Siniša Nikolić

Sadržaj

- Upoznavanje sa osnovnim konceptima objektno orijentisanog programiranja u Javi,
  - modelovanje
  - atibuti i metode, podrazumevane vrednosti atributa, null, this
  - modifikatori pristupa, enkapsulacija podataka, get/set metode
  - konstruktori
- Nizovi i objekti modelovanih entiteta,
- Klasa Object, metode toString i equals,
- Relacije između objekata
- Immutable Objects

Bez korišćenja koncepta OOP

- Kako bi se rešavao određeni problem? Kako bi uskladištili podatke o određenim entitetima?
- Nizovima?

```
String[][] osoba = new String[2][3]();
osoba[0][0] ="Pera";
osoba[0][1] ="Perić";
osoba[0][2] ="11.06.1990";
osoba[1][0] ="Đura";
osoba[1][1] ="Đurić";
osoba[1][2] ="30.01.1980";
```

- Mnogo prirodnije bi bilo da sve podatke o jednom entitetu grupišemo i postavimo na jednom mestu
- Rešenje bi bilo koristiti Klase

Upoznavanje sa osnovnim konceptima OOP u Javi

- Temin *object-oriented programming* (OOP) predstavlja način razmišljana za rešavanje programerskih problema.
- Srž ovog načina razmišljanja čini koncept objekta.
- Neki ovaj način razmišljanja nazivaju još i *class-oriented programming*, zato što se za opis objekta koriste klase
- U objektno orijentisanom programiranju, u sklopu opisa problema, potrebno je uočiti entitete (jedinice posmatranja) koji se nalaze u svetu (domenu) u kojem se nalazi problem koji se rešava.
- Da bi se podržalo rešavanje određenog problema, potrebno je uočene entitete opisati na nekakav način i navesti operacije nad tim entitetima i njihovim osobinama.

Upoznavanje sa osnovnim konceptima OOP u Javi

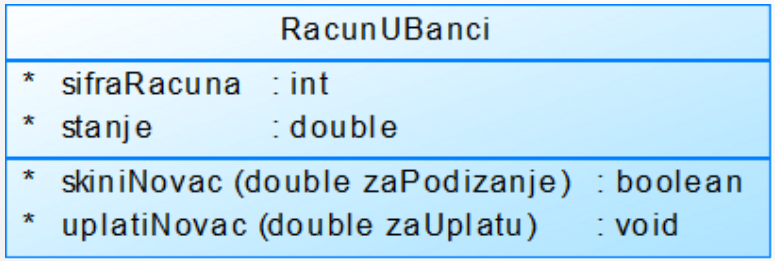
- Objektno orijentisano programiranje se svodi na identifikaciju entiteta u nekom domenu, navođenje njihovih osobina i pisanje operacija nad tim osobinama.
- U objektno orijentisanoj terminologiji, entiteti su opisani klasama, osobine su atributi, a operacije su metode.

Klase

- Klasa predstavlja generalizovani opis grupe entiteta.
- Dobija se procesom apstrakcije čiji je cilj da na osnovu posmatranja konkretnih entiteta definiše zajedničku kategoriju kojom bi se svi posmatrani entiteti da se opišu
- Klase se dizajniraju tako da sadrže sve najbitnije osobine entiteta.
- Klasa predstavlja model objekta i uključuje atribute i metode.
- U Javi je sve predstavljeno klasama, sve što napišemo (metode, promenljive…) mora se naći u okviru neke klasenije tj. moguće definisati funkcije i promenljive izvan neke klase.
- I *main* metoda se nalazi u nekoj klasi!!!
- Listing za definisanje klase počinje ključnom reči **class** .

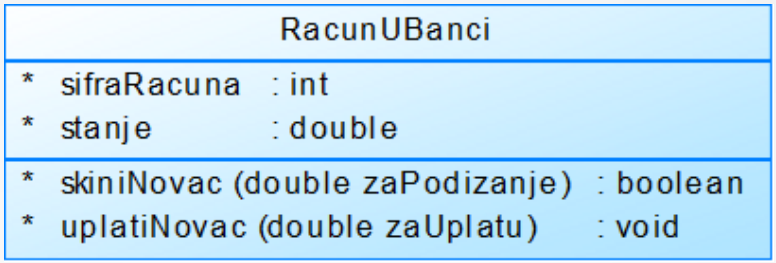
Klase

- Primer klase
  - Za Bankarsku aplikaciju potrebno je modelovati entitet račun.
    - Račun je opisan sa šifrom i stanjem.
    - Dozvoljene operacije nad računom omogućuju skidanje i uplaćivanje novca.
      - Skidanje novca je jedino moguće ako ima dovoljno novca na stanju.
- Na slici je prikazan izgled klase koji je modelovan alatom Power Designer.



