

# **OBJEKTNE BAZE**

## OBJEKTNE BAZE

- Poslednja decenija u softverskom inženjerstvu je **decenija "objektne" orijentacije**. Objektna orijentacija je pristup u kome se neki sistem organizuje kao kolekcija međusobno povezanih objekata koji, sarađujući, ostvaruju postavljene ciljeve. Objektna orijentacija je danas preovlađujuća u **programiranju i programskim jezicima, metodološkim pristupima razvoju sofvera i informacionih sistema**, a postepeno preuzima primat i u **sistemima za upravljanje bazom podataka, bilo preko "čistih" objektnih SUBP, bilo preko objektno relacionih SUBP** koji predstavljaju proširenje relacionih sa objektnim konceptima.

## OBJEKTNE BAZE- TIPOVI PODATAKA

- Jedna od najbitnijih karakteristika SUBP-a je skup tipova podataka koje on podržava.
- Konvencionalni SUBP (hijerarhijski, mrežni i relacioni) su zasnovani na tipu rekorda koji predstavlja agregaciju polja (atributa u relacionom modelu) definisanih nad standardnim tipovima. U relacionom modelu se još definiše i tabela kao skup rekorda, u mrežnom se uvodi pojam "rekord seta", a u hijerarhijskom pojam "stabla" rekorda.
- Objektni SUBP su postavili sebi cilj da stvore mnogo bogatiji skup tipova i time omoguće prirodniju manipulaciju podacima u aplikacijama koje se razvijaju nad bazom.

## OBJEKTNE BAZE

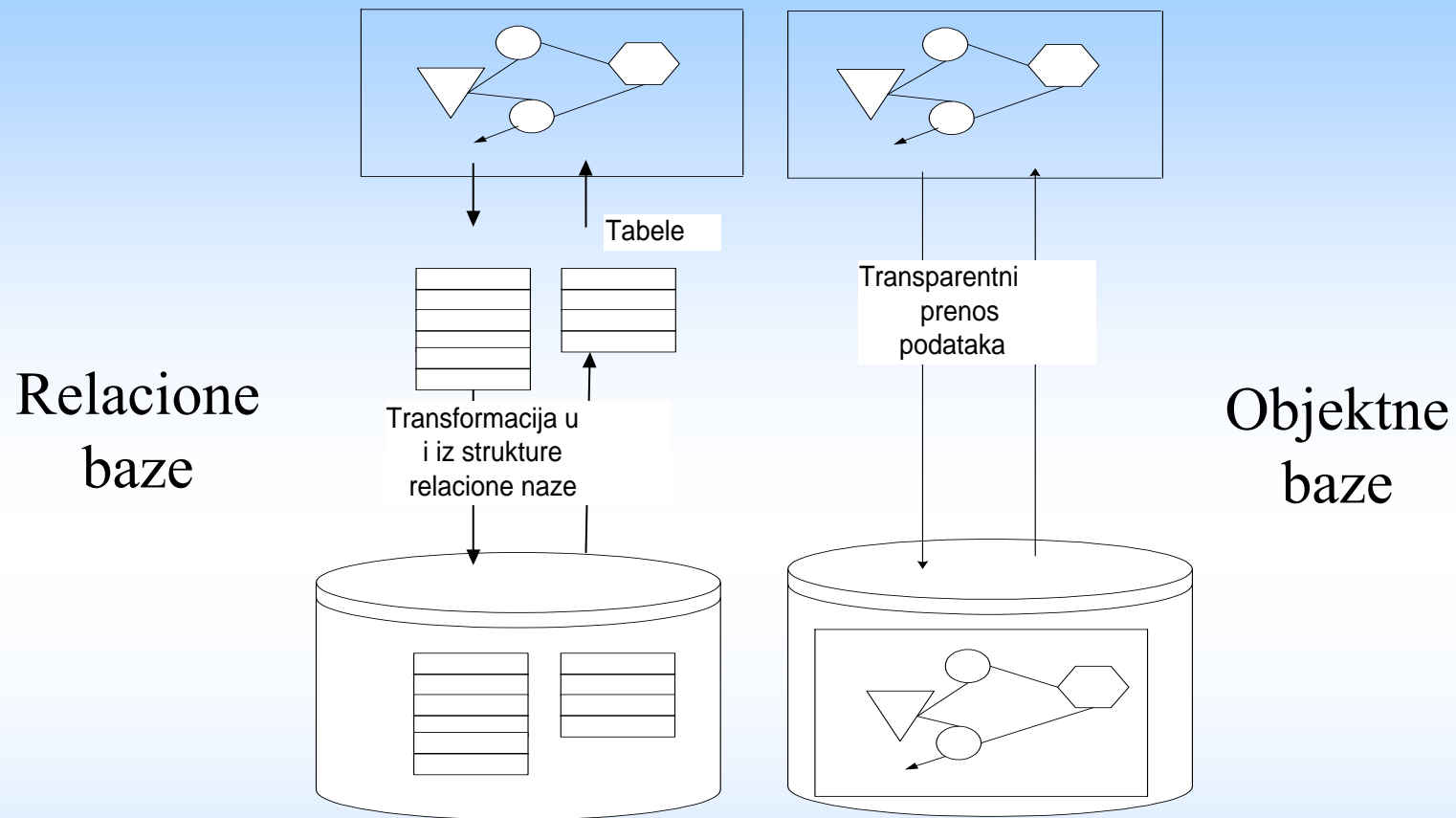
- U konvencionalnim SUBP, trionivska ANSI/SPARC arhitektura imala je za cilj da omogući da se baza podataka tretira kao samostalna komponenta nekog IS. Zbog toga je Jezik baze podataka (DDL i DML) u ovim SUBP potpuno odvojen (nezavisan) od programskih jezika u kojima se razvijaju aplikacije, omogućujući na taj način da se nad jednom bazom podataka razvijaju i izvršavaju aplikacije napisane i u različitim programskim jezicima.

**Nezavisnost Jezika baze podataka od programskih jezika, bez obzira na probleme koji su proisticali iz njihove moguće nesaglasnosti, tretirala se kao prednost, a ne kao nedostatak ovih SUBP-a.**

## OBJEKTNE BAZE

- Objektni SUBP teže da integrišu funkcije SUBP-a u programski jezik da bi se nesaglasnost Jezika baze podataka i programskog jezika eliminisale i značajno poboljšale performanse sistema.
- Teži se tome da se izjednače u potpunosti objekti baze podataka i objekti aplikacija napisanih u nekom objektnom programskom jeziku. Objekti u aplikacijama su ***tranzijentni***, životni vek im je jednak trajanju aplikacije, dok su objekti u bazi podataka ***perzistentni***, nezavisni od postojanja aplikacija koje ih koriste.
- Ne eliminiše se mogućnost da više programa, napisanih čak i u različitim objektnim jezicima, koriste istu bazu podataka.
- Raspoložući sa mnogo bogatijim skupom tipova, objektni SUBP treba da podrže i znatno moćniji upitni jezik od relacionog (SQL-a).

## STRUKTURE APLIKACIONIH PROGRAMA



U nekom objektnom jeziku jednostavno se mogu realizovati složene strukture podataka i takve usladištiti u OBP. U slučaju relacionih baza podataka, ta struktura podataka se prvo transformiše u odgovarajući skup tabela relacione baze, što je ponekad složen i vremenski zahtevan postupak.

## OBJEKTNE BAZE

- Rani komercijalni objektni SUBP nisu bili zasnovani na jedinstvenom standardu, navedene ciljeve su ostvarivali, sa manje ili više uspeha, na različite načine. Njihov komercijalni prodor je bio zanemarljiv.
- Uspeh relacionih SUBP, pored već diskutovanih prednosti, bio je i u tome što su zasnovani na jedinstvenom standardu, SQL-u, što je značajno podiglo njihovu prihvatljivost, prenosivost i mogućnost kooperacije različitih sistema.
- Zbog toga se u poslednjih nekoliko godina intenzivno razvija standard za objektnu bazu podataka u okviru takozvane ODMG (Object Database Management Group). Verzija 1.2. ODMG standarda pojavila se 1996, a značajno izmenjena i dopunjena verzija 2.0, 1997. godine. Ovde se izlažu elementi ODMG 2.0 standarda.



