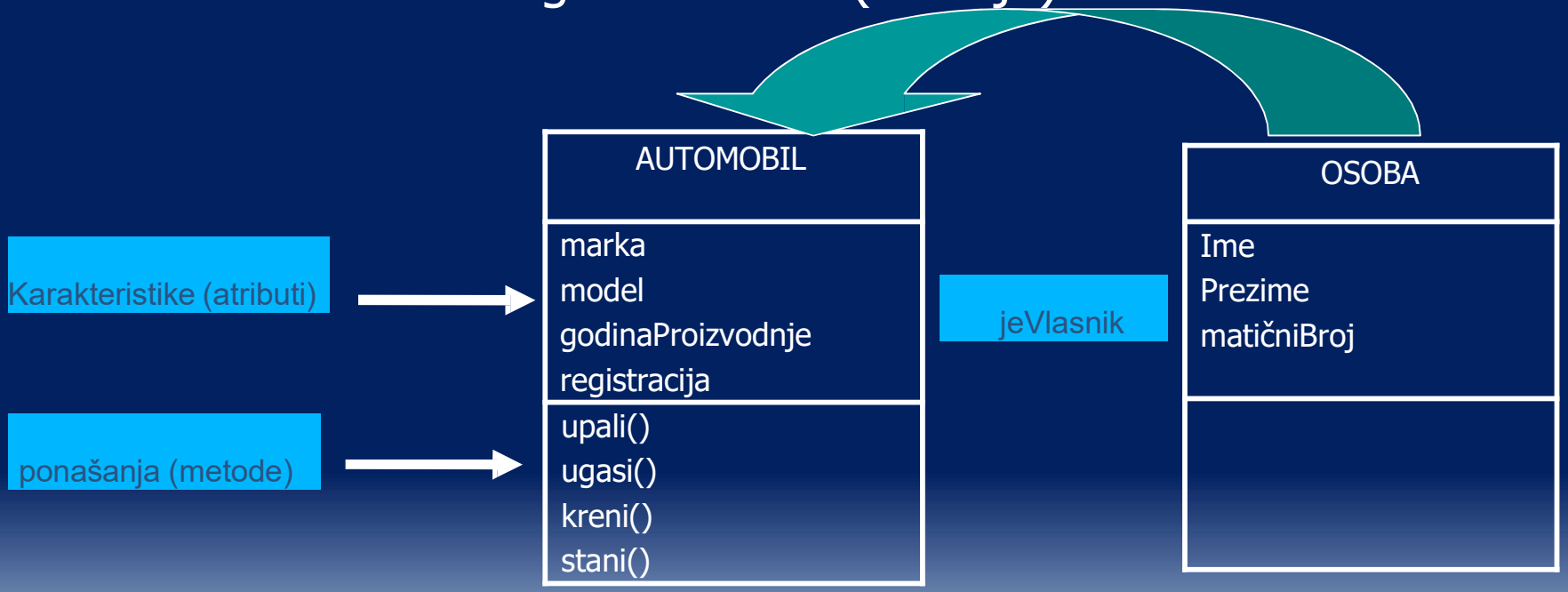
Objektno-orijentisano programiranje:

Relacije

[Prilagođeno od prof. dr Gordana Đorđević]

Klase i njeni elementi

- Klasa obuhvata:
  - karakteristike (atribute)
  - ponašanje (metode)
  - odnose sa drugim klasama (relacije)



Relacije

- Relacije
  - Asocijacija
    - Kompozicija – dekompozicija
    - Generalizacija - specijalizacija
  - Relacija korišćenja

- Asocijacija i njeni specijalizovani oblici (kompozicija – dekompozicija i generalizacija – specijalizacija) se odnose na strukturu jer uspostavljaju strukturne odnose između klasa, a relacija korišćenja se odnosi na ponašanje.

Asocijacija

- je najopštiji oblik relacije u kome objekti jedne klase imaju neku strukturnu vezu ili odnos sa objektima druge klase.

- Svaka asocijacija definiše se preko tri elementa:
  - Kardinalnost
  - Navigacija (smer)
  - Naziv (uloga)