Klase

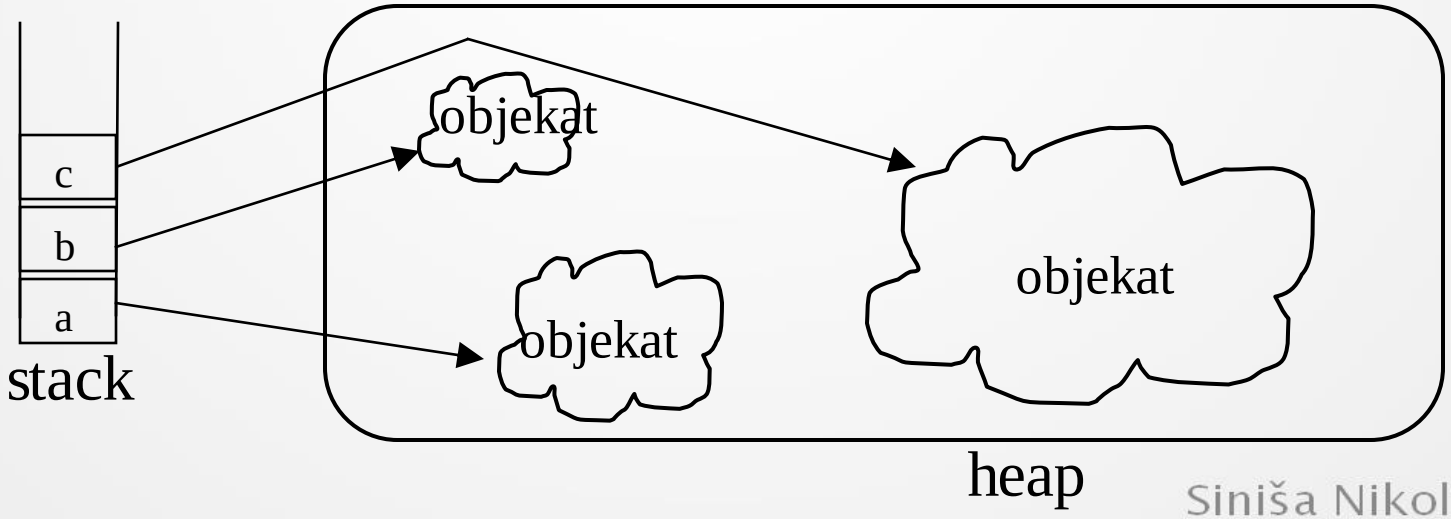
- Primer klase




```
class RacunUBanci {
    int    sifraRacuna;
    double stanje;
    boolean skiniNovac(double zaPodizanje){
        if(stanje-zaPodizanje < 0){
            System.out.println("Nedovoljan saldo");
            return false;
        }
        stanje-=zaPodizanje;
        return true;
    }
    void uplatiNovac(double zaUplatu){
        stanje+=zaUplatu;
    }
}
```

Objekti

- Instance klasa se zovu objekti.
- Objekti se kreiraju upotrebom ključne reči **new**.
- za čuvanje kreiranih objekata koristi se **heap**
- na **heap**-u se zauzima (alocira) memorija za objekat, dok se referenca (oznaka memorijske lokacije) ka objektu čuva kao vrednost promenljive na **stack**-u.