## ARHITEKTURA OBJEKTNIH SUBP – komponente

- Objektni model.
- Objektni specifikacioni jezici
- Objektni upitni jezik
- C++, Smalltalk, Java Language Binding



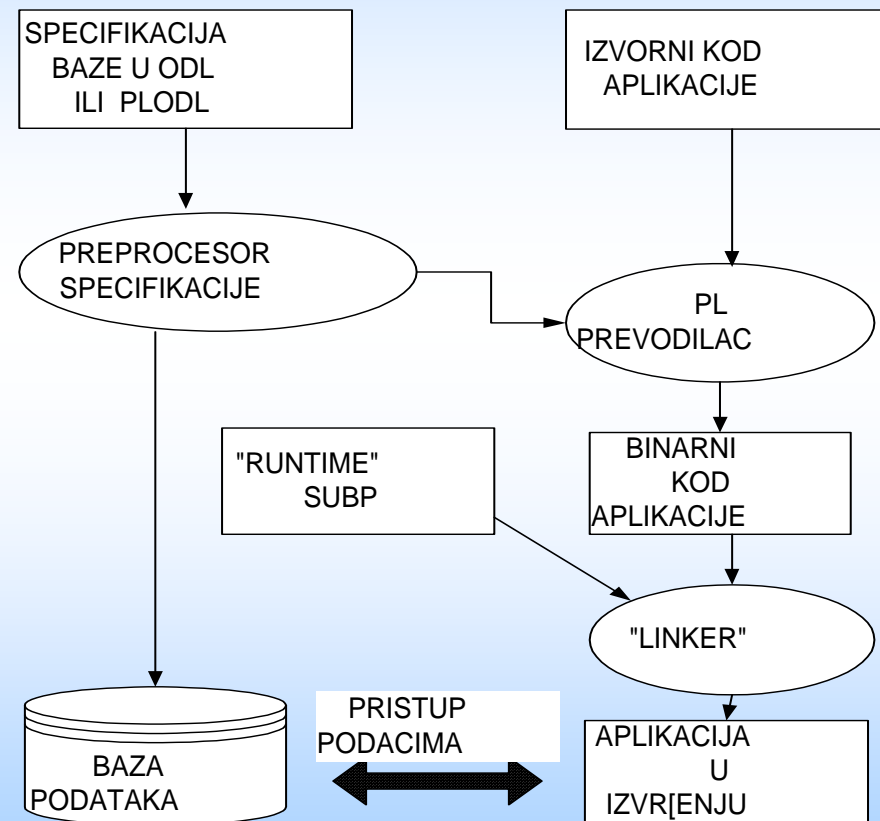
## ARHITEKTURA OBJEKTNIH SUBP – komponente

- ***Objektni model.*** Objektni model na kome treba da bude zasnovan svaki objektni SUBP (ODMG model) izveden je iz OMG (Object Management group) objektnog modela. OMG je objektni model definisan kao zajednička osnova za objektnu programske jezike, komunikaciju objekata u nekoj klijent server arhitekturi (Object Request Brockers, ORB, na primer CORBA) i objektnu baze podataka.
- ***Objektni specifikacioni jezici*** koji služe da opišu sistem kao skup međusobno povezanih objekata. Objektni specifikacioni jezik - ODL (Object Definition Language) ima istu ulogu koju JOP (DDL) ima u konvencionalnim jezicima.

## ARHITEKTURA OBJEKTNIH SUBP – komponente

- ***Objektni upitni jezik***, neproceduralni jezik za postavljanje upita i modifikaciju baze podataka. Ovaj upitni jezik nazvan OQL (Object Query Language) je namerno veoma sličan SQL-u, sa većim mogućnostima koje pruža objektni model.
- ***C++, Smalltalk, Java Language Binding***, su posebne nezavisne komponente arhitekture koje pokazuju kako se u C++ (Smalltalk, Java) ugrađuju mogućnosti manipulacije perzistentnim objektima, mehanizmi za povezivanje sa OQL-om, upravljanje transakcijama i slično. Kao što je već rečeno, za razliku od konvencionalnih JMP jezika, JMP u objektnim bazama se “kroje” prema “jeziku domaćinu”.

# ARHITEKTURA OBJEKTNIH SUBP – korišćenje SUBP



## OBJEKTNI MODEL

- Bilo koji sistem se može posmatrati kao ***skup međusobno povezanih objekata***. Pod objektima u nekom sistemu se podrazumevaju fizički objekti, koncepti, apstrakcije, bilo šta što ima jasne granice i jasno značenje, što se jasno razlikuje od drugih objekata u sistemu.
- U realnom sistemu, objekti i način ostvarivanja njihovih veza mogu da budu veoma raznovrsni.
- U nekoj vrsti ***modela realnog sistema*** svi ti raznovrsni objekti i njihove veze predstavljaju se sa određenim (malim) brojem precizno definisanih koncepata. Modeli koji služe za opis baze podataka, nazivaju se ***modeli podataka***.

## OBJEKTNI MODEL: OSNOVNI POJMOVI

- Osnovni primitivni koncepti objektnog modela su ***objekat i literal***.
- Pod objektom se podrazumeva entitet koji je sposoban da čuva svoja ***stanja*** i koji stavlja okolini na raspolaganje skup ***operacija*** preko kojih se ta stanja prikazuju ili menjaju.
- Literal je u osnovi vrednost, podatak koji se koristi u modelu. Brojevi i karakteri su primeri "atomskih" literala, a datum je primer složenog (struktuiranog) literala.
- Analogija sa OO Programskim jezicima: "Objekti neke klase su bilo vrednosti datog tipa (immutable objects - literal) ili promenljive koje mogu uzimati vrednosti datog tipa (mutable object - objekat)".
- Posebno je potrebno naglasiti sledeću razliku objekta i literala: *objekat ima jedinstveni identifikator, a literal nema.*

## OBJEKTNI MODEL: OSNOVNI POJMOVI

- Objekti i literali se kategorizuju u ***tipove***. Svi objekti, odnosno literali istog tipa imaju zajednički skup stanja i jedinstveno ponašanje. Konkretni objekat se ponekad naziva ***pojavljivanje (instanca)*** datog tipa. Za pojavljivanja literala se pretpostavlja da implicitno postoje.
- ***Stanje objekta*** se predstavlja vrednostima njegovih ***osobina***. Osobine su ***atributi objekta*** i njegove ***veze*** sa drugim objektima u sistemu.
- ***Ponašanje objekta*** se opisuje preko skupa ***operacija*** koje on izvršava ili se nad njim izvršavaju. Svaka operacija ima kao implicitni argument objekat kome je pridružena. Operacija može da ima i listu ulaznih parametara definisanih tipova, a može i da vrati tipizovan rezultat.

## OBJEKTNI MODEL: OSNOVNI POJMOVI

- ***Baza podataka*** skladišti objekte i stavlja ih na korišćenje većem broju korisnika, odnosno aplikacija. Baza podataka se opisuje preko svoje ***šeme*** koja se specifikuje preko ODL-a. U šemi se definišu tipovi objekata čija se pojavljivanja čuvaju u bazi.

## SPECIFIKACIJA I IMPLEMENTACIJA TIPRA

- Svaki tip ima jednu **specifikaciju** i jednu ili više **implementacija**.
- Specifikacija definiše eksterne karakteristike, vidljive korisniku tipa: **operacije** koje mogu biti pozvane, osobine kojima se može pristupiti i **izuzeci** (exceptions) koje operacije pobuđuju.
- Implementacija tipa definiše interne karakteristike objekata datog tipa. Moguće je definisati više implementacija jedne specifikacije u jednom ili u više različitih jezika.
- Odvajane specifikacije od implementacije tipa je veoma bitno, jer se na taj način ostvaruje **učaurenje** (encapsulation) objekta, korišćenje servisa koji objekat pruža okolini, bez poznavanja implementacije.



# SPECIFIKACIJA I IMPLEMENTACIJA TIPRA

- Specifikacija tipa se daje preko:
  - **interfejsa** koji predstavlja samo apstraktno ponašanje
  - **klase** koja predstavlja i apstraktno stanje i apstraktno ponašanje tipa objekta.
  - definicija literala daje samo apstraktno stanje tipa literala.
    - **interface** Radnik {...};
    - **class** Lice {...};
    - **struct** Kompleks {**float** reldeo, **float** imdeo};
- Klasa je tip koji se može direktno instancirati, pojavljivanja ovoga tipa mogu se kreirati u nekom programu ili bazi. Interfejs je tip koji se ne može direktno instancirati.

## SPECIFIKACIJA I IMPLEMENTACIJA TIPRA

- Implementacija tipa, u bilo kom programskom jeziku, ostvaruje se preko ***struktura podataka*** za opis stanja i skupa **metoda**, preko kojih su implementirane operacije tipa. U većini objektnih jezika postoji ***koncept klase***, pa se može reći da je u njima klasa osnovni mehanizam za implementaciju tipova objekata. Koncept klase u objektnim jezicima ne treba izjednačavati sa pojmom klase u ODMG.
- Specifikacija tipova neke baze podataka se definiše preko ODL-a, a njihova implementacija u okviru povezivanja sa jezicima (language bindings). Oni definišu način transformacije klase ODMG u sopstveni implementacioni koncept klase.

