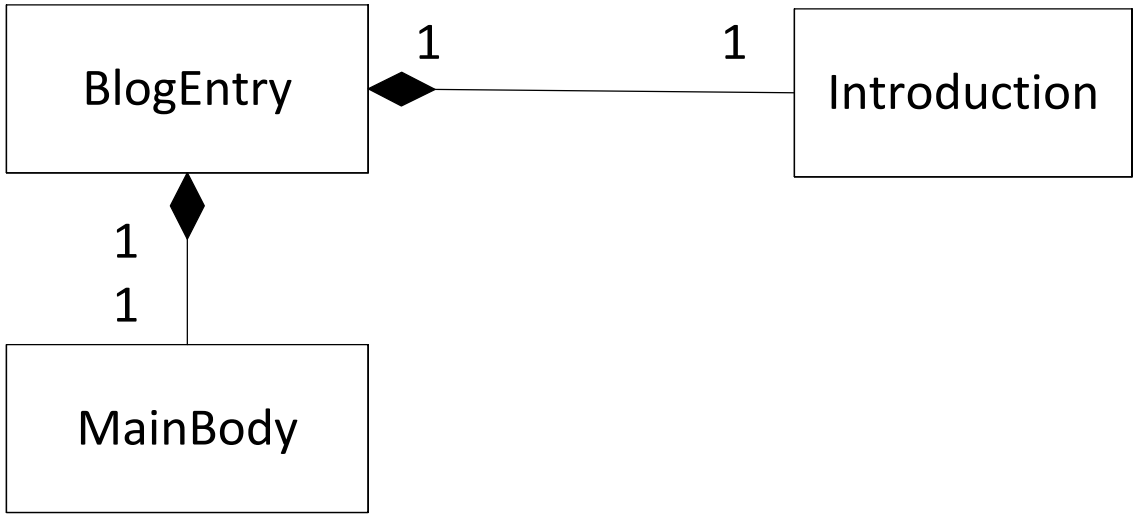


UML – Dijagram klasa - Agregacija

Kompozicija (*Composition*)

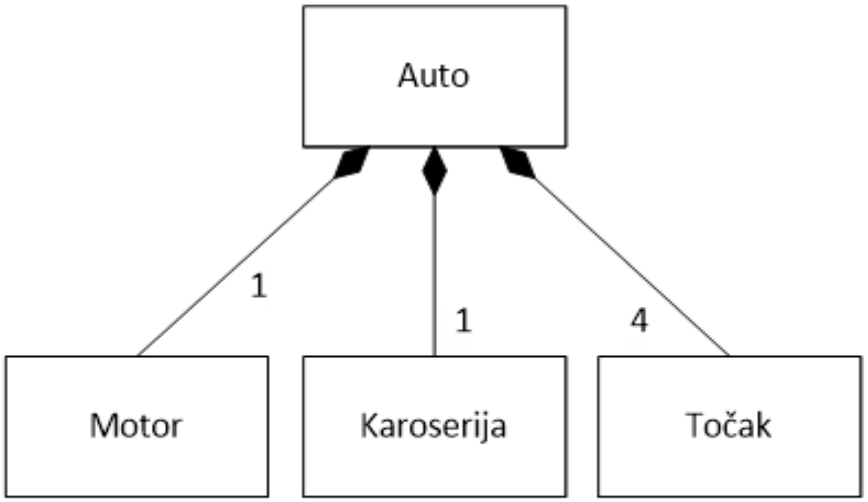
Kompozicija je još snažnija veza od agregacije, mada ona funkcioniše na veoma sličan način. Kompozicija se prikazuje crno ispunjenim rombom. Unos u blog (*BlogEntry*) je sastavljen od Uvoda (*Introduction*) i GlavnogTela (*MainBody*) i on se ne deli sa drugim *BlogEntry.*

U slučaju da je unos bloga obrisan, brišu se i njegovi delovi sa kojima sarađuju. Ovo je suština kompozicije, gde vi u stvari modelujete unutrašnje delove koji kasnije čine klasu. Kao i kod agregacije, softverski kod je sličan asocijaciji.