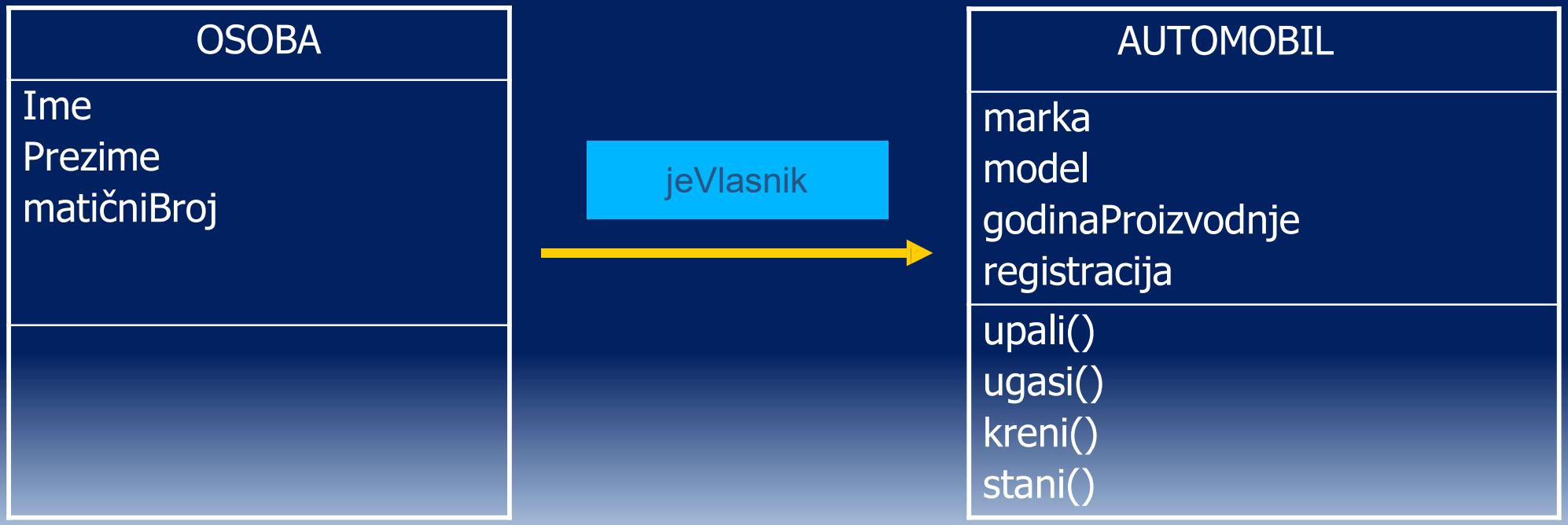
Kardinalnost

- broj koji označava koliko objekata jedne klase može biti u realciji sa objektom druge klase
  - Jedna porudžbina mora imati tačno jednog kupca
  - Firma mora imati makar jednog zaposlenog
- Kardinalnost ne mora da bude jedan broj, može da bude i raspon, tada postoji donja i gornja granica kardinalnosti

Kardinalnost	Primjer
1..1 ili samo 1	Jedna porudžbina mora imati tačno jednog kupca (donja granica je 1 i gornja granica je 1)
0..1	Student u biblioteci može iznajmiti jednu knjigu, ali i ne mora (donja granica je 0 i gornja granica je 1)
0..* ili samo *	Osoba može biti vlasnik jednog ili više automobila, ali ne mora posjedovati automobil (donja granica je 0 i gornja granica je "više")
1.. *	Firma mora imati bar jednog zaposlenog, ali ih može imati i više (donja granica je 1 i gornja granica je "više")

Navigacija (smjer)

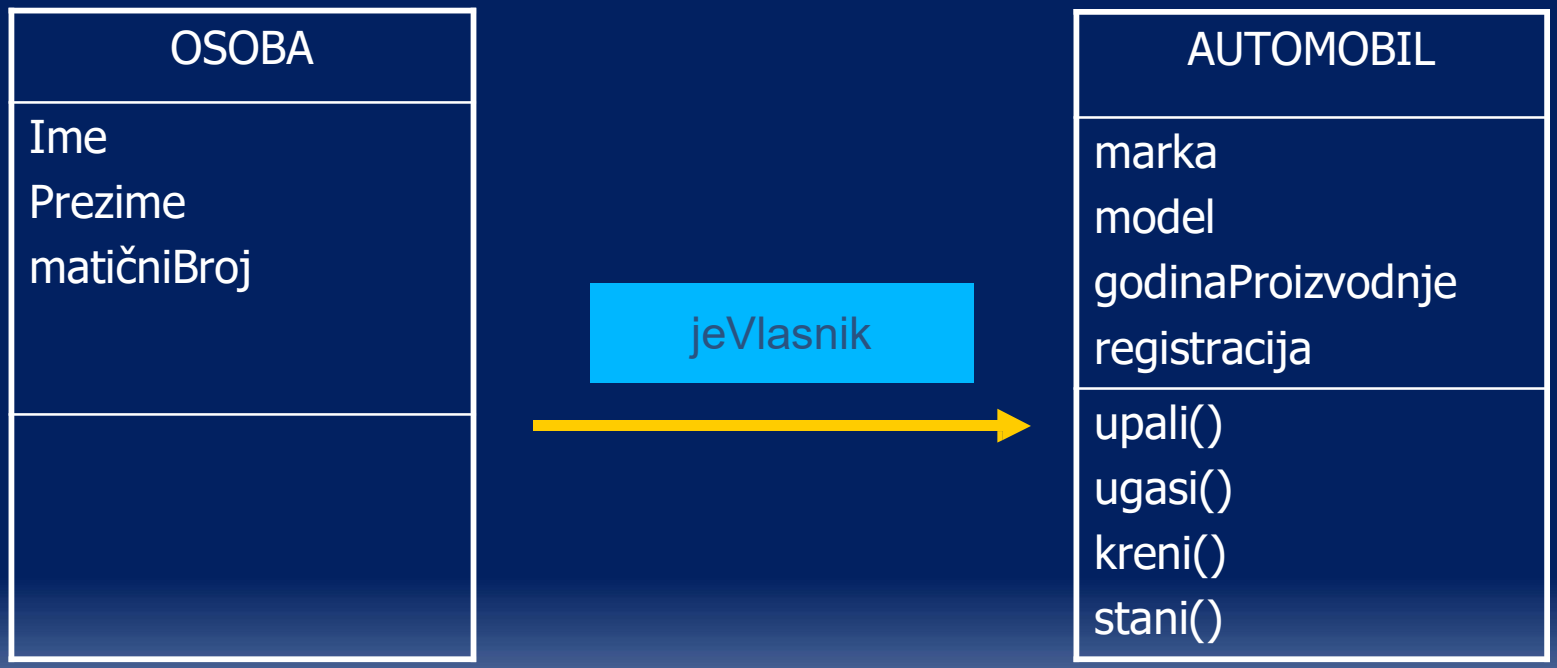
- govori o tome kako je asocijacija usmjerena: od koje klase ka kojoj klasi
- Asocijacije se dijele na jednosmjerne i dvosmjerne pri čemu se dvosmjerne asocijacije mogu posmatrati u oba smjera