# UGRAĐENI TIPOVI

- *Tipovi literala*

- *Atomski tipovi*

- long, short, unsigned long, unsigned short, float, double, boolean, octet, char, string, enum <t>

- *Kolekcije literala*

- set <t>, bag<t>, list<t>, array<t>, dictionary<t,p>

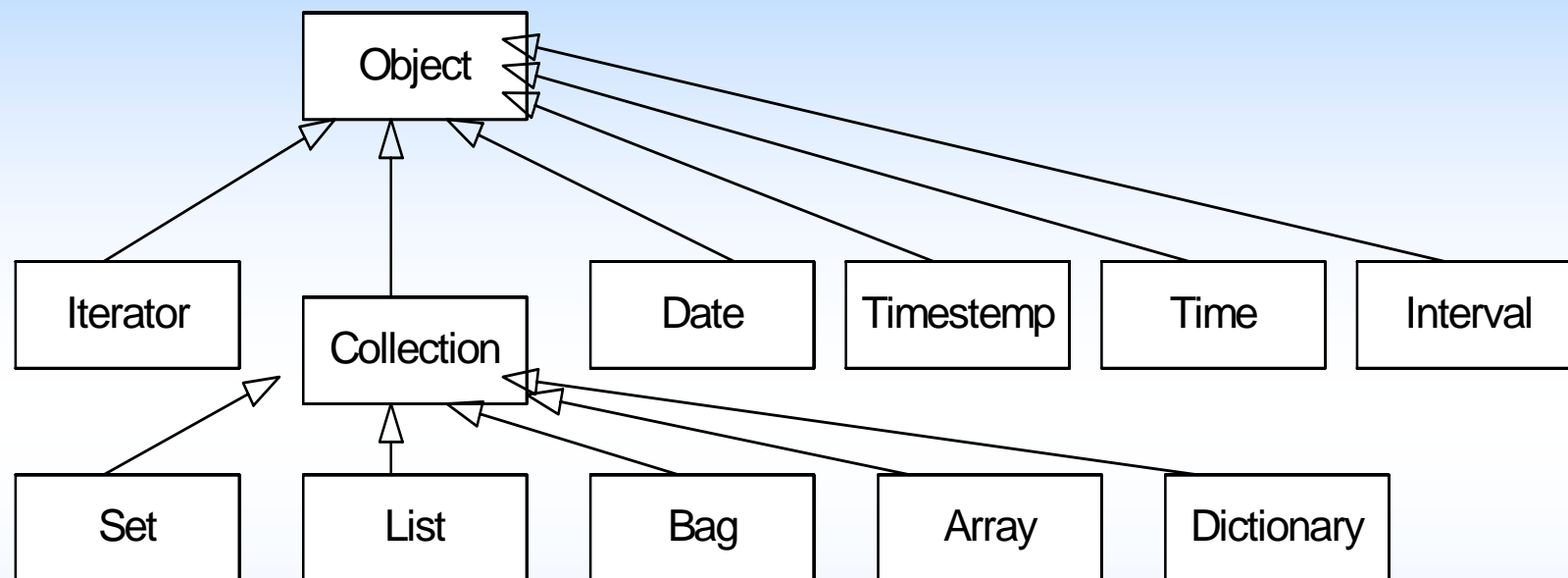
- *Strukturirani literali*

- date, time, timestamp, interval, structure<t>

# UGRAĐENI TIPOVI

- *Tipovi objekata*
  - *Kolekcije objekata*  
Set<t>, Bag<t>, List<t>,  
Array<t>, Dictionary<t>
  - *Stukturirani objekti*  
Date, Time, Timestamp, Interval
- Ne postoji ugrađeni tip atomskog objekta. Svi atomski objekti su korisnički definisani.

# HIJERARHIJA UGRAĐENIH TIPOVA OBJEKATA



U ODMG svi objekti nasleđuju operacije osnovnog interfejsa **Object**. Objekti se kreiraju pozivajući operacije interfejsa **ObjectFactory**, a kolekcije pozivanjem operacija interfejsa **CollectionFactory**

## UGRAĐENI TIPOVI

```
interface ObjectFactory {  
    Object new();  
};
```

```
interface Object {  
    ....  
    boolean same_as(in Object  Objekat);  
    Object copy();  
    void delete();  
};
```

Operacije se primenjuju na objekte korišćenjem "**dot**" notacije.

```
o.same_as(p);
```

## UGRAĐENI TIPOVI

*Pošto jedan objektni SUBP može da upravlja sa više baza podataka definiše se objekat za "fabrikovanje" baza kao i objekat koji definiše samu bazu.*

```
interface databaseFactor{
```

```
    Database new();};
```

```
interface Database {
```

```
    exception ElementNotFound {};
```

```
    void open(in string ime_base);
```

```
    void close ();
```

```
    void bind (in any neki_obj, in string ime_obj);
```

```
    Object unbind(in string ime_objekta);
```

```
    Object lookup (in string ime_objekta)
```

```
        raises(ElementNotFound);
```

```
    ....};
```

## UGRAĐENI TIPOVI

```
interface CollectionFactory : ObjectFactory {  
    Collection  new_of_size(in long velicina);  
};
```



# UGRAĐENI TIPOVI

```
interface Collection {  
    exception          InvalidCollectionType{};  
    exception          Element not found{ bilo koji el};  
    unsigned long      cardinality();  
    boolean            is_empty();  
    boolean            is_ordered();  
    boolean            allows_duplicates();  
    boolean            contains_element(in neki element);  
    void               insert_element (in neki element);  
    viod              remove_element(in neki element);  
                                raises (ElementNotFound);  
    Iterator          create_iterator(in boolean stable);  
    BidirectionalIterator create_bidirectional_iterator(in  
                                                        boolean stable)  
  
    raises(InvalidCollectionType);;}
```

## UGRAĐENI TIPOVI

```
interface Iterator {  
    exception NoMoreElements{};  
    exception InvalidCollectionType{};  
    boolean is_stable();  
    boolean at-end();  
    void reset();  
    any      get_element() raises (NoMoreElements);  
    void      next_position (in neki element);  
    void      replace_element( in neki element)  
        raises InvalidCollectionType);  
};
```

## UGRAĐENI TIPOVI

```
interface Set: Collection{  
    Set      create_union(in Set drugi-skup);  
    Set      create_intersection(in Set drugi-skup);  
    Set      create_difference(in Set drugi-skup);  
    boolean  is_subset_of(in drugi-skup);  
    boolean  is_proper_subset_of(in drugi-skup);  
    boolean  is_superset_of(in drugi-skup);  
    boolean  is_proper_superset_of(in drugi-skup);  
};
```

## MODELIRANJE STANJA

- Stanje pojavljivanja nekog tipa objekta iskazuje se preko vrednosti njegovih osobina. Postoje dve vrste osobina: **atributi i veze**.
- **Atribut**. Atributi nekog tipa objekta se specifikuju preko tipova literala ili tipova objekata. Drugim rečima, vrednost atributa može da bude literal ili identifikator nekog objekta.

## PRIMER DEFINISANJA ATRIBUTA KLAZE

```
interface Lice {  
    attribute short    starost;  
    attribute string   ime;  
    attribute enum    pol {muški, ženski};  
    attribute    Adresa kućna-adresa;  
    attribute set <String>   telefon;  
    attribute    Organizacija zaposlen;  
};
```

Pretpostavlja se da je definisan objekat Organizacija i

```
struct Adresa {string grad,  
               string ulica_i_broj};
```

## MODELIRANJE STANJA - VEZE

- ODMG podržava samo binarne veze (relationship), tj. veze između dva tipa objekta.
- Veza se definiše implicitno, unutar opisa tipa objekta, kao tzv "prelazna putanja" (traversal path).
- "Prelazna putanja" se uvek deklariše u paru, u okviru deklaracije oba tipa koji su u vezi. Deklariše se, na primer, da radnik **radi** u preduzeću (u okviru tipa Radnik) i da preduzeće **zapošljava** radnika (u okviru tipa Preduzeće).
- Činjenica da ovakve dve deklaracije predstavljaju jednu vezu iskazuje se u ODL-u preko klauzule **inverse**.

## PRIMER DEKLARISANJA VEZE

**interface** Radnik {

.....

**relationship** Preduzeće radi

**inverse** Preduzeće:: zapošljava;

.....

};

**interface** Preduzeće {

.....

**relationship set** <Radnik> zapošljava

**inverse** Radnik:: radi;

.....

};

## VEZE

- Kardinalnost veze se definiše preko činjenice da veza "uzima" kao vrednost jedan atomski objekat (kardinalnost "jedan") ili neku njihovu kolekciju (kardinalnost "više").
- Objektni SUBP treba da ostvari integritet veze: ako je objekat koji učestvuje u vezi izbrisan iz baze tada se i svaka "prelazna putanja" prema tom objektu, takođe briše.
- Veza u OMG-u je uvek dvosmerna. Pri implementaciji veze u nekom pridruženom jeziku veza se može učiniti jednosmernom. Isto tako, kao što je ranije pokazano, atribut čija je vrednost identifikator nekog objekta, može da posluži za iskaz jednosmernih veza.



# MODELOVANJE PONAŠANJA – OPERACIJE

- Ponašanje tipa predstavlja se skupom operacija
- Sintaksa za specifikaciju operacije:

```
Specifik_oper:: tip_rezult naziv_oper(lista_argum)
                raises(lista_naziv_izuzetka)
```

lista\_argum:: argument | argument, lista\_argum

## argument::parametar\_argumenta

specifik\_ tipa\_argumenta naziv\_argumenta

## parametar\_argumenta:: in | out | inout

```
lista_naziv_izuzetka:: naziv_izuzetka |
```

naziv\_izuzetka, lista\_naziv\_izuzetka

## Primer operacije je:

**Boolean** upisan\_na\_predmet (**in unsigned short** kurs)  
**raises** (nema\_uslova, popunjen\_broj)

## NASLEĐIVANJE

- U ODMG se definišu dve posebne vrste nasleđivanja: *nasleđivanje ponašanja* i *nasleđivanje stanja*.
- Za nasleđivanje ponašanja se koristi veza nadtip-podtip (koja se ponekad naziva veza generalizacija-specijalizacija ili ISA veza. Nadtip u nasleđivanju ponašanja mora da bude Interfejs.
- Za nasleđivanje stanja koristi se specifična veza EXTENDS (Proširenje). U ovoj vezi između klasa, podređena klasa nasleđuje celokupno stanje i ponašanje klase koju proširuje.