UML – Dijagram klasa - Agregacija

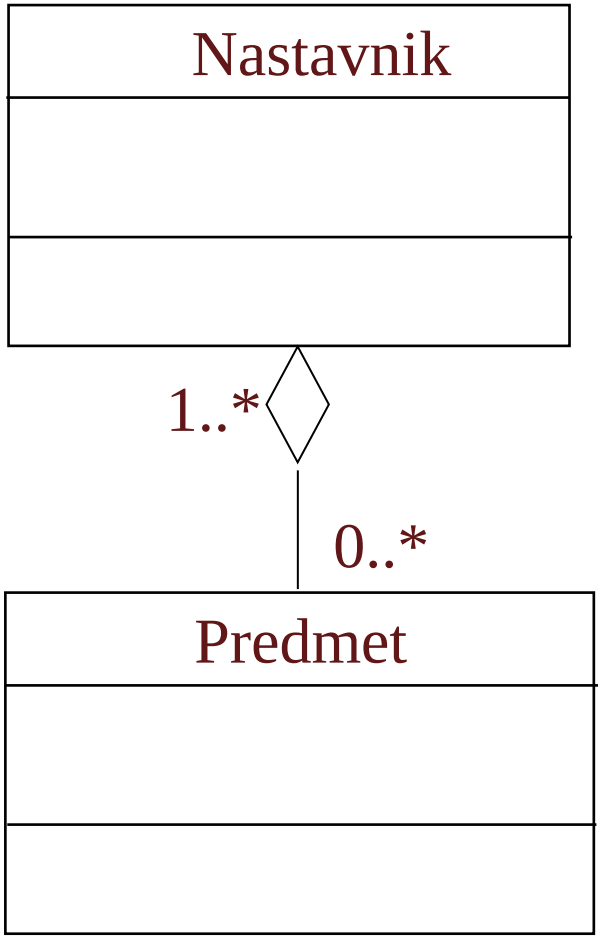
Deljiva agregacija



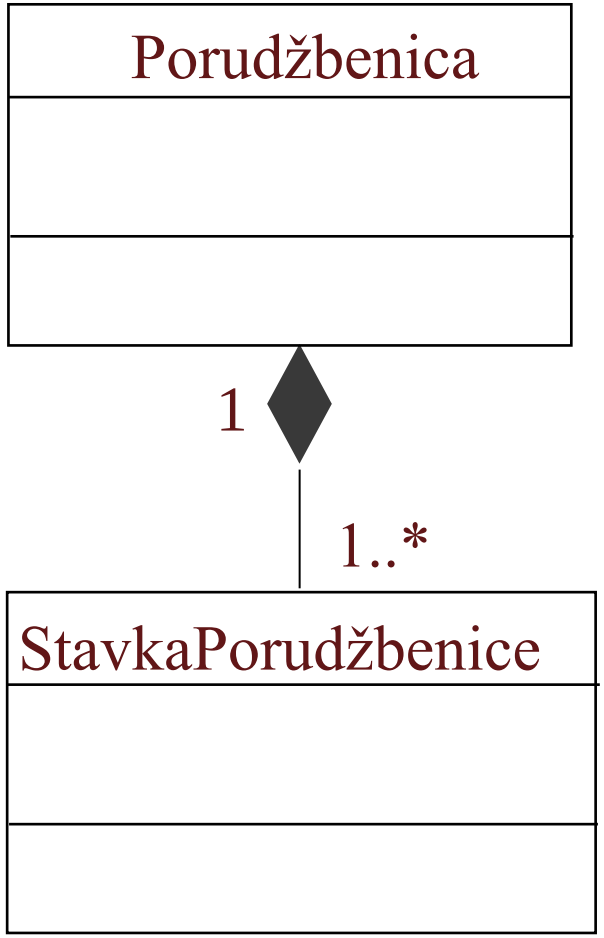
Kompozicija

UML – Dijagram klasa - Agregacija

Agregacija



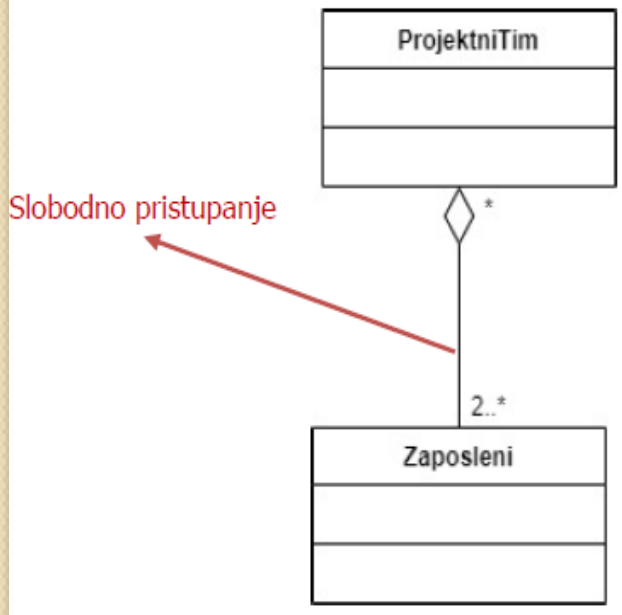
Kompozicija



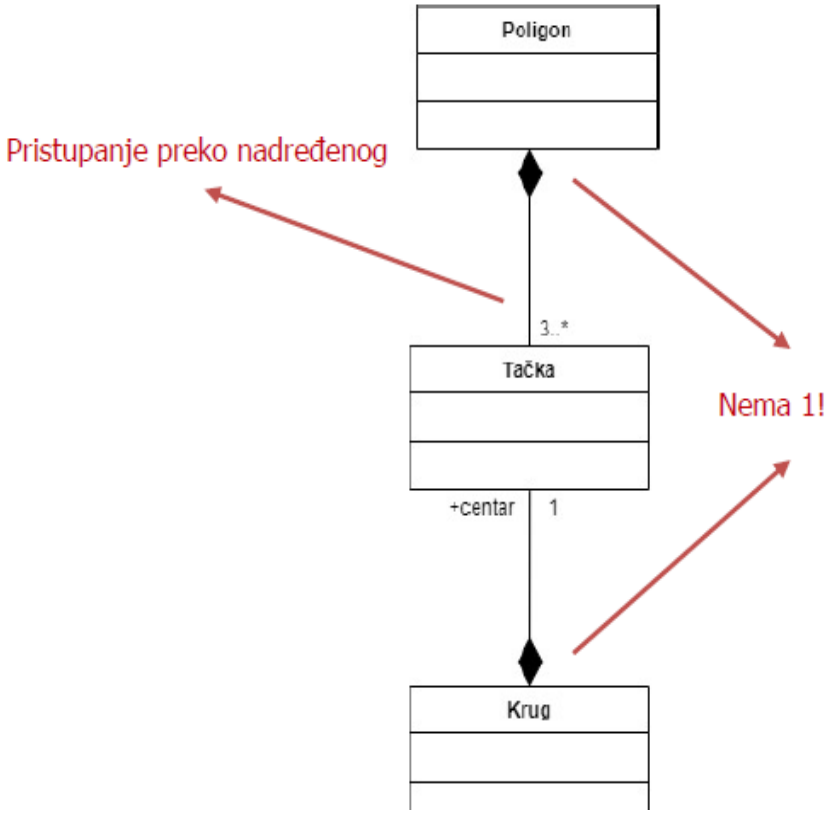
(primer: ruka --> prst)