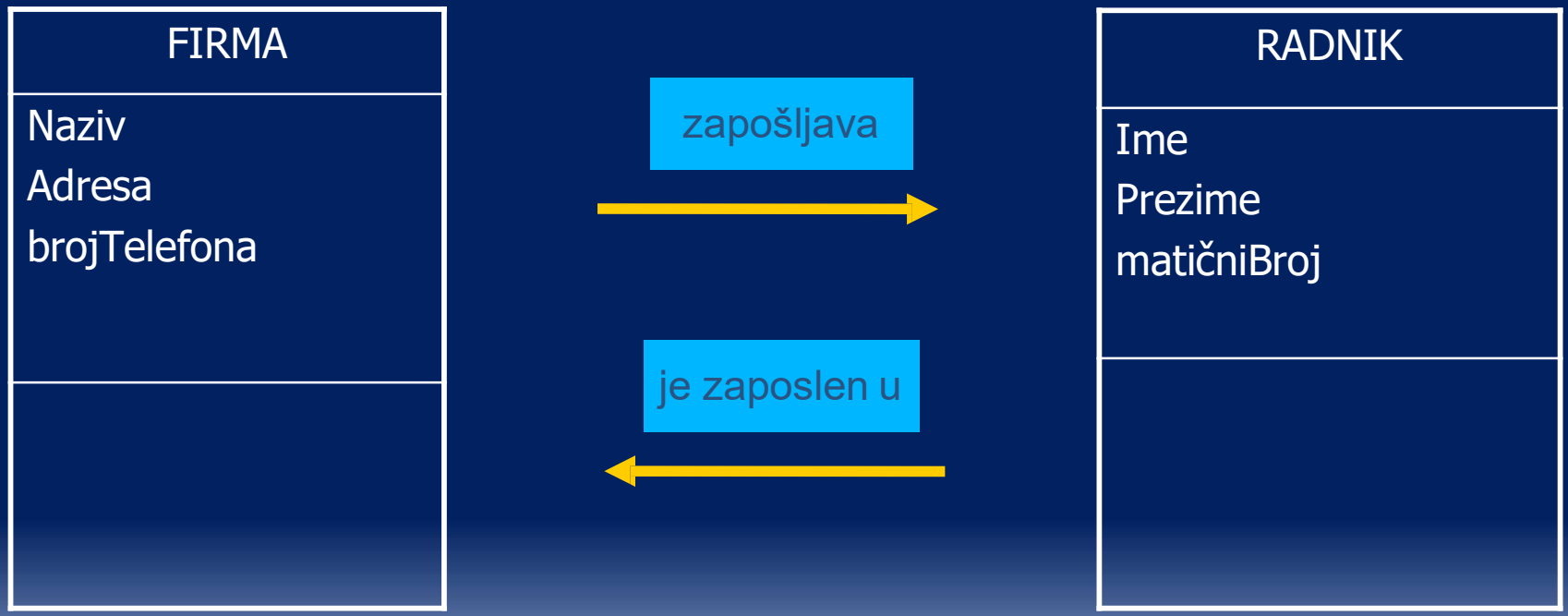
Naziv (uloga)