## NASLEĐIVANJE PONAŠANJA

Za nasleđivanje ponašanja se koristi veza nadtip-podtip. Činjenica da je Nastavnik podtip Radnika, a Asistent podtip Nastavnika označava se na sledeći način:

**interface** Radnik {...specifikacija tipa Radnik..}

**interface** Nastavnik: Radnik {...specifikacija tipa Nastavnik..}

**interface** Asistent: Nastavnik {...specifikacija tipa Asistent..}

**class** StalniRadnik: Radnik {...specifikacija tipa StalniRadnik..}

**class** PrivremeniRadnik: Radnik {...specifikacija tipa PrivremeniRadnik..}

## NASLEĐIVANJE PONAŠANJA

- U podtipu se može i redefinisati ponašanje specifikovano u nadtipu. Na primer, ako je u tipu Radnik specifikovana operacija ObračunZarade, ista operacija se može različito implementirati za podtipove StalniRadnik i PrivremeniRadnik. Osobina da se ista operacija izvršava na različit način u zavisnosti od tipa objekta sa kojim se izvršava, naziva se **polimorfizam**.
- U ODMG modelu podržano je višestruko nasleđivanje ponašanja. Tada se, međutim, može desiti da dve (ili više) nasleđene operacije imaju isti naziv, a različit broj i/ili tip argumenata. Da bi se ovaj problem izbegao, nije dozvoljeno "preopterećenje" (overloading) imena operacija (davanje istih imena operacijama različitih tipova) u takvoj hijerarhiji nasleđivanja.

# NASLEĐIVANJE STANJA

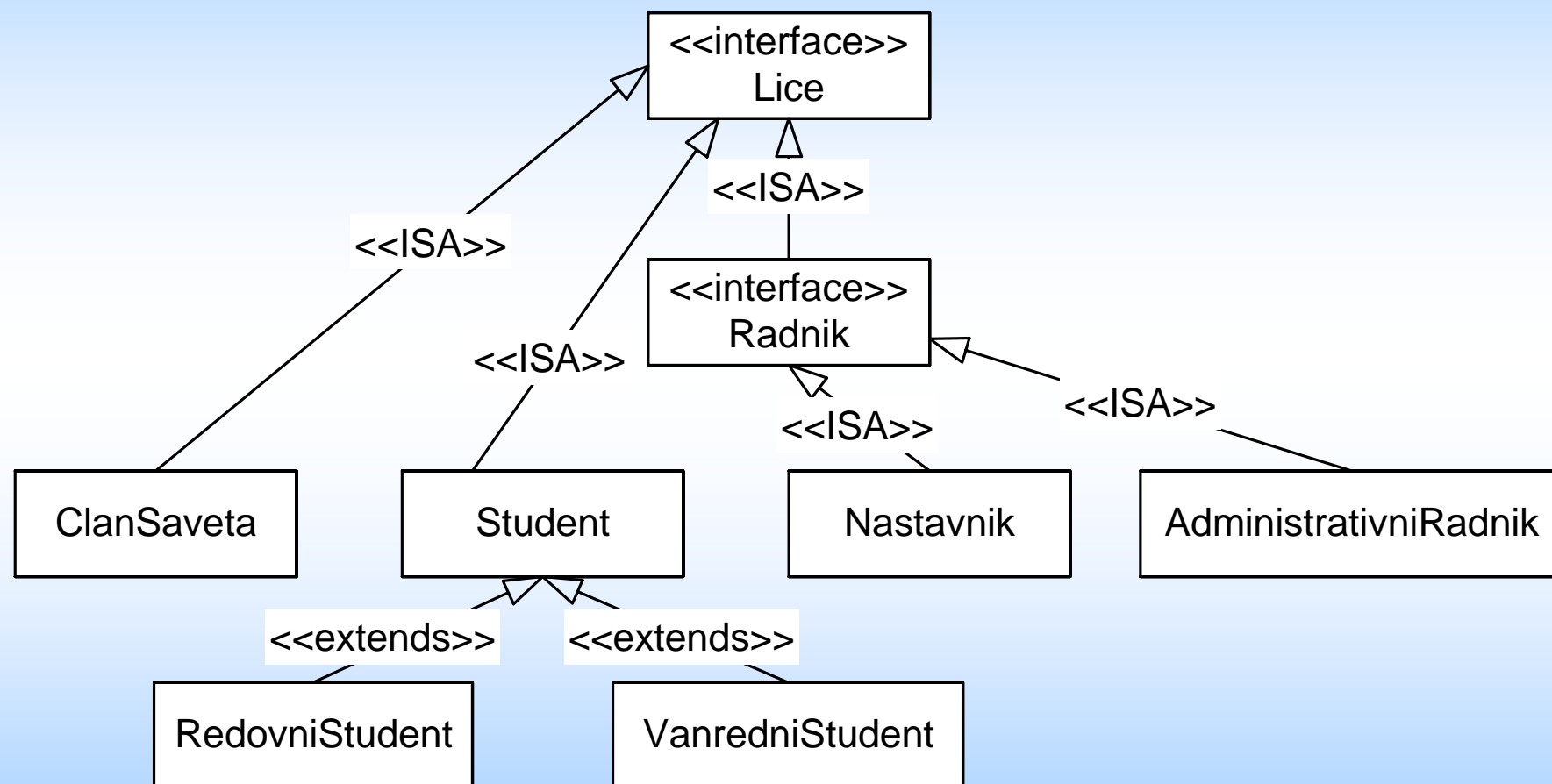
- ODMG model uvodi EXTENDS vezu za definisanje nasleđivanja stanja. Na primer,

```
class Lice {  
    attribute string ime;  
    attribute Date datumRođenja;};  
class ZaposlenoLice extends Lice: Radnik {  
    attribute Date datumZapošljavanja;  
    attribute Carency plata;  
    relationship Rukovodilac šef inverse Rukovodilac ::  
        podređeni;};  
class Rukovodilac extends ZaposlenoLice {  
    relationship set <ZaposlenoLice> podređeni inverse  
        ZaposlenoLice šef;};
```

## NASLEĐIVANJE

- Odnos tip-podtip (generalizacija-specijalizacija) u ODMG modelu se primenjuje samo na nasleđivanje ponašanja. Zbog toga interfejs i klasa mogu biti podtipovi samo interfejsa. Interfejsi i klase ne mogu biti podtipovi klasa.
- Pošto se interfejs ne može instancirati, on (slično apstraktnoj klasi u nekim OO jezicima) služi samo za to da se iz njega naslede neke operacije.
- Pošto se preko odnosa podtip/nadtip nasleđuje samo ponašanje, ako se želi u podtipu isti atribut koji postoji u interfejsu koji je nadtip, on se u podtipu mora "kopirati".