UML – Dijagram klasa - Agregacija

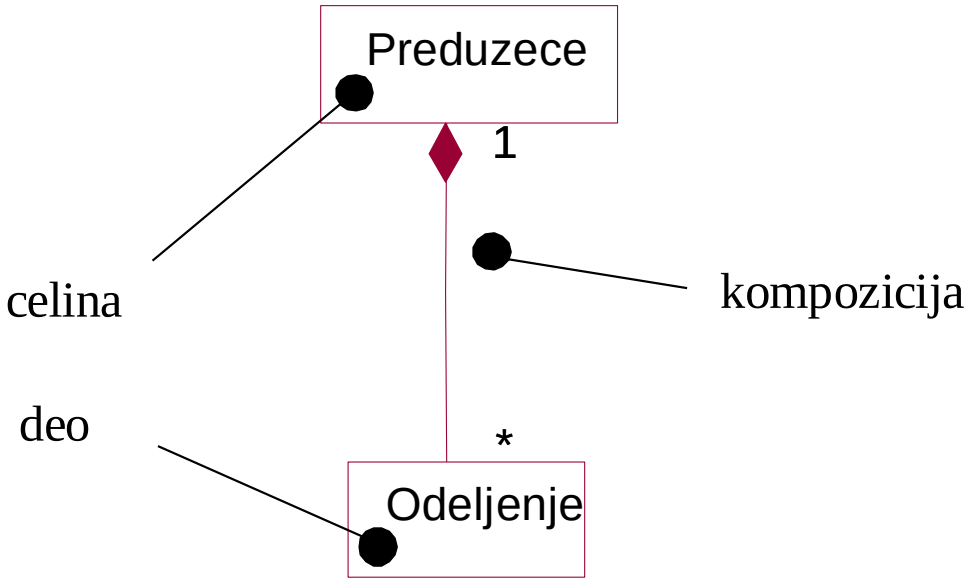
AGREGACIJA
aggreagtion (part – of relationship)



KOMPOZICIJA
composition (part – of relationship)



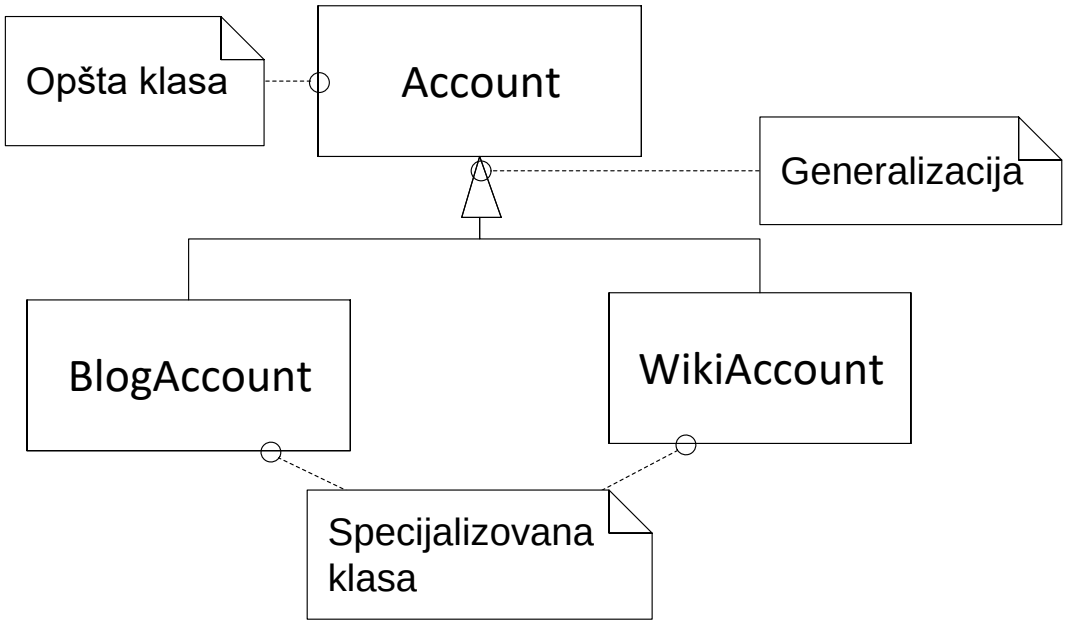
UML – Dijagram klasa - Agregacija



- Odeljenje pripada samo određenom preduzeću i nestankom preduzeća nestaje i odeljenje.

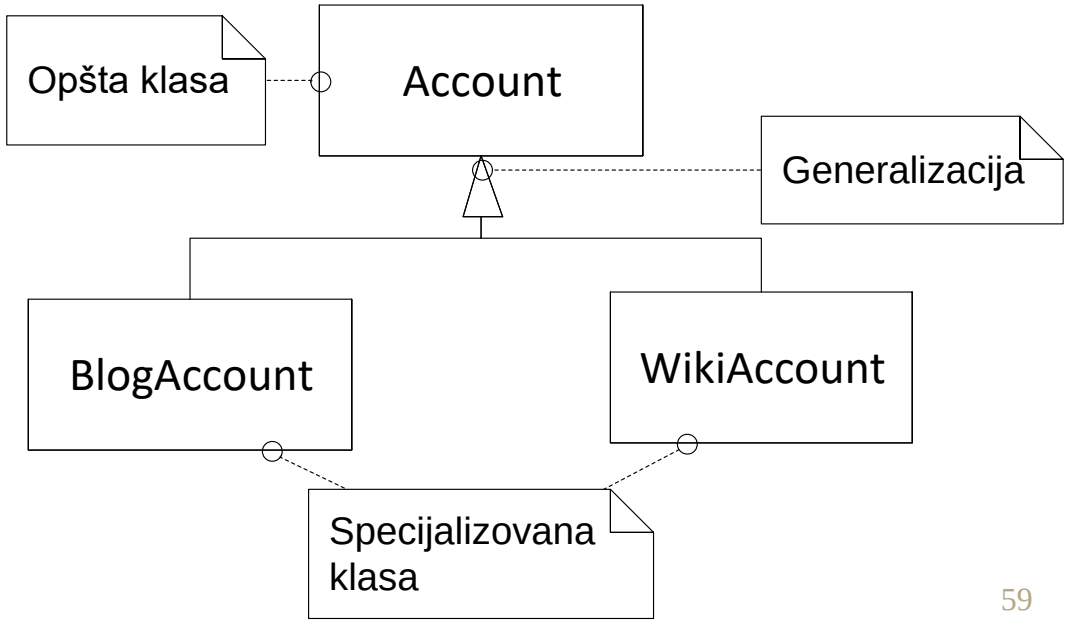
UML – Dijagram klasa

Nasleđivanje (*Inheritance*)
Nasleđivanje (generalizacija) se koristi da opiše klasu koja je tip neke druge klase. Glavni termini *ima* i *je tipa imati*, postali su prihvatljiv način odlučivanja da li je veza između dve klase agregacija ili generalizacija. Ukoliko se nađete u položaju da klasa ima deo koji je objekat druge klase, onda je veza verovatno asocijacija, kompozicija ili agregacija.
Ukoliko se nađete u poziciji da vam je jedna klasa tip neke druge klase, onda možete uzeti u obzir mogućnost da vam veza bude **generalizacija**.