- Svaka asocijacija može da ima i naziv (ulogu) koji opisuje odnos koji asocijacija predstavlja.
- Jednosmjerne asocijacije mogu da imaju samo jedan naziv, dok dvosmjerne asocijacije mogu imati dva naziva, zavisno od smjera posmatranja

Jednosmjerna asocijacija



Dvosmjerna asocijacija



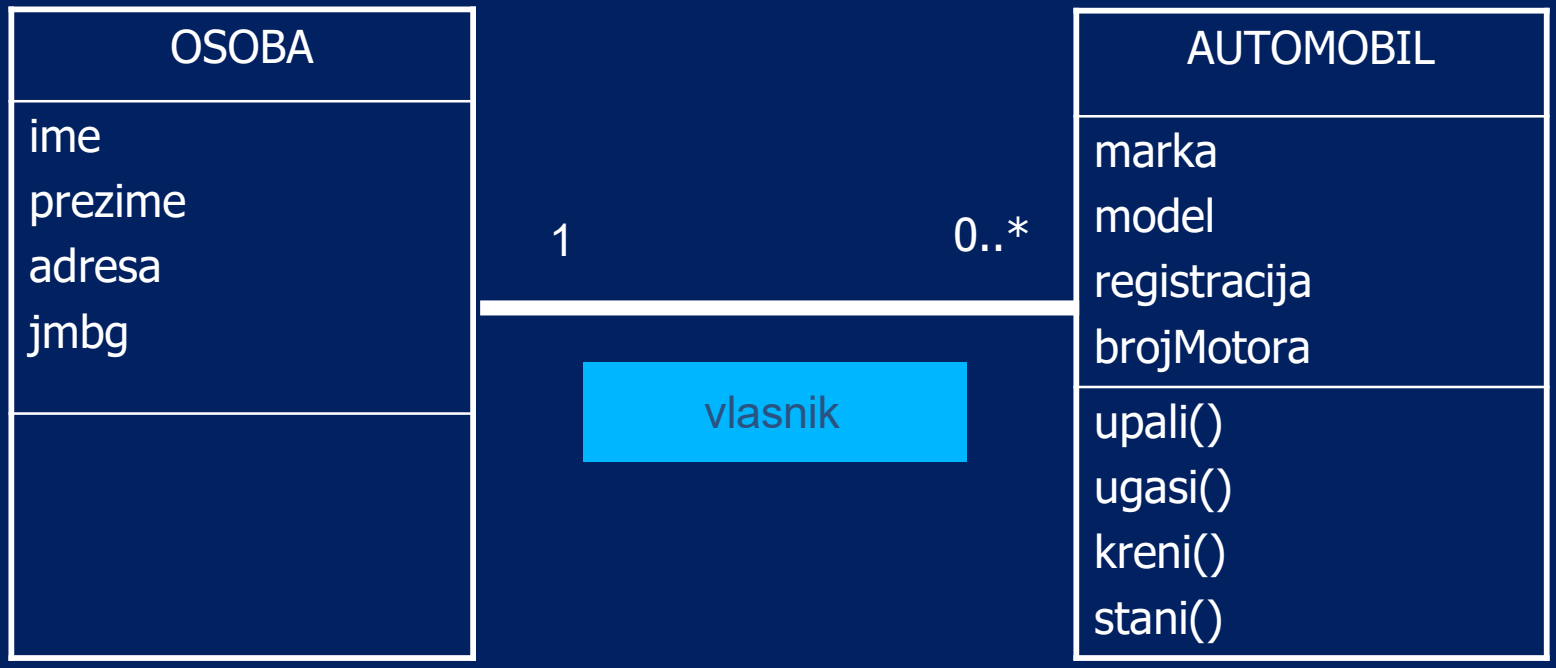
Asocijacije u JAVI

- U JAVI asocijacije se najčešće implemetiraju kao obični **atributi** klase, pri čemu je **tip** atributa sama klasa (navodi se naziv klase)

```
nazivKlase nazivRelacije
```

- Implementacija asocijacije zavisi od kardinalnosti (asocijacije "više prema više" se mogu implementirati uvođenjem nove klase)
- Asocijacije se grafički predstavljaju punom linijom između dve klase na kojoj može biti napisana kardinalnost i naziv asocijacije
- Ako je asocijacija jednosmerna ova linija može na kraju imati strelicu koja upućuje na smer asocijacije, dok se kod dvosmjernih asocijacija navigacione strelice najčešće izostavljaju

Primer asocijacije




```
Class Osoba {
      String ime;
      String prezime;
      String adresa;
      String jmbg;
}
```

```
Class Automobil {
      String marka;
      String model;
      String
      registracija;
      int brojMotora;
      Osoba vlasnik;
}
```

Primjer: pozivanje objekata preko asocijacije

- Napisati klasu TestAutomobil koja kreira jedan objekat klase Automobil marke "Ford", model "Focus", registracije "BG123-456" i broj motora "123456". Postaviti da vlasnik ovog automobila bude Pera Perić, JMBG 2112980710018 koji živi u Resavskoj 40.