# NASLEĐIVANJE - UML notacija

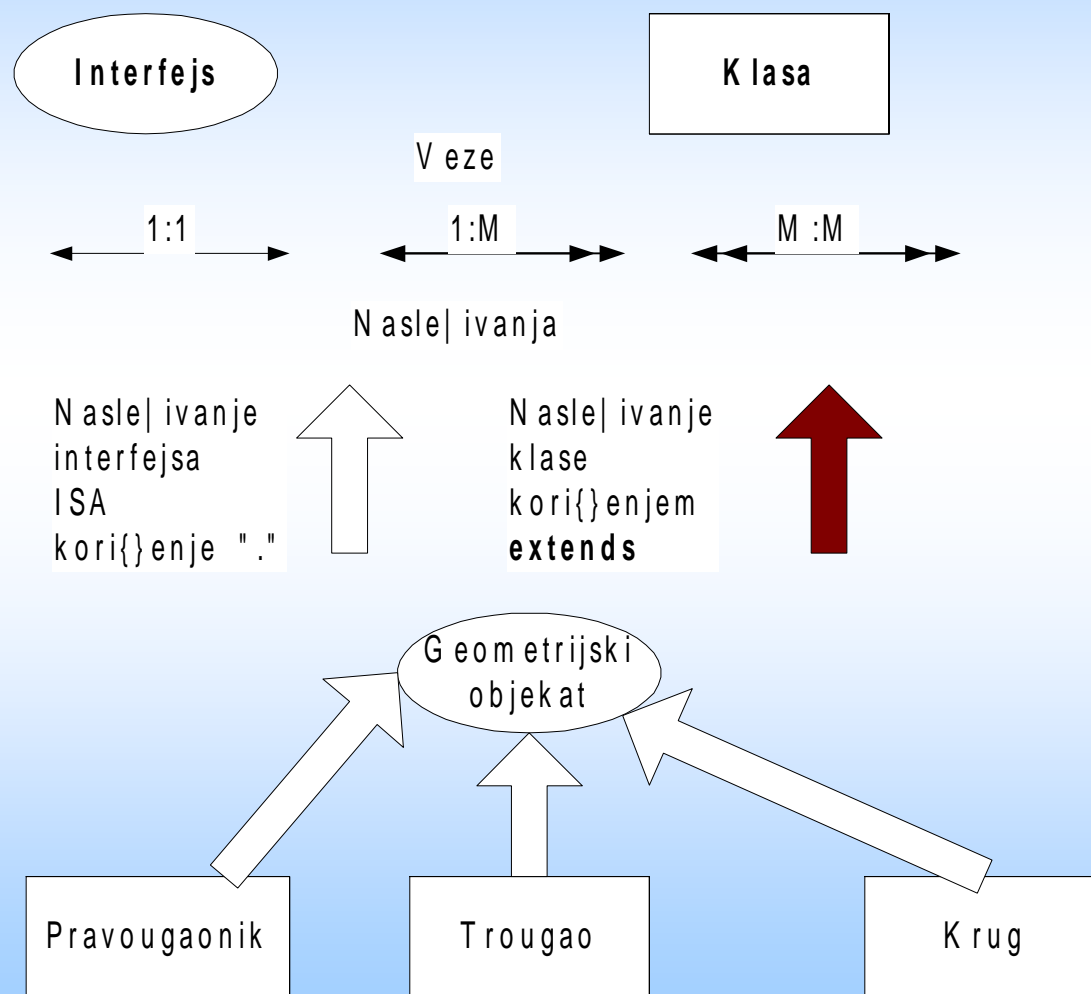


## OPSEG (EXTENT) TIPA

- Opseg tipa je skup svih instanci datog tipa u okviru posmatrane baze podataka. Ako je neki objekat instanca tipa A, tada je on element opsega A.
- U konvencionalim bazama podataka čuvaju se sva pojavljivanja tipova definisanih u njima. U objektnim bazama, projektant može da odluči da li će se opseg tipa čuvati u bazi ili ne. Objektni SUBP treba da obezbedi ubacivanje i izbacivanje pojavljivanja u i iz opsega, kao i kreiranje indeksa.
- Moguće je definisati da jedan atribut (za prost) ili više atributa (za složen) predstavljaju korisnički ključ tipa. Ključ služi za jedinstvenu identifikaciju, preko atributa definisanih u njegovoj specifikaciji, jednog pojavljivanja u opsegu tipa.



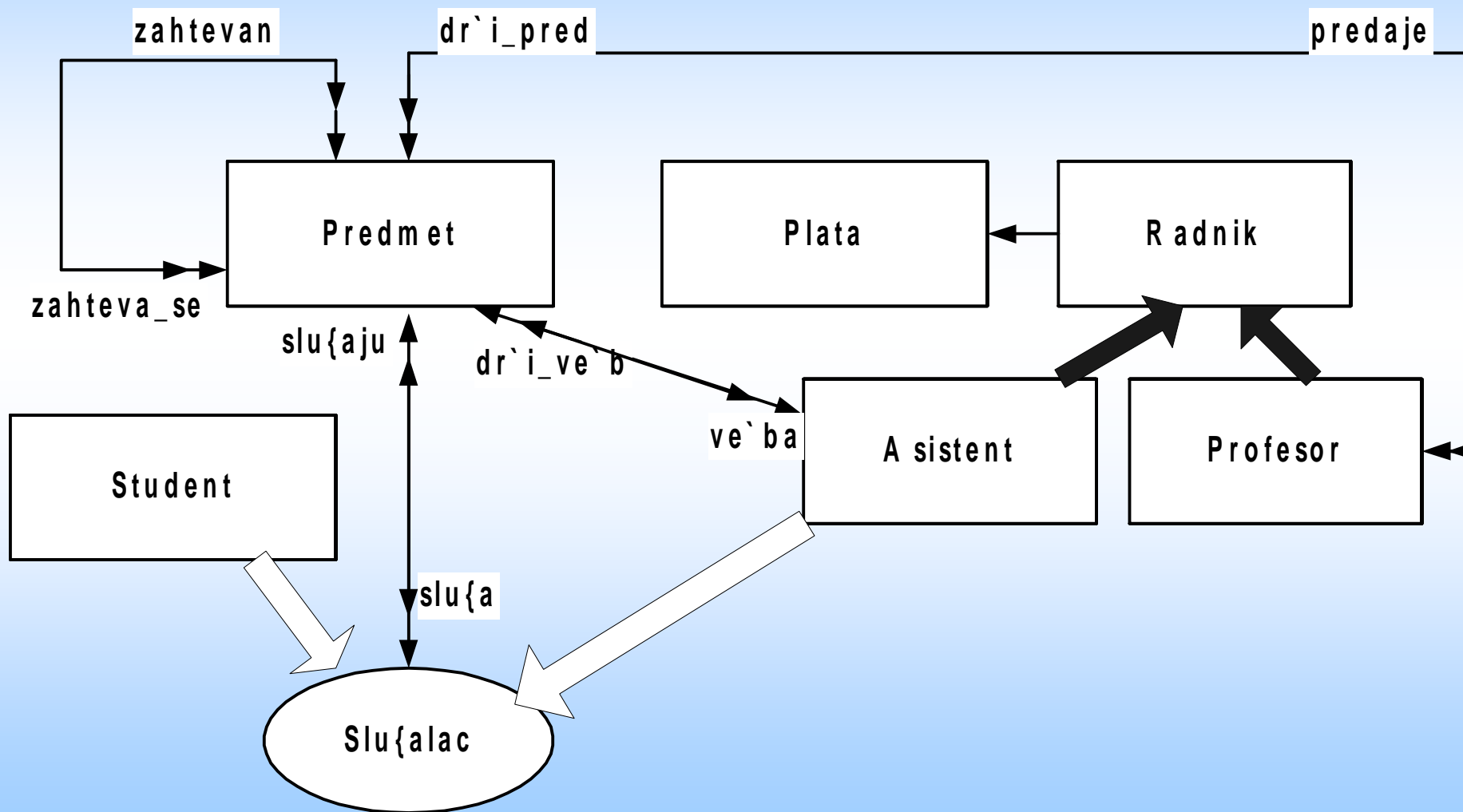
# GRAFIČKE OZNAKE ZA PRIKAZ ŠEME BP U ODMG



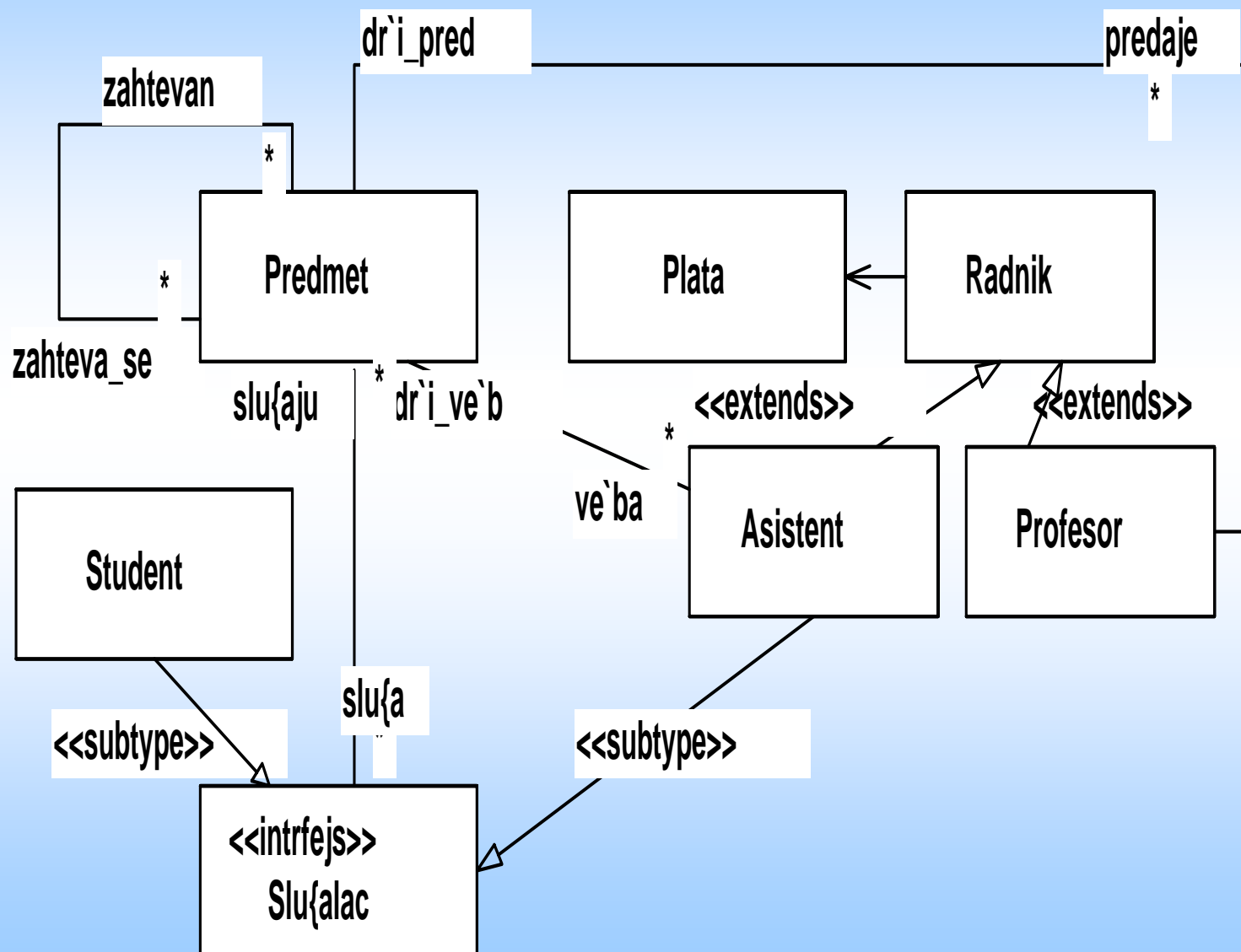
# PRIMER1 - GOEMETRIJSKI OBLICI

```
interface GeometrijskiObjekat {  
    attribute enum Oblik {Pravoug, Trougao, Krug} oblik;  
    attribute struct Tačka {short x, short y} referentna_tačka;  
    float obim ();  
    float površina ();  
    void pomeri (short x_pomeraj, short y_pomeraj);  
    void rotiraj (short ugao);};  
class Pravougaonik: Geometrijski oblik  
    (extent pravougaonici ) {  
    attribute struct Tačka {short x, short y} referentna_tačka;  
    attribute short dužina;  
    attribute short širina;};
```