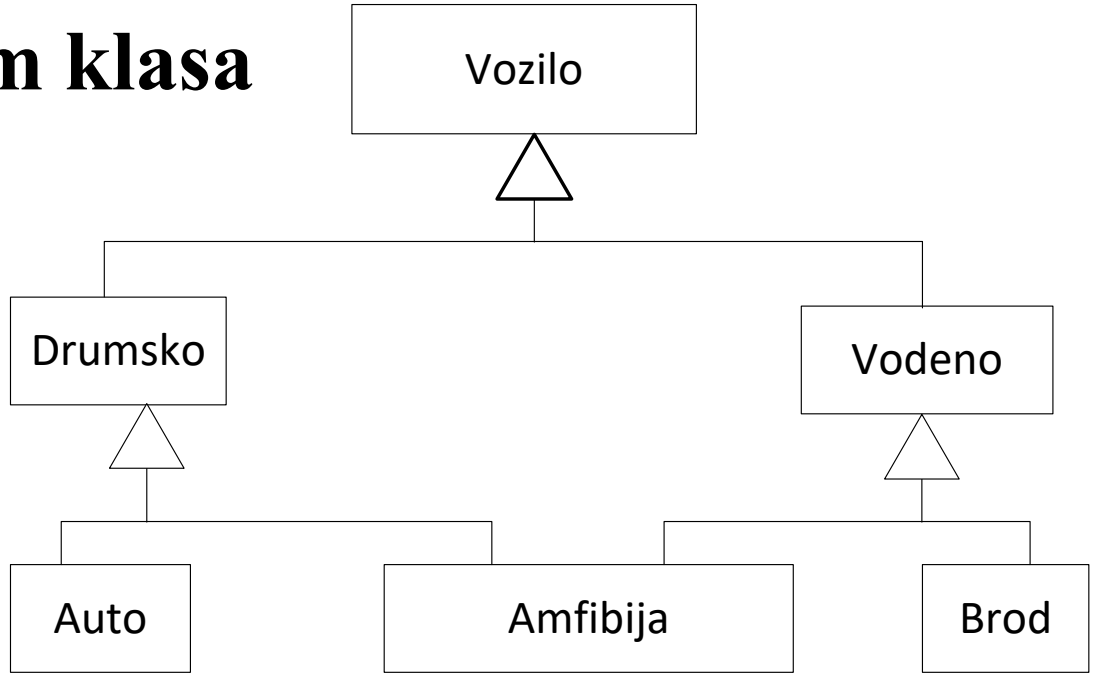


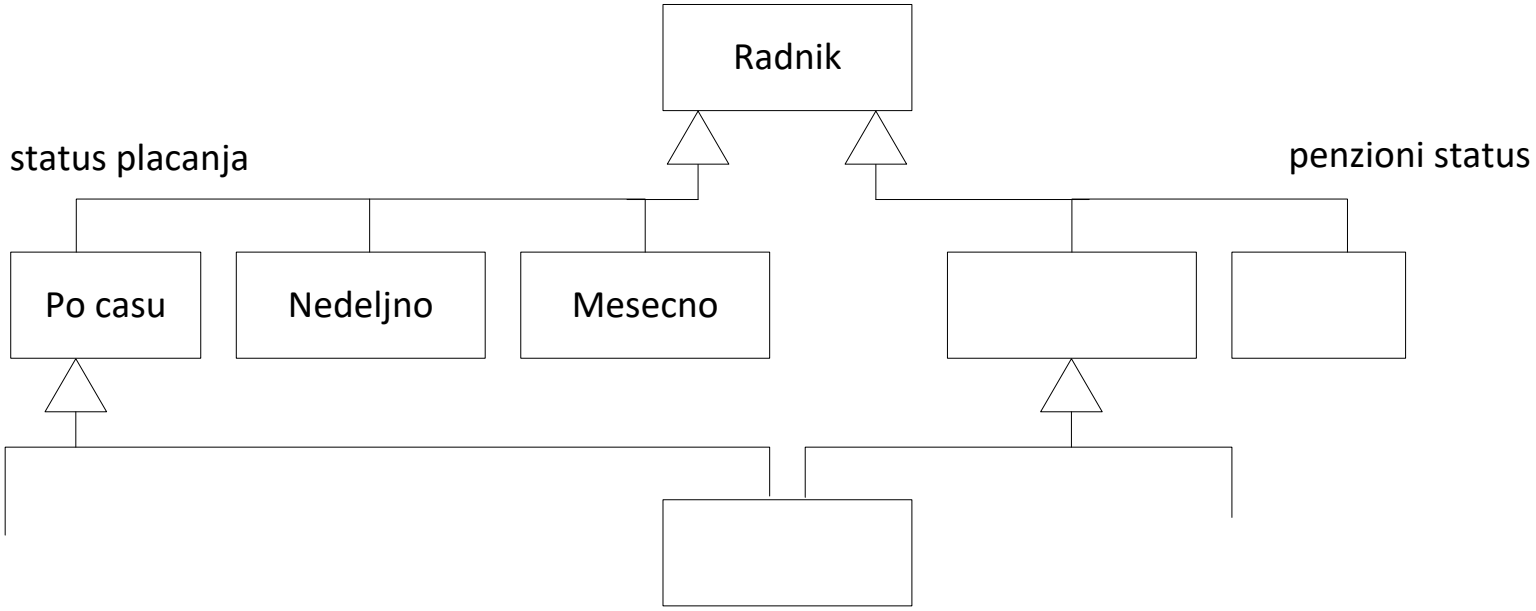
UML – Dijagram klasa

Generalizacija nudi sjajan način izražavanja da je jedna klasa tip druge klase i nudi način ponovnog korišćenja atributa i ponašanja između dve klase. Jedno od principa OO projektovanja jeste da se izbegava uska povezanost klasa, da kada se promeni jedna klasa, ne mora da se menja još dosta stvari u drugim klasama. Generalizacija je najjači oblik odnosa između klasa, zato što stvara usku povezanost između klasa. **Dobro je pravilo da se generalizacija koristi samo kada je klasa specijalizovani tip druge klase, a ne samo pogodnost da se iskoristi ponovno korišćenje**.