Objekti – Instanciranje Klase

- Primer kreiranje Objekta

```
class Test {
    public static void main(String args[]) {
      RacunUBanci a;
      a = new RacunUBanci(); //instanciranje klase
      //u promenljivoj a se čuva referenca na objekat
      a.sifraRacuna=1; //postavljanje vrednosti atributa
      a.uplatiNovac(999.99); //pozivanje metode
    }
}
```

Podrazumevene vrednosti atributa

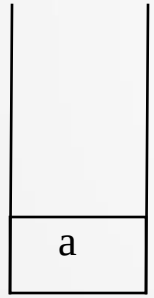
- Atributi klasa imaju podrazumevane vrednosti
- Za atribute primitivnih tipova to su:

Primitivni tip	Default
boolean	false
char	'\u0000'
byte	(byte)0
short	(short)0
int	0
long	0L
float	0.0f
double	0.0d

Rad sa referencama i Heap memorjom

- Na steku se samo deklariše promenljiva a čija je inicijalna vrednost undefined.

```
RacunUBanci a;
```



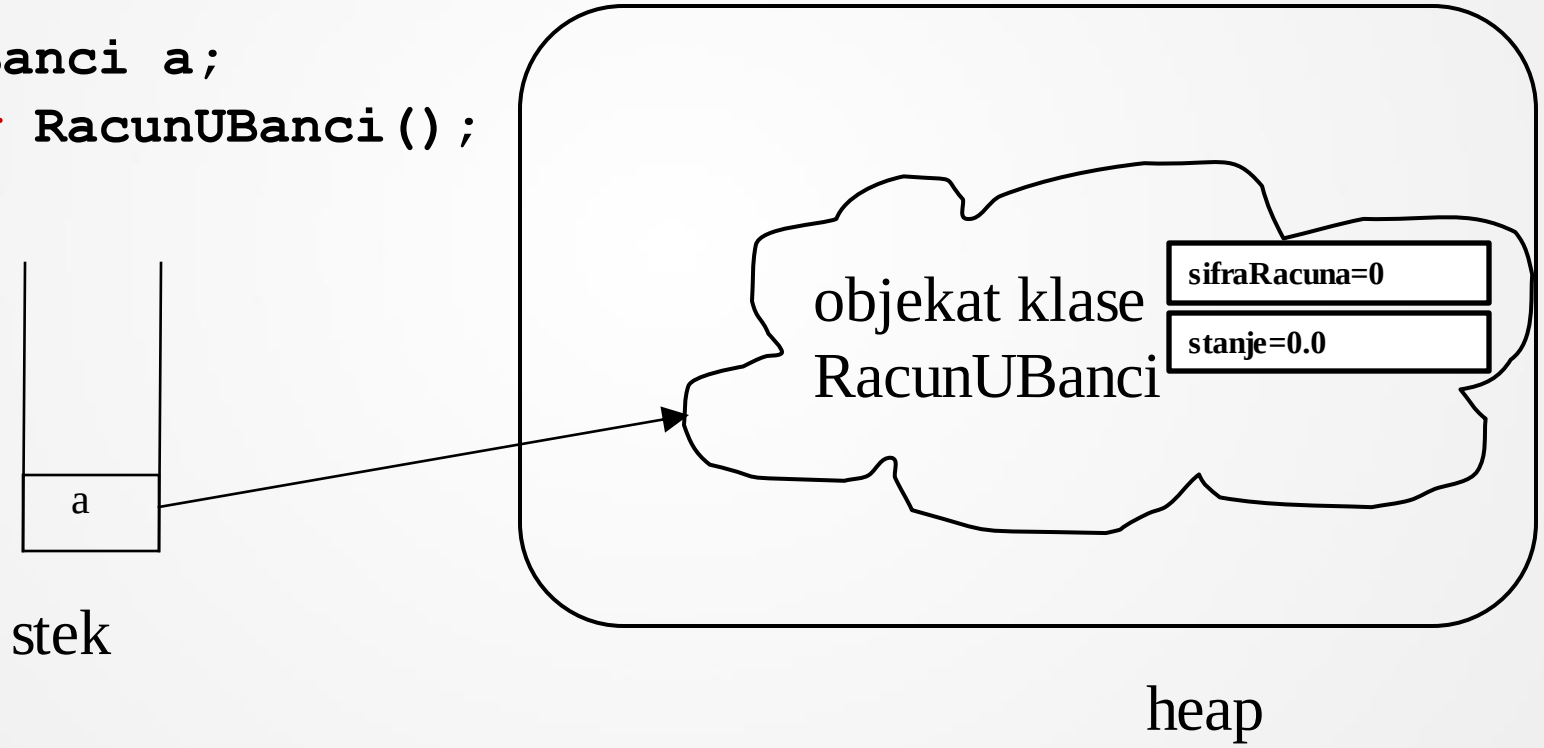
stek

heap

Rad sa referencama i Heap memorjom

- Promenljivoj a se dodeljuje vrednost reference na kreirani objekat na heap-u.
- Lokalna promenljiva **a** nije objekat, već referenca na objekat