Napomena: Pozivanje objekata preko asocijacije vrši se na sličan način kao i pozivanje običnog atributa, sa tim što je objekat potrebno inicijalizovati pre prvog korišćenja.

```
Class Osoba {
       String ime;
          String prezime;
          String adresa;  String jmbg;
}

Class Automobil {
          String marka;  String model;
          String registracija;
          int brojMotora;  Osoba vlasnik;
}
Class TestAutomobil {
       public static void main (String [ ] args) {  Automobil a = new
       Automobil ();

            a.marka = "Ford"  a.model = "Focus";
       a.registracija = "BG123-456";  a.brojMotora =
         123456;

            a.vlasnik = new Osoba ();

          a.vlasnik.ime = "Pera ";  a.vlasnik.prezime = "Perić";
          a.vlasnik.jmbg = "2112980710018";  a.vlasnik.adresa =
          "Resavska 40";
       }
}
```

Primer: pozivanje objekata preko asocijacije

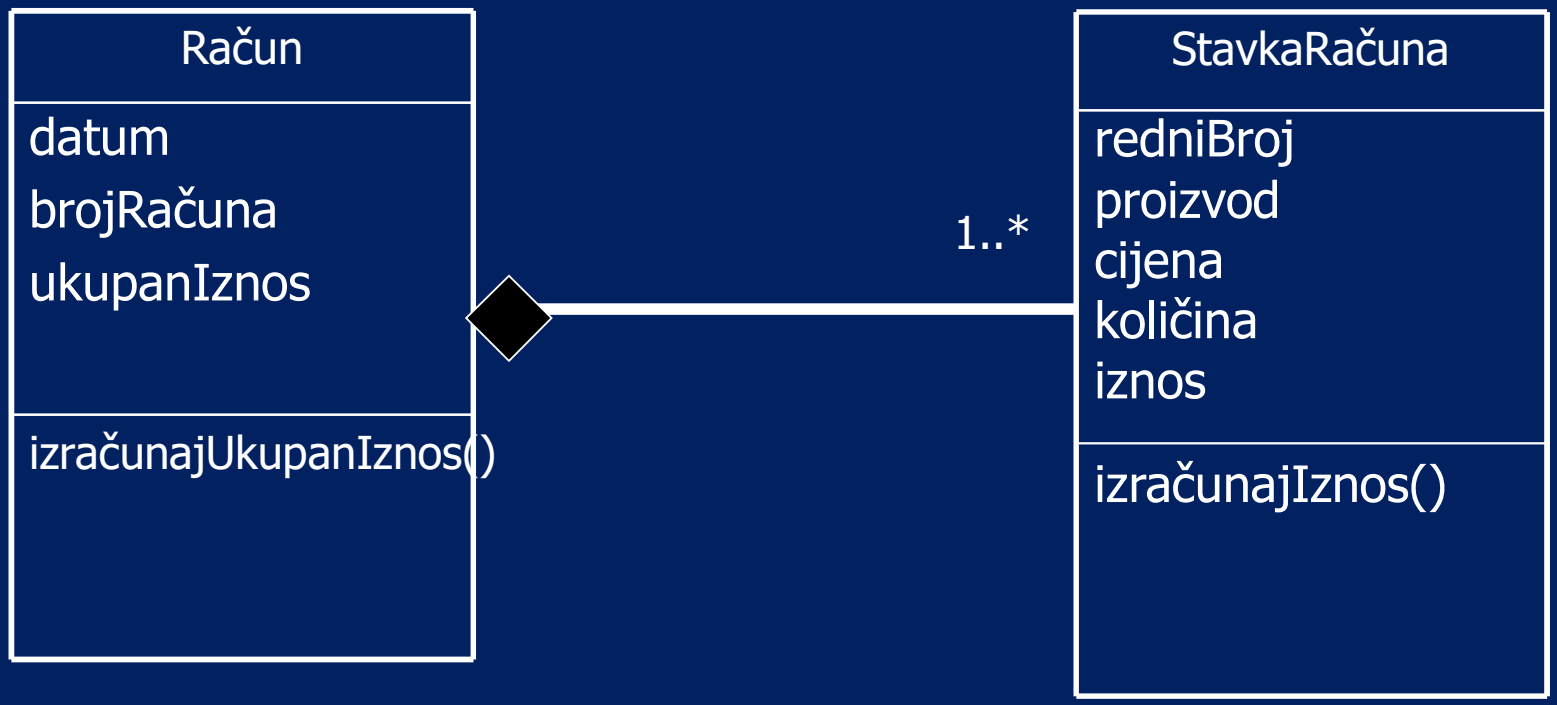
```
Class Osoba {
     String ime;
        String prezime;
        String adresa;  String jmbg;
}

Class TestAutomobil {

     public static void main (String [ ] args) {

         Automobil a = new Automobil ();
         a.marka = "Ford"  a.model = "Focus";
         a.registracija = "BG123-456";
         a.brojMotora = 123456;
         a.vlasnik = new Osoba ();

         a.vlasnik.ime = "Pera ";
         a.vlasnik.prezime = "Perić";
         a.vlasnik.jmbg = "2112980710018";
         a.vlasnik.adresa = "Resavska 40";
      }
}
```

```
Class Automobil {
        String marka;
        String model;
        String registracija;
        int brojMotora;
        Osoba vlasnik;
}
```