UML – Dijagram klasa





UML – Dijagram klasa

Postoji statički i dinamički polimorfizam.

- Statički polimorfizam: *method Overloading* dolazi kada dve ili više metoda u jednoj klasi imaju isto ime metode, ali različite parametre.
- Dinamički polimorfizam: *method Overriding* se dešava kada dve metode imaju isto ime metode i iste parameter i to je postupak kojim se menja neka nasleđena karakteristika klase u podklasi. Mogu se predefinisati difoltne vrednosti atributa ili znatno češće metode neke operacije.
- Ove mogućnost koje postoje u većini OO jezika treba oprezno koristiti, jer otežava razumevanje modela sistema.

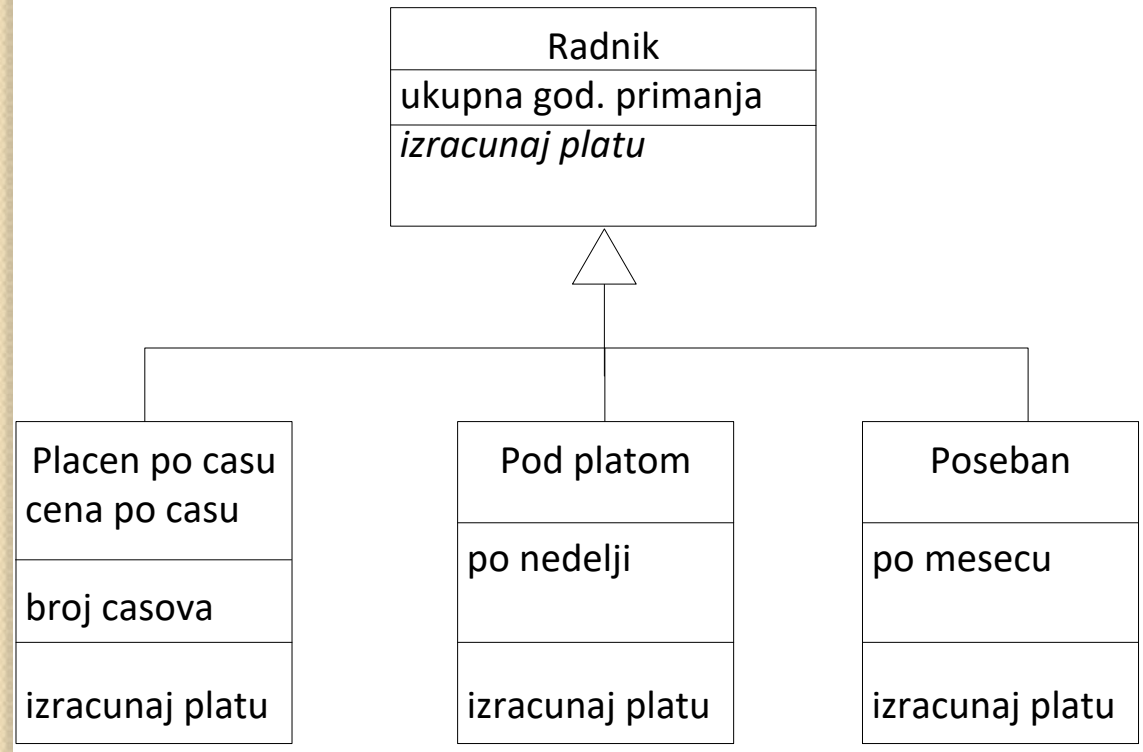
UML – Dijagram klasa

Apstraktne klase su klase koje nemaju direktnih pojavljivanja, instanciranja. Indirektna pojavljivanja apstraktne klase su pojavljivanja njenih podklasa. **Konkretne klase su one klase koje imaju direktna pojavljivanja** (koje se mogu instancirati).

Konkretna klasa može imati apstraktu podklasu a ove moraju imati konkretne potomke. **Apstraktne klase služe da pogodno prikažu zajedničke osobine više klasa. Na primer, kada više različitih klasa ima istu asocijaciju prema istoj klasi dobro je kreirati apstraktnu klasu i preko nje prikazati tu asocijaciju**.

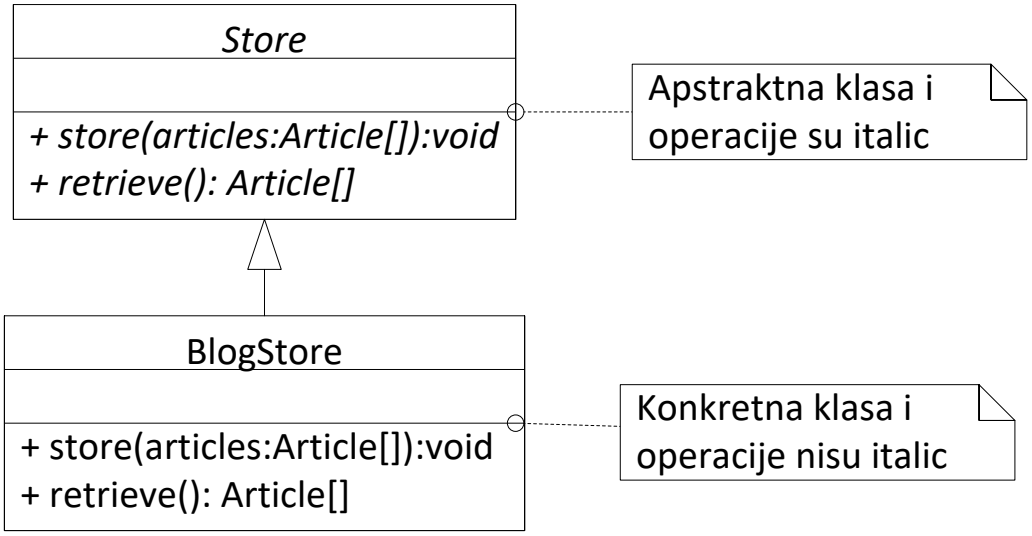
Apstraktna klasa se definiše da bi se uvela tzv. apstraktna operacija koja predstavlja operaciju u apstraktnoj klasi za koju nije definisan metod. Konkretne klase moraju imati svoju sopstvenu implementaciju te apstraktne operacije.