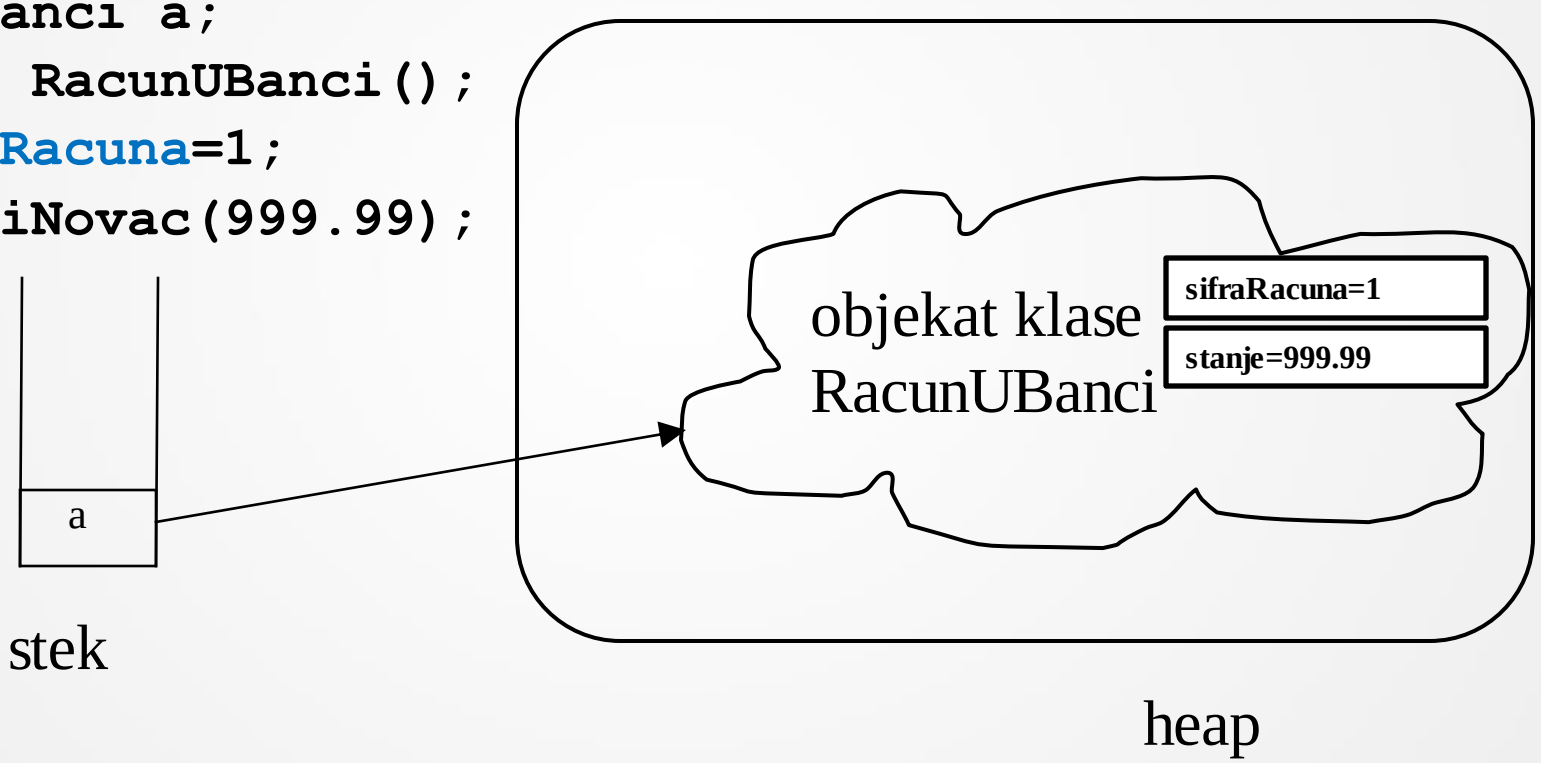
```
RacunUBanci a;
a = new RacunUBanci();
```