## PRIMER 2 - PRIKAZ ŠEME SA ODMG OZNAKAMA



## PRIMER2 PRIKAZ ŠEME SA UML OZNAKAMA



## PRIMER 2

```
■ class Kurs
  (extent kursevi
   key sif_kurs)
  {
    attribute string sif_kurs;
    attribute string naziv;
    relationship set <Kurs> zahteva_se inverse Kurs:: zahtevan;
    relationship set <Kurs> zahtevan inverse Kurs:: zahteva_se;
    relationship set <Slušalac> slušaju inverse Slušalac:: sluša;
    relationship Asistent drži_vežb inverse Asistent:: vežba;
    relationship Profesor drži_pred inverse Profesor:: predaje;
    boolean drzi_se (in unsigned short semestar) raises (vec_plan);
    boolean izostavljen (in unsigned short semestar)
      raises(vec_izost);
  };
};
```

## PRIMER 2

```
class Radnik
(  extent radnici
  key id)
{attribute short id;
  attribute string ime;
  attribute Plata mesecna_plata;
  void() zaposli;
  void otpusti() raises(nema_tog_radnika);};

class Plata {
  attribute float osnovna;
  attribute float prekovremena;
  attribute float minuli_rad;};
```

## PRIMER 2

```
class Profesor extends Radnik  
( extent radnici)  
{attribute enum Zvanje{redovni, vanredni, docent} zvanje;  
relationship set<Kurs>predaje inverse Kurs:: drži_pred;};
```

```
interface Slušalac  
{struct Adresa {string grad, string ulica_i_ broj};  
  attribute string ime;  
  attribute string broj_indeksa;  
  attribute Adresa adresa-stud;  
  relationship set <Kurs> sluša inverse Kurs :: slušaju;  
  boolean upisan_za_kurs (in unsigned short Kurs)  
  raises (nema_preduslove, popunjeno);  
  void ispiši(in unsigned short Kurs) raises (nije_upisan);  
};
```

## PRIMER 2

**class** Asistent **extends** Radnik: Slušalac

{ **attribute string** ime;

**attribute string** broj\_indeksa;

**attribute** Adresa adresa-stud;

**relationship set** <Kurs> sluša **inverse** Kurs :: slušaju;

**relationship set** <Kurs> vežba **inverse** Kurs :: drži\_vezb;};

**class** Student: Slušalac (**extent** studenti)

{**attribute string** ime;

**attribute string** broj\_indeksa;

**attribute** Adresa adresa-stud;

**relationship set** <Kurs> sluša **inverse** Kurs :: slušaju;};



# **UML DIJAGRAM KLASA KAO OBJEKTNI MODEL**

# KLASE, INTERFEJS I TIPOVI U UML-U

UML uglavnom poštuje ovaj objektni model. U njemu se definiše:

- *"Klasa kao opis skupa objekata koji imaju iste attribute, operacije veze i semantiku"*
- *"Interfejs kao skup operacija koji definiše neki servis klase ili softverske komponente."* Jedna klasa može da ima više servisa, odnosno interfejsa. Drugim rečima može se reći da klasa implementira odgovarajući interfejs i za takav prikaz se definiše posebna vrsta veze u UML-u



## PREDSTAVLJANJE TIPRA OBJEKTA U UML-u

Za tip se ne uvodi poseban koncept već se **tip definiše kao *stereotip klase***.

**Stereotip je** mehanizam preko koga se može proširiti skup koncepata UML-a tako što se izvode novi koncepti iz prethodno definisanih. Stereotip mogu da definišu korisnici, da bi uveli koncepte pogodne za opis njihovog problema.

Neki stereotipovi su definisani i u samom UML-u



## TIP

<b>NAZIV</b> <b>&lt;&lt;tip&gt;&gt;</b>
<b>ATRIBUTI</b>
<b>OPERACIJE</b>

## KLASA

<b>NAZIV</b>
<b>ATRIBUTI</b>
<b>OPERACIJE</b>

## INTERFEJS

<b>NAZIV</b> <b>&lt;&lt;interface&gt;&gt;</b>
<b>OPERACIJE</b>

# SINTAKSA ZA SPECIFIKACIJU KLAZE

## NAZIV KLAZE

**vidljivost nazivi-atributa- 1:tip-podatka-1=pocetrna vrednost {iskaz  
osobina}**

**vidljivost naziv-atributa-2: tip-podatka-2=pocetna vrednost-2 {iskaz  
osobina}**

.....

**vidljivost naziv-operacije-1 (lista-argumenata-1): tip-rezultata-1 {iskaz  
osobina}**

**vidljivost naziv-operacije-2 (lista-argumenata-1): tip-rezultata-2 {iskaz  
osobina}**

.....

# SINTAKSA ZA DEFINISANJE ATRIBUTA I OPERACIJA

## ARIBUTI:

[vidljivost] ime [kardinalnost] [:tip] [= poc.-vredn]  
[ {niz-karaktera} ]

## OPERACIJE:

[vidljivost] ime [ {lista-paraetara} ] [:povratni-tip]  
[ {niz-karaktera} ]

### **Vidljivost može da bude:**

- + **javna** (atribut, odnosno operacija su dostupni svim objektima)
- # **zaštićena** ( atribut, odnosno operacija su dostupni samo nekim objektima)
- **privatna** (atribut i operacija su dostupni samo posmatranoj klasi, odnosno njenim objektima)

## VEZE U UML-U

**ZAVISNOST:** promena u jednom "predmetu" može da utiče na sematiku drugog

----->

**ASOCIJACIJA:** opisuje vezu između pojavljivanja objekata, odnosno predstavlja "klasu" veza između pojavljivanja objekata

0..1

poslodavac

\*

radnik

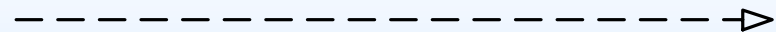
## VEZE U UML-U

### **GENERALIZACIJA:**

Veza između dva objekta u kojoj je prvi objekat generalizacija (nadtip) drugog, odnosno drugi specijalizacija (podtip) prvog. Podtip nasleđuje stanje i ponašanje nadtipa.