UML – Dijagram klasa



Apstraktna operacija ne sadrži metod ubacivanja i ona jednostavno poručuje: "Ostavljam implementaciju ovog ponašanja mojim podklasama". Ako je bilo koji deo klase proglašen za *abstract* onda i sama klasa mora biti proglašena kao *abstract*, a u UML se ovo označava italik slovima.

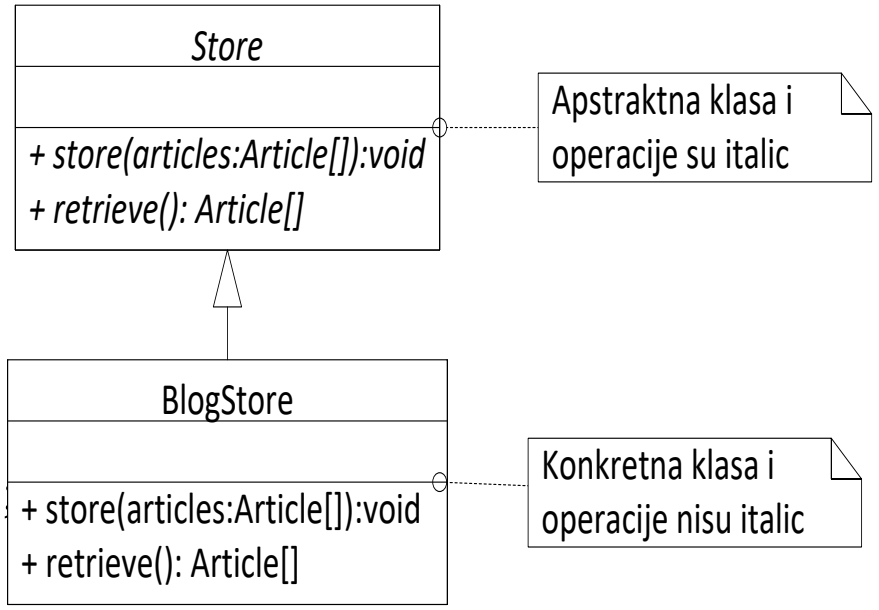
UML – Dijagram klasa



Na slici se vidi apstraktna klasa *Store* sa operacijama *store* i *retrieve* čije metode treba da znaju kako da smeste (*store*) i vrate (*retrieve*) kolekciju artikala. Sada kada su *store* i *retrieve* u *Store* klasi proglašene kao *abstract* one ne prikazuju njihovu implementaciju.

Apstraktna klasa ne može da se instancira u neki objekat niti da implementira operacije, ali klase koje je nasleđuju mogu da implementiraju njihove metode. *BlogStore* klasa nasleđuje apstraktnu klasu *Store* i implementira metode *store* i *retrieve*. Ponekad se klase koje kompletno implementiraju sve apstraktne operacije nazivaju **konkretne** klase.

UML – Dijagram klasa

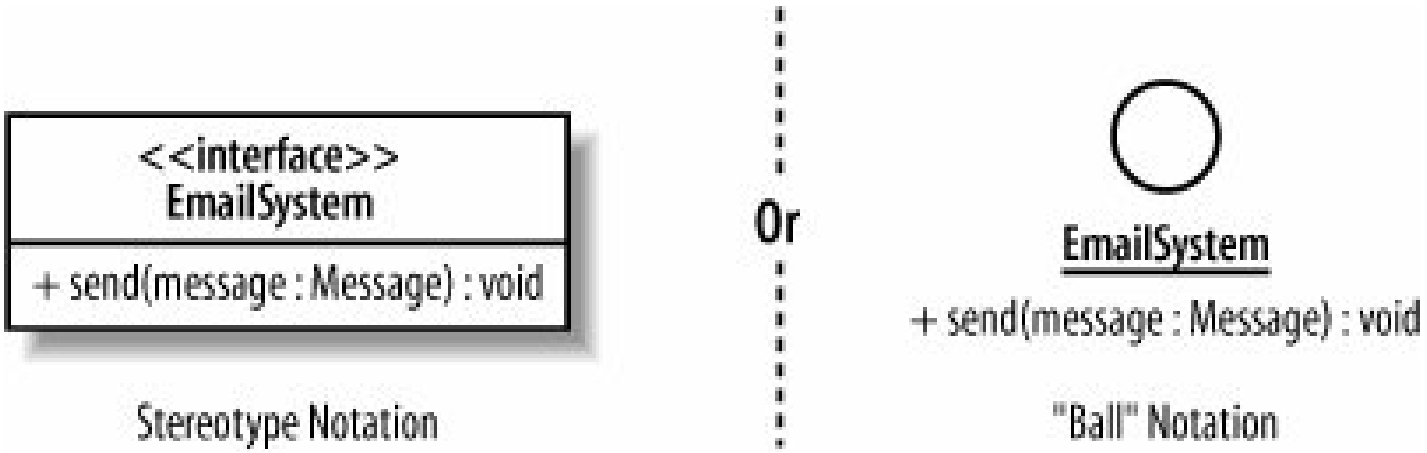



```
public abstract class Store
{
  public abstract void store(Article[] articles);
  public abstract Article[] retrieve(  );
}

public class BlogStore:Store
{
    public void store(Article[] articles)
              {
                // Ovde ide kod za smeštanje bloga...
              }
    public Article[] retrieve(  )
               {
                // Ovde ide kod za vraćanje smeštenih blogova...
              }
}
```

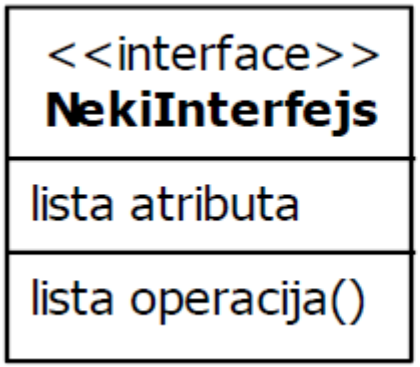
UML – Dijagram klasa

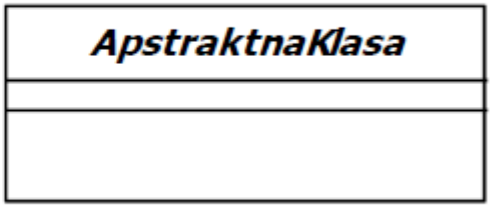
Interfejs je kolekcija operacija koje nemaju odgovarajuće implementacije metoda i koje specificiraju servis klase ili komponente, koji je **veoma sličan apstraktnoj klasi koja sadrži samo apstraktne metode**. **Interfejs opisuje ponašanje elementa koje je spolja vidljivo.**
Interfejs definiše skup specifikacija (prototipova) operacija, ali ne i njihove implementacije. **Klasa i komponenta mogu da implementiraju više interfejsa**.
Ponekad i interfejs sadrži atribute, ali u tim slučajevima atributi su obično statični i često konstantni. U UML-u interfejs može biti prikazan oznakom za klasu sa stereotipom <<*interface*>> (*Stereotype Notation*) ili kao lopta (*Ball Notation*).