Kompozicija - dekompozicija

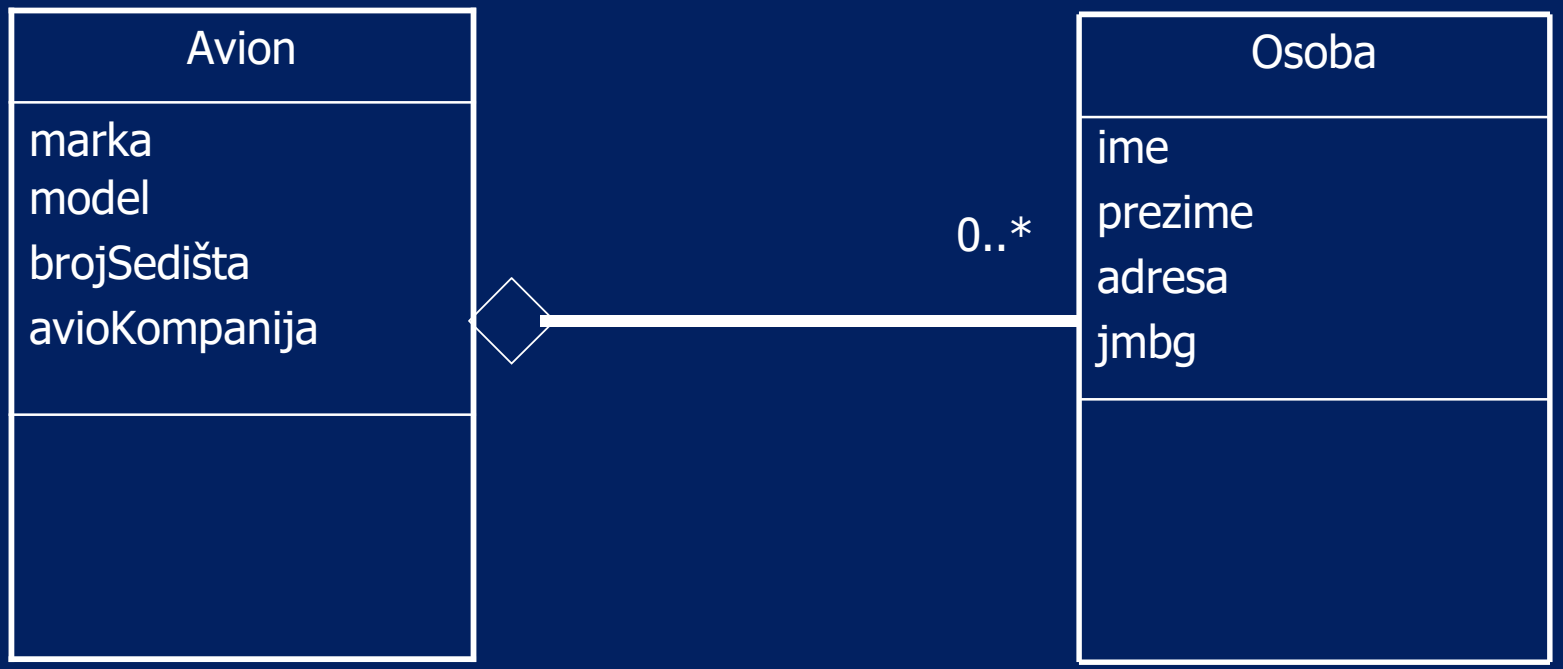
- Predstavlja posebnu vrstu asocijacije u kojoj objekat jedne klase, kao sastavni dio, sadrži jedan ili više objekata druge klase
- Postoje dvije vrste relacija kompozicija – dekompozicija:
  - kompozicija - ako objekat (ili više njih) koji je sadržan ne može da postoji nezavisno od objekta koji ga sadrži
  - Agregacija
- Kod kompozicije, kada se obriše objekat koji sadrži druge objekte, brišu se svi objekti; kod agregacije to nije slučaj
- Relacija kompozicija – dekompozicija se grafički predstavlja linijom koja na jednom kraju ima romb. Ako je romb ispunjen u pitanju je kompozicija, a ako je prazan u pitanju je agregacija
- Kompozicija – dekompozicija može da ima naziv, a kardinalnost se piše samo sa jedne strane jer je sa druge uvijek 1.
- Kompozicija – dekompozicija se implementira na isti način kao i obična asocijacija – preko atributa klase; jedina razlika je u načinu na koji se tumači asocijacija