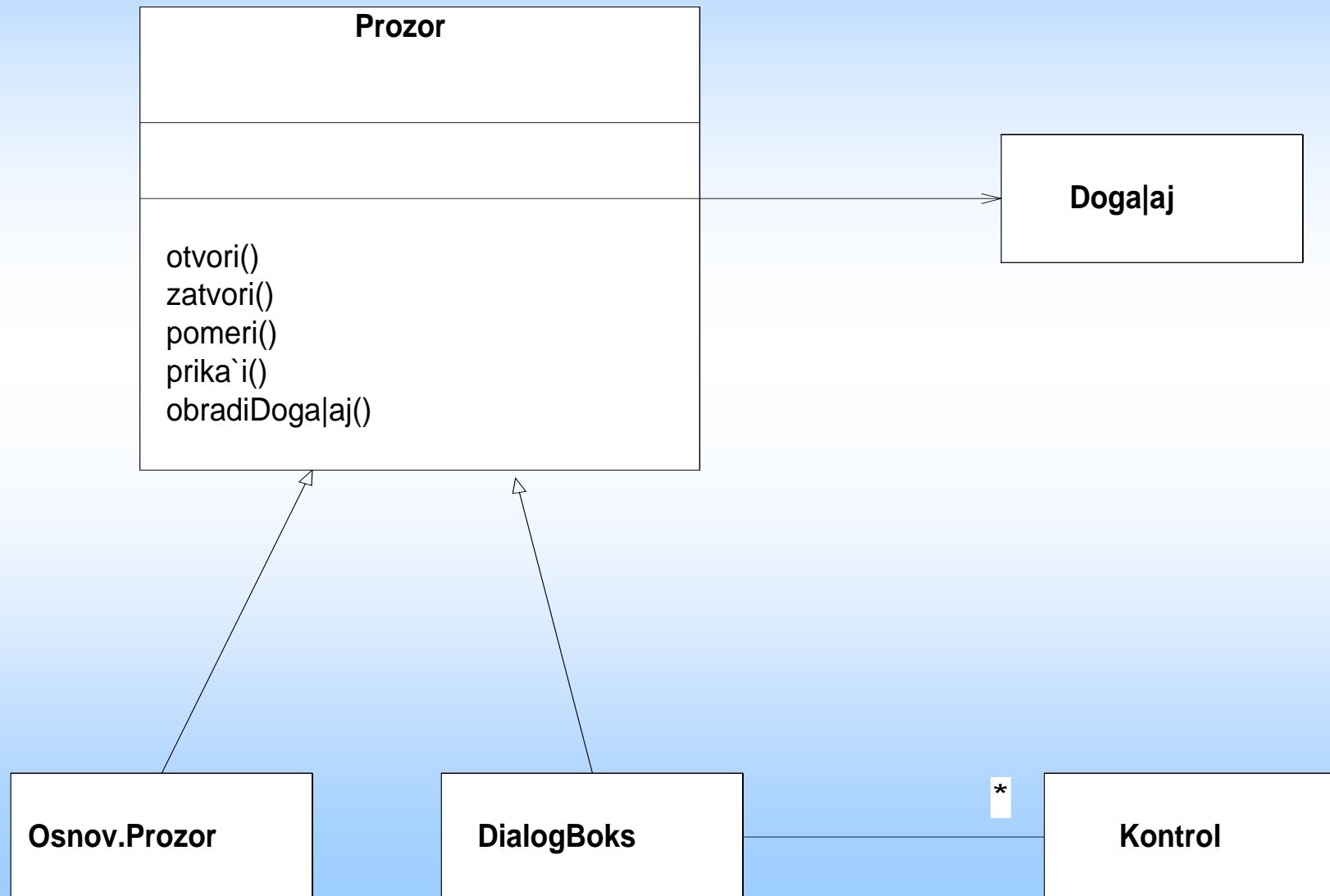


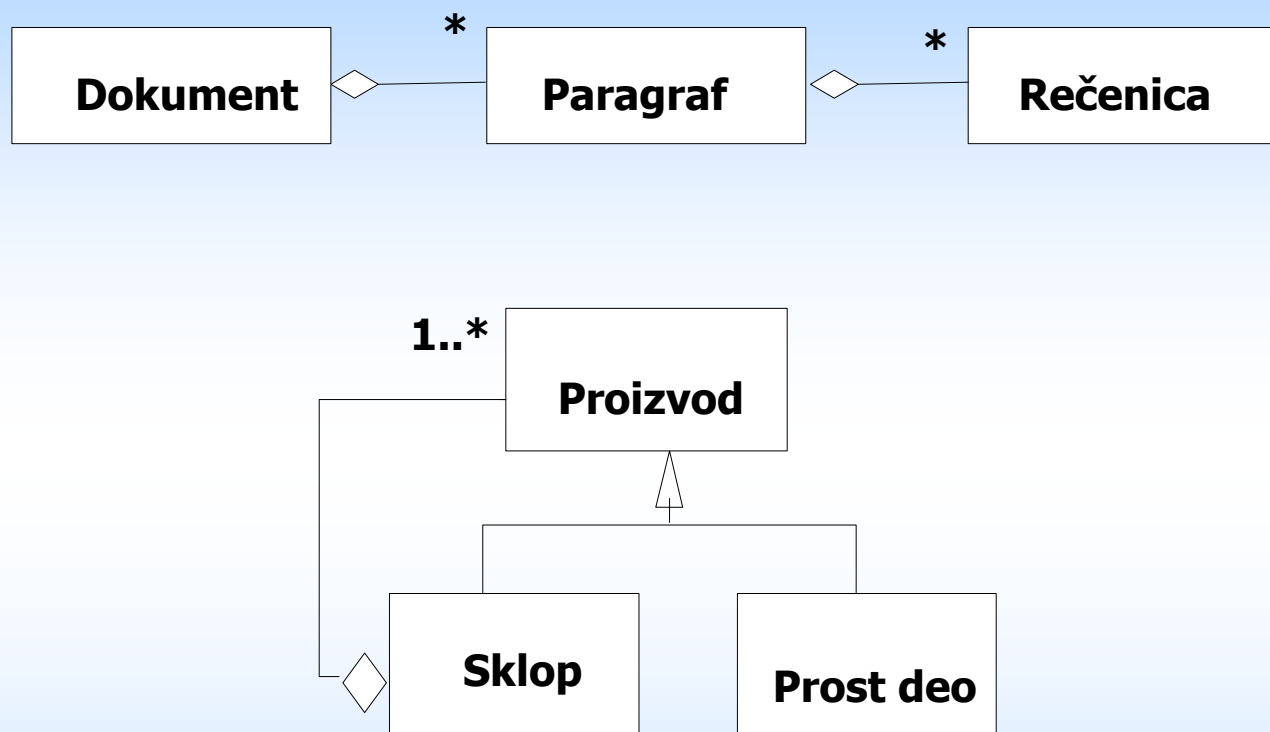
**REALIZACIJA:** veza između specifikacije i implementacije nekog predmeta (klasifikatora)