UML – Dijagram klasa

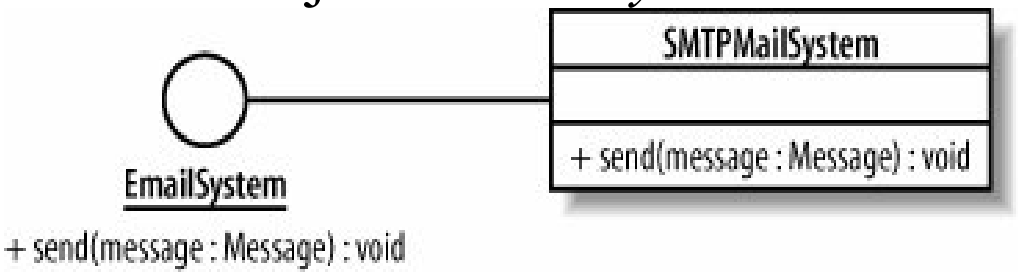
Interfejs vs apstraktna klasa



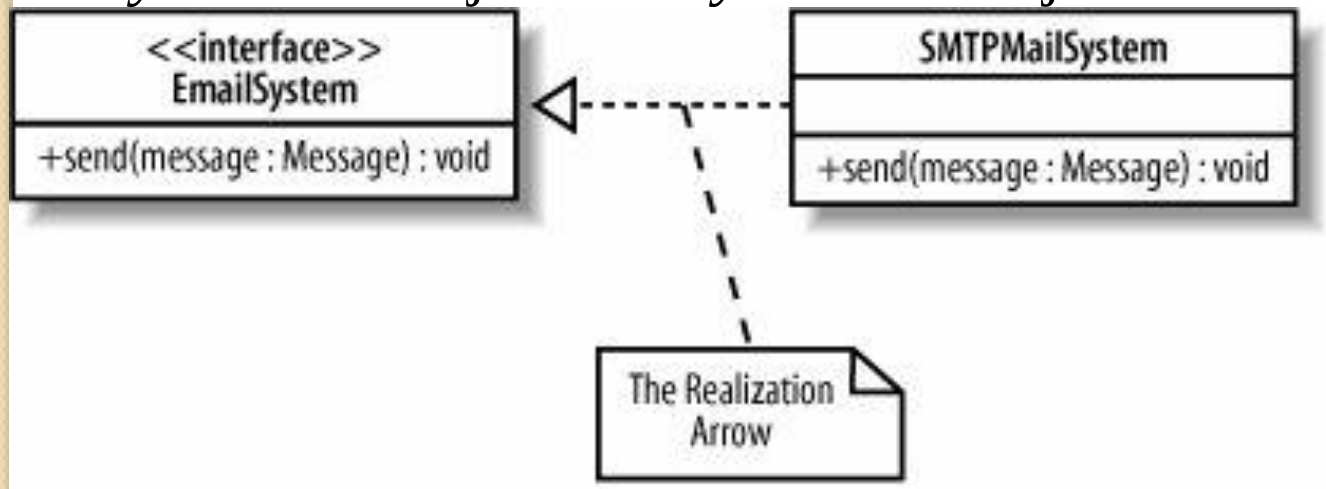


UML – Dijagram klasa

Ne možemo instancirati interfejs, baš kao ni apstraktnu klasu. Na slici *SMTPMailSystem* klasa implementira, ili realizuje, sve operacije specificirane interfejsom *EmailSystem*.



Ako koristite stereotipnu notaciju za svoj interfejs, onda je potrebna nova strelica da prikaže vezu realizacije. Strelica realizacije specificira da *SMTPMailSystem* realizuje *EmailSystem* interfejs.



UML – Dijagram klasa

```
public interface EmailSystem
{
   public void send(Message message));
}
public class SMTPMailSystem implements EmailSystem
{   public void send(Message message)
            {     // Implementira interakciju sa SMTP serverom za slanje
            poruka
            }    // ... Implementacija drugih operacija...
}
```

Interfejsi su **odlični u razdvajanju ponašanja** koje se zahteva od klase i načina kako su **implementirani**.
Kada klasa implementira interfejs, objekti te klase mogu da se prepoznaju koristeći ime interfejsa, a ne samo preko imena klase.
Tako druge klase mogu zavisiti od interfejsa, a ne od same klase.
Ovo osigurava da su klase slabo povezane (*loosely coupled*) jer kada se promeni njena implementacija (neke metode), ostale klase ne bi trebale da se menjaju (zato što zavise od interfejsa, a ne od klase).

```
namespace InterfaceApplication {

   public interface ITransactions {
      // interface members
      void showTransaction();
      double getAmount();
   }
   public class Transaction : ITransactions {
      private string tCode;
      private string date;
      private double amount;

      public Transaction() {
         tCode = " ";
         date = " ";
         amount = 0.0;
      }
      public Transaction(string c, string d, double a) {
         tCode = c;
         date = d;
         amount = a;
      }
      public double getAmount() {
         return amount;
      }
      public void showTransaction() {
         Console.WriteLine("Transaction: {0}", tCode);
         Console.WriteLine("Date: {0}", date);
         Console.WriteLine("Amount: {0}", getAmount());
      }
   }
   class Tester {

      static void Main(string[] args) {
         Transaction t1 = new Transaction("001", "8/10/2012", 78900.00);
         Transaction t2 = new Transaction("002", "9/10/2012", 451900.00);

         t1.showTransaction();
         t2.showTransaction();
         Console.ReadKey();
      }
   }
}
```

```
Transaction: 001
Date: 8/10/2012
Amount: 78900
Transaction: 002
Date: 9/10/2012
Amount: 451900
```