Primer Kompozicije



- kada se obriše objekat koji sadrži druge objekte, brišu se svi objekti;

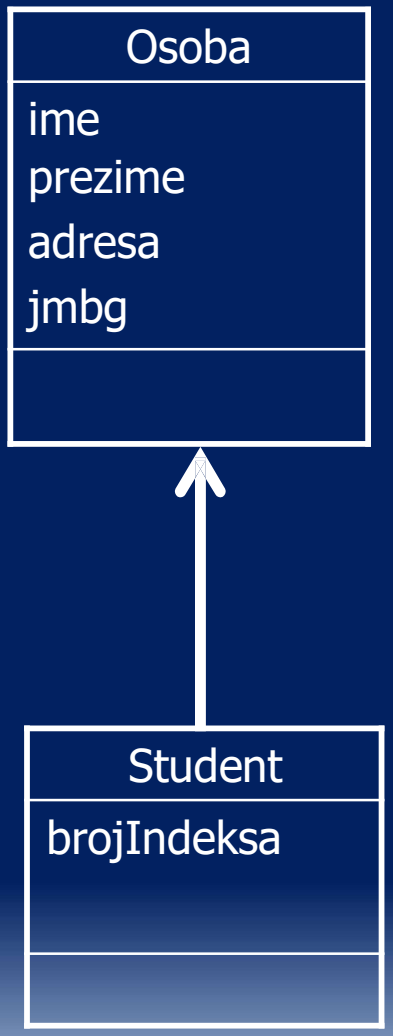
Primer Agregacije ◊



Generalizacija - specijalizacija

- vrsta relacije u kojoj jednu klasu "nasljeđuje" druga klasa, pa se zato često naziva **nasljeđivanje**.
- Nasleđivanje podrazumijeva preuzimanje svih atributa i javnih metoda nadklase (klasa koja je naslijeđena) i njihovo prenošenje u podklasu (klasa koja nasljeđuje). Podklasa osim nasljeđenih atributa i javnih metoda može imati i sopstvene karakteristike i ponašanja
- Relacija nasleđivanja se grafički predstavlja linijom na čijem se vrhu nalazi prazna strelica, a kardinalnost se ne navodi