Rad sa referencama i Heap memorjom

- Dodela vrednosti određenom atributu objekta i pozivanje metode objekta.

```
RacunUBanci a;
a = new RacunUBanci();
a.sifraRacuna=1;
a.uplatiNovac(999.99);
```

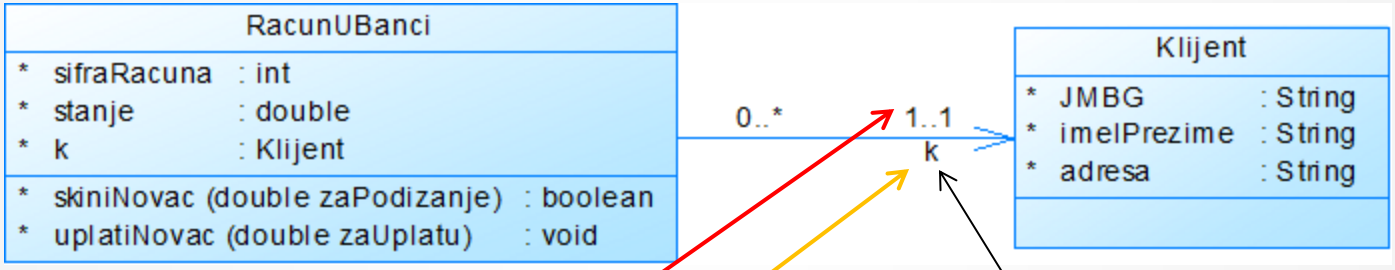


Klase – atibuti koji nisu primitivni tipovi

- Klasa takođe može imati atribut koji nije primitivni tip tj. atribut može sadržati referencu ka nekom objekatu koji je na heap-u.
- Atributi koji nisu primitivni tipovi imaju podrazumevanu vrednost **null** što može izazvati **NullPointerException** grešku u radu aplikacije (runtime exception).
- Smernice za modelovanje
  - Ako u tekstu zadatka identifikujemo varijaciju reči **više** znamo da možemo imati atribut koji će biti Set/Lista/Mapa i koji će sadržati reference ka više objekata.
  - Ako u tekstu zadatka identifikujemo varijaciju reči **jedan** znamo da možemo imati atribut koji će sadržati referencu ka jednom objektu

Klase – atibuti koji nisu primitivni tipovi

- Modifikacija primera Bakarske aplikacije.
  - Proširiti model podataka tako da se obuhvati entitet klijent.
  - Klijent je opisan JMBG, imenom i prezimenom i adresom.
    - Za svaki račun je poznato kojem klijentu on pripada.



Određeni račun u banci mora da ima podatke o određenom klijentu kome on pripada

Klasa račun u banci imaće atribut tipa klase Klijent čija vrednost ne sme biti *null.*

role name *k* koji će postati naziv atributa

Klase – atibuti koji nisu primitivni tipovi

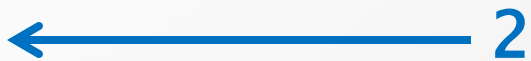
```
class RacunUBanci {
   int    sifraRacuna;
   double stanje;
   Klijent k;
   boolean skiniNovac(double zaPodizanje){
      if(stanje-zaPodizanje < 0){
         System.out.println("Nedovoljan saldo");
         return false;
      }
      stanje-=zaPodizanje;
      return true;
   }
   void uplatiNovac(double zaUplatu){
      stanje+=zaUplatu;
   }
}
```

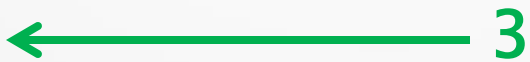
```
class Klijent{
   String JMBG;
   String imeIPrezime;
   String adresa;
}
```