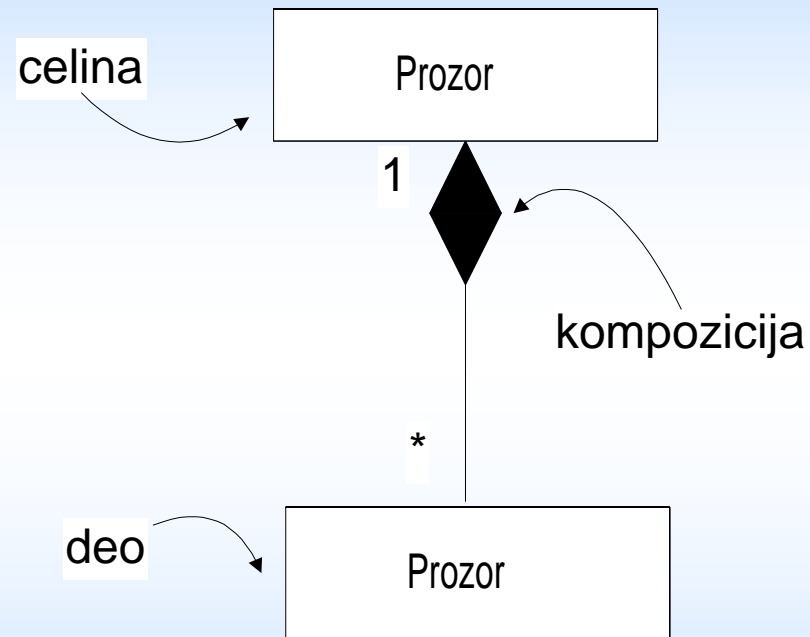
# PRIMERI VEZA





**PRIMERI AGREGACIJE: specifična asocijacija, sa smislom "deo od"**

# KOMPOZICIJA – POSEBNA VRSTA AGREGACIJE



Jedan objekat u jednom trenutku vremena može biti deo samo jedne kompozicije

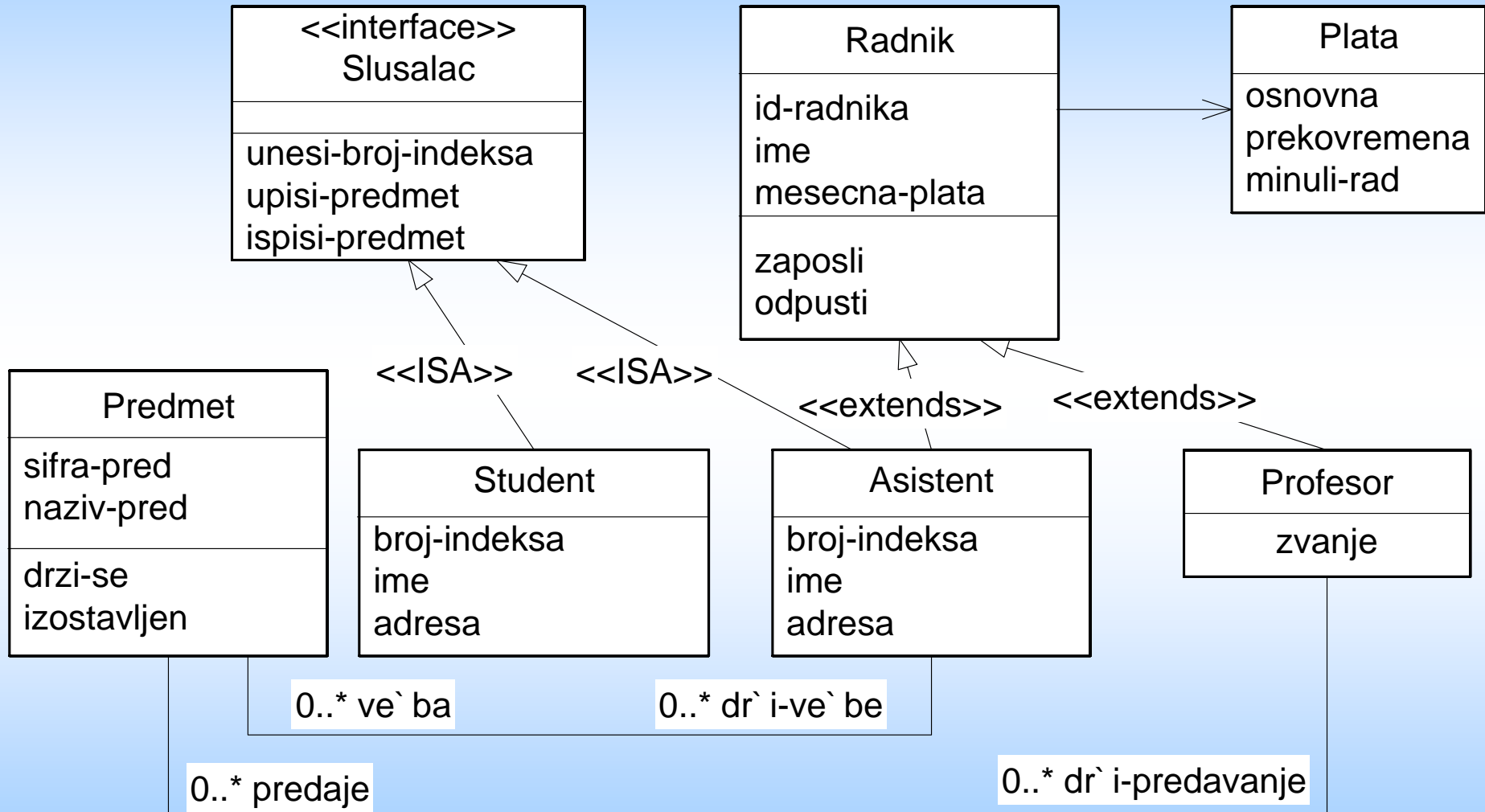
# GENERALIZACIJA

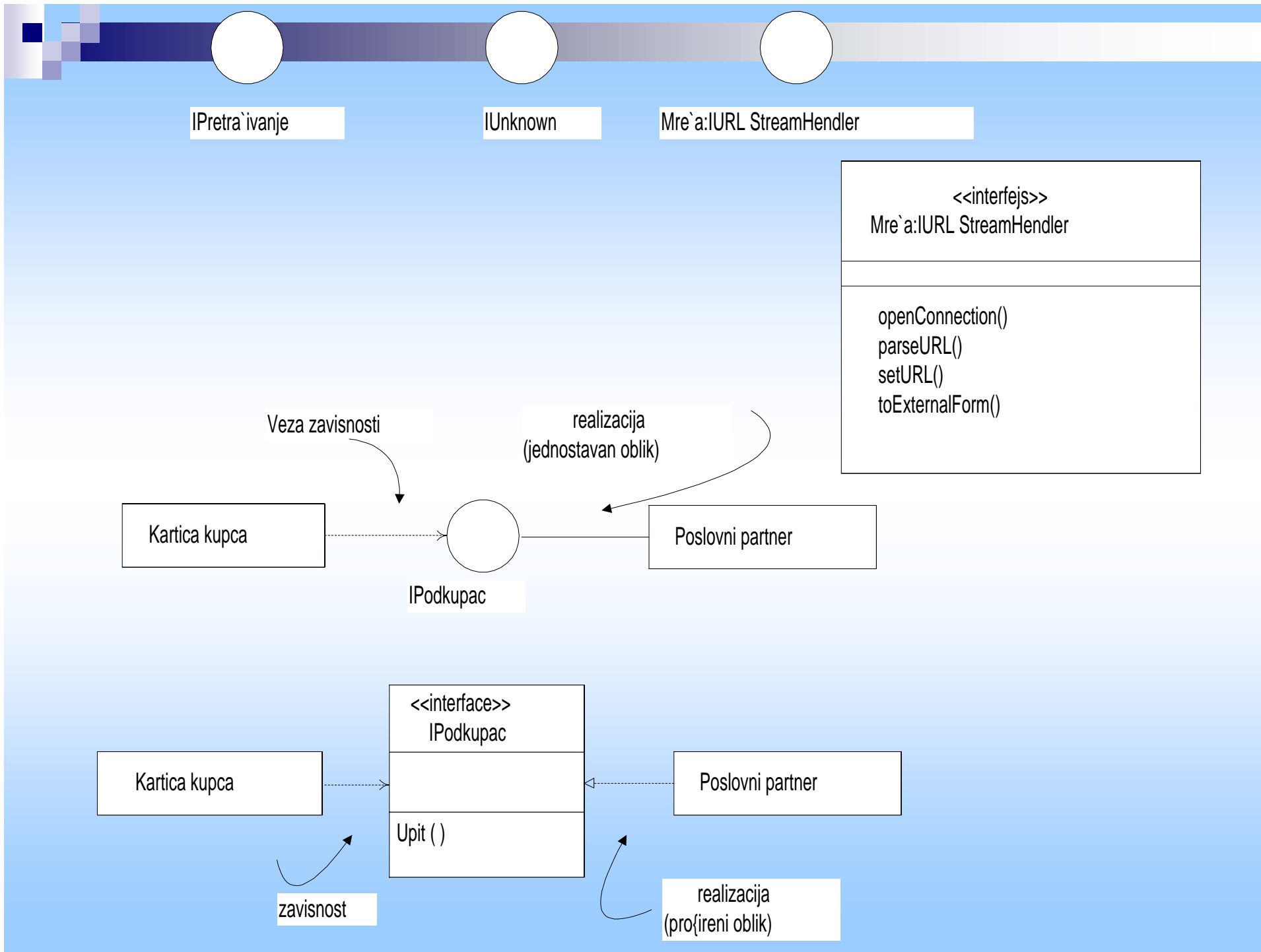




# INTERFEJS

- **Interfejs** je kolekcija operacija koje opisuju "servise" koje pruža klasa ili komponenta nekog sistema.
- **Interfejs** predstavlja liniju razgraničenja između specifikacije i implementacije neke apstrakcije.
- **Interfejs** se koristi da prikaže i dokumentuje granice između podsistema nekog sistema
- Deklarišući **interfejs** definiše se željeno ponašanje neke apstrakcije, nezavisno od načina njene implementacije



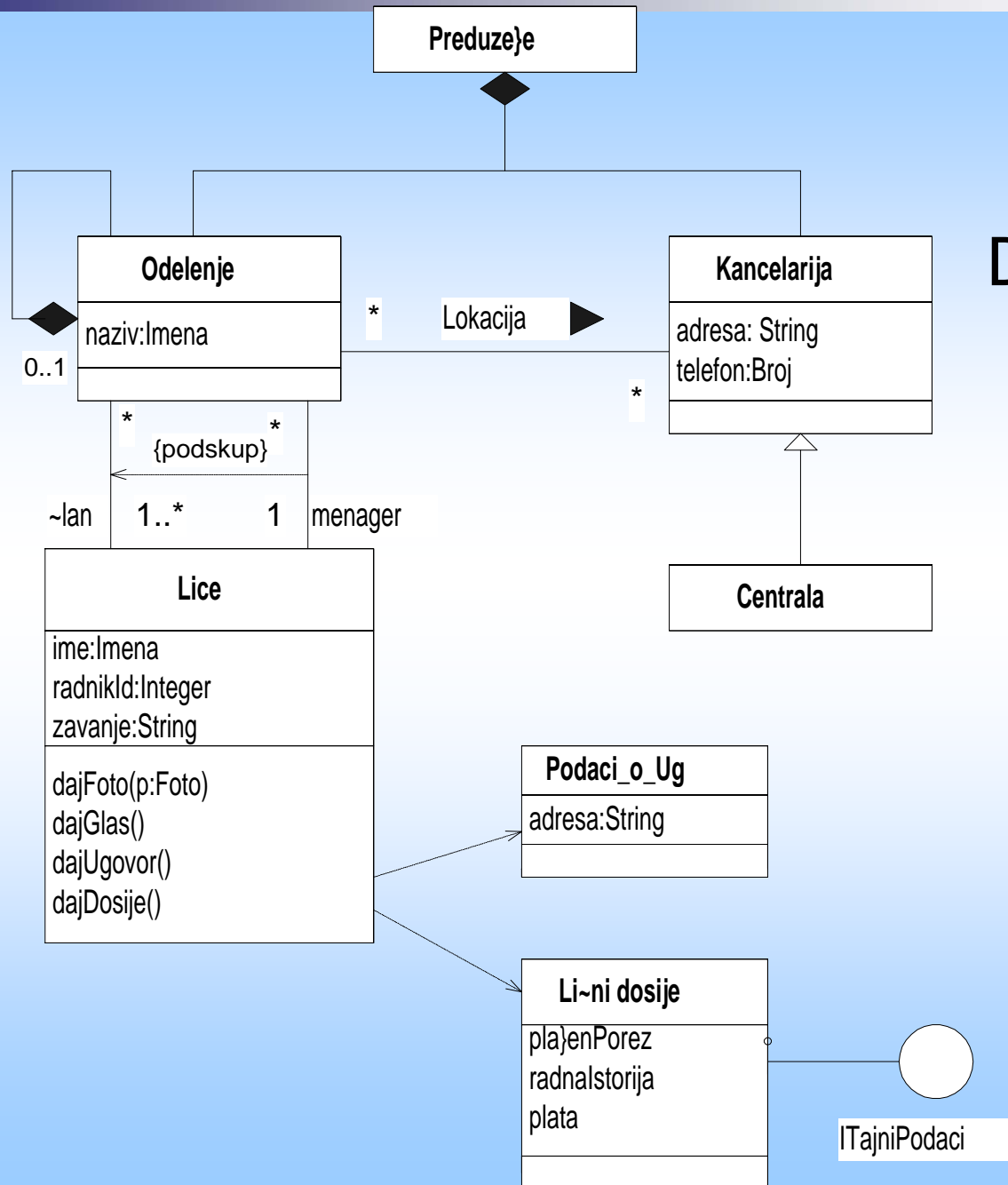


## DIJAGRAMI KLASA

- Dijagram klasa najčešće korišćeni dijagram u OO pristupima razvoju softvera
- On predstavlja skupove klasa, interfejsa, kolaboracija i njihove međusobne veze
- Koristi se da predstavi:
  1. *Osnovni "rečnik sistema" – definiše pojmove koji se u tom sistemu koriste,*
  2. *Opiše strukturu neke kolaboracije,*
  3. *Predstavi logičku šemu baze podataka*



# PRIMER DIJAGRAMA KLASA



# PRIMER DIJAGRAMA KLASA KAO LOGIČKE ŠEME BAZE PODATAKA