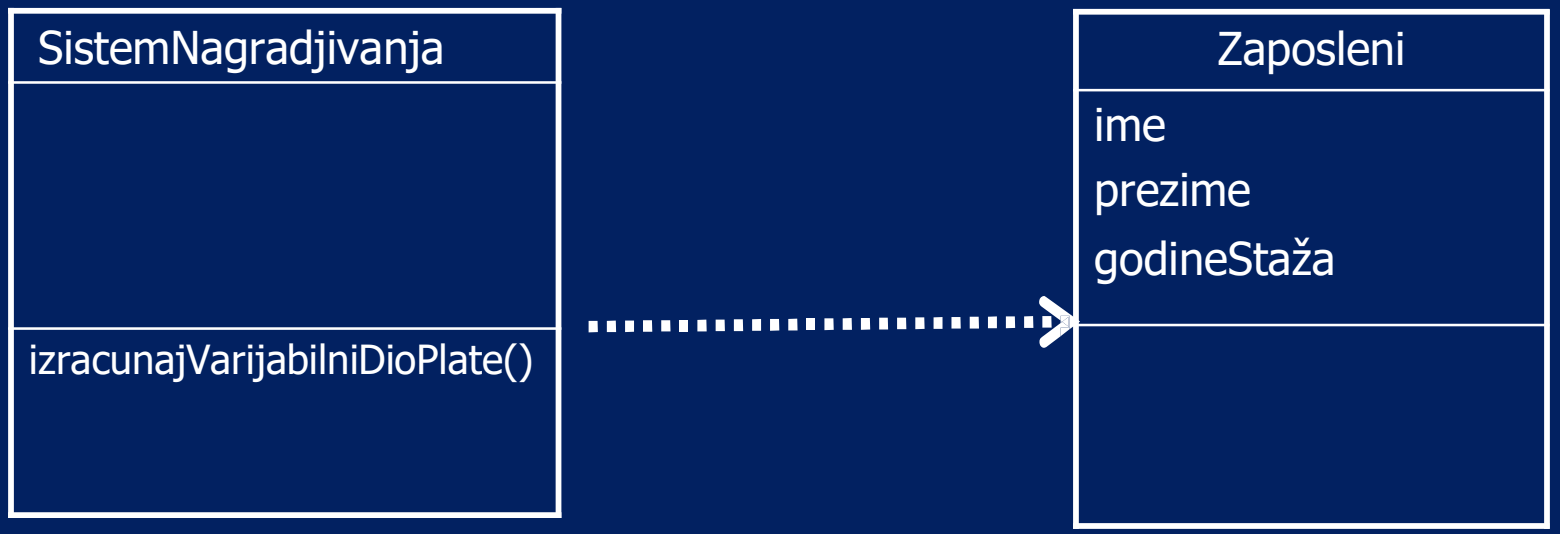
Primer Generalizacija - specijalizacija



Relacija korišćenja

- se javlja kada jedna klasa koristi objekat druge klase u okviru neke svoje metode
- Nije bitno da li je objekat te druge klase parametar metode, povratna vrijednost ili se samo koristi u okviru tijela metode
- Ova relacija obično nema eksplicitno označnenu kardinalnost, niti ime, grafički se predstavlja isprekidanom linijom usmjerenom ka klasi koja se koristi

Primer Relacija korišćenja



Primer Relacija korišćenja

- Napisati klasu Zaposleni koja ima atribute: ime, prezime i godinaStaza.
- Napraviti klasu SistemNagradjivanja koja ima metodu izracunajVariabilniDioPlate. Ova metoda kao parametar prima objekat klase Zaposleni i izračunava i vraća procenat koji predstavlja varijabilni dio plate po formuli: godinaStaza * 1.2

```
Class Zaposleni {
      String ime;
      String prezime;
      int godineStaza;
}
Class SistemNagradjivanja {
     double izracunajVarijabilniDioPlate(Zaposleni z) {
          double var
          var = z.godineStaza * 1.2;
          return var;
     }
}
```

Zadatak

Napraviti klasu Osoba koja ima atribute
- atribut ime
- atribut prezime
- Metodu ispisi koja ispisuje sve podatke o osobi

Napraviti klasu Formula1Tim koja ima
- Aribut naziv
- Atribut menadžer koji je objekat tipa Osoba i predstavlja menadžera trkačkog tima
- Atribut prviVozac koji je objekat tipa Osoba i predstavlja prvog vozača
- Atribut drugiVozac koji je objekat tipa Osoba i predstavlja drugog vozača
- Metodu ispiši koja ispisuje sve podatke o timu, uključujući i imena i prezimena menadžera i oba vozača. Ova metoda bi trebalo da poziva metodu ispiši klase Osoba u cilju ispisivanja podataka o menadžerima i vozačima

Napraviti klasu TestFiormula1Tim koja kreira tim formule 1 “Ferari”, čiji je vlasnik “Berny Eckleston”. Prvi vozač tima je “Kimi Raikkonen”, a drugi “Felipe Massa”. Ispisati na ekranu sve podatke o timu.

```
Class Osoba {
    String ime;
    String prezime;
    void ispisi () {
       System.out.println ("Ime: "+ime);
       System.out.println ("Prezime: "+prezime);
    }
 }
```

```
Class Formula1Tim {
    String naziv;
    Osoba menadzer;
    Osoba prviVozac;
    Osoba drugiVozac;
    void ispisi () {
       System.out.println ("Naziv tima: "+naziv);
       System.out.println ("Menadzer");
       Menadzer.ispisi;
              //Direktno se poziva metoda ispisi klase Osoba
              //da bi se ispisalo ime i prezime menadzera.
       System.out.println ("Prvi vozac");
       prviVozac.ispisi;
              //Direktno se poziva metoda ispisti klase Osoba
              //da bi se ispisalo ime i prezime prvog vozaca.
       System.out.println ("Drugi vozac");
       Menadzer.ispisi;
              //Direktno se poziva metoda ispisti klase Osoba
               //da bi se ispisalo ime i prezime drugog vozaca.
       }
 }
```

```
Class TestFormula1Tim {
    public static void main (String [ ] args) {
      Formula1Tim t = new Formula1Tim ();
      t.naziv = "Scuderia Ferrari Marlboro";
      t.menadzer = new Osoba ();
      t.menadzer.ime = "Berny";
      t.menadzer.prezime = "Eckleston";
      t.prviVozac = new Osoba ();
      t.prviVozac.ime = "Kimi";
      t.prviVozac.prezime = "Raikkonen";
      t.drugiVozac = new Osoba ();
      t.drugiVozac.ime = "Felipe";
      t.drugiVozac.prezime = "Massa";
      t.ispisi ();
    }
}
```