UML – Dijagram klasa

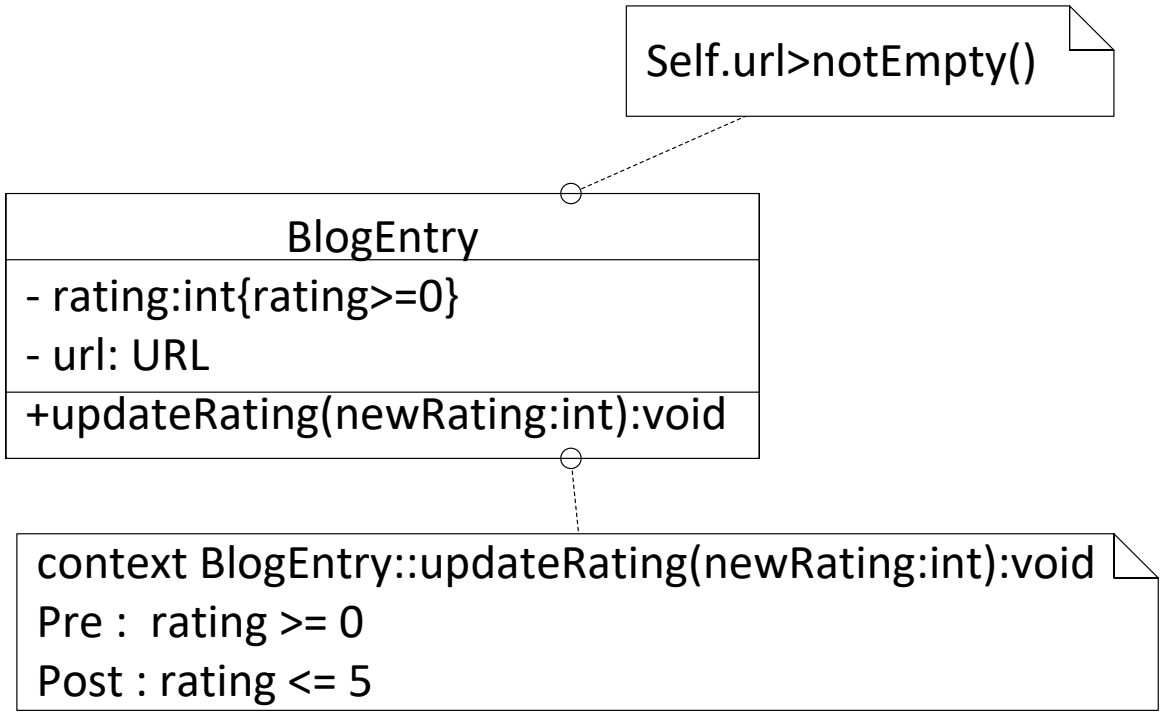
Neophodan i jedan formalan jezik za specifikaciju ograničenja **OBJECT CONSTRAINT LANGUAGE (OCL).** Postoje tri tipa ograničenja koja se mogu primeniti na pripadnike klasa primenjivanjem OCL-a:

Nepromenljiva su ona ograničenja koja uvek moraju biti tačna, u suprotnom sistem će biti u stanju koje prikazuje grešku. Nepromenljiva ograničenja se definišu na atributima klase.

Preduslov je ograničenje koje se definiše na metodi i koji se proverava pre nego što se izvrši metoda. On se najčešće koristi za validaciju ulaznih parametara metode.

Postuslov se takođe definiše na metodi, samo što se on proverava nakon izvršenja metode. Postuslov se najčešće koristi da opiše kako su vrednosti promenjene metodom.

UML – Dijagram klasa



url atribut je ograničen da nikada ne bude *null (notEmpty)*. Da bi se osigurali da operacija *updateRating*(…) ne radi sa vrednostima manjim od 0, postavljen je preduslov (*rating*>=0). I na kraju, atribut *rating* nikada ne bi trebalo da bude manje od 5 nakon ažuriranja, tako da je ovo specificirano kao postuslov ograničenje na operaciji *updateRating*(…).

UML – Dijagram klasa - primer

Na osnovu sledećeg UML dijagrama klasa odgovoriti na postavljena pitanja:

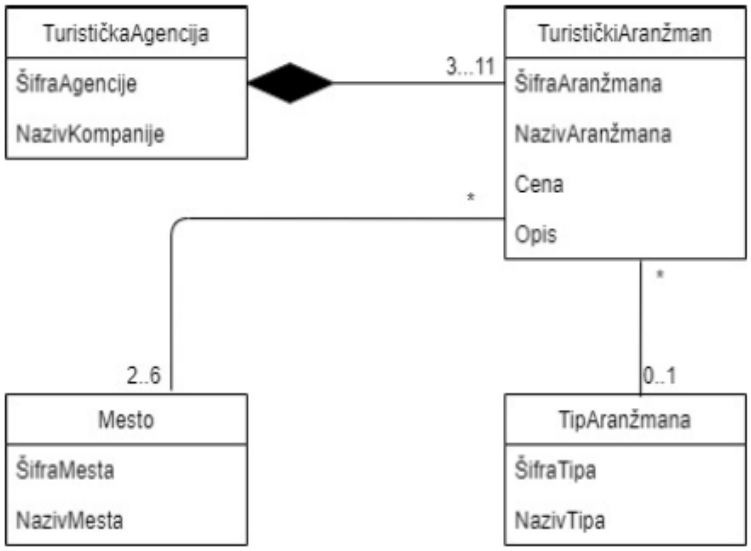
a) Koliko minimalno, a koliko maksimalno Mesta može da postoji za jedan Turistički aranžman?

b) Da li se određeni Tip aranžmana mora naći u ponudi neke Turističke agencije?

c) Da li dve Turističke agencije mogu da ponude isti Turistički aranžman?

d) Da li svaki Turistički aranžman mora pripadati nekom Tipu aranžmana?

e) Da li mogu da postoje dva Turistička aranžmana sa istim nazivom?



Projektovanje informacionih sistema

HVALA !

dr Rade Matić