Objekti – atibuti koji nisu primitivni tipovi

- Primer kreiranje Objekta

```
class Test {
    public static void main(String args[]) {
      Klijent kli = new Klijent();
      kli.JMBG="123";
      kli.imeIPrezime="Petar Petrovic";
      kli.adresa="AAA";
      RacunUBanci a;
      a = new RacunUBanci();
      a.sifraRacuna=1;
      a.uplatiNovac(999.99);
      a.k=kli;
      a.k.adresa="BBB";
    }
}
```

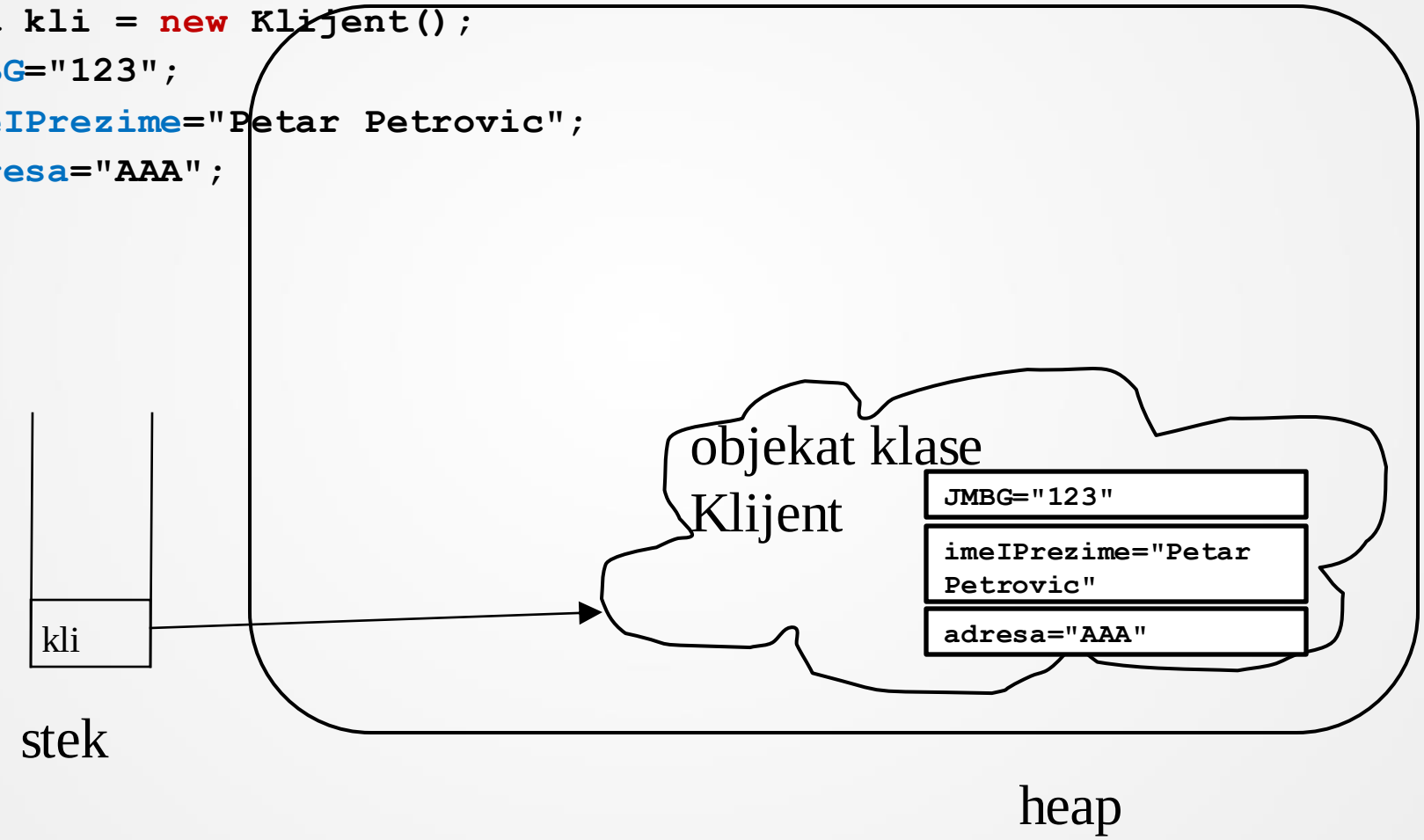




Rad sa referencama i Heap memorjom

- Kod izvršen do 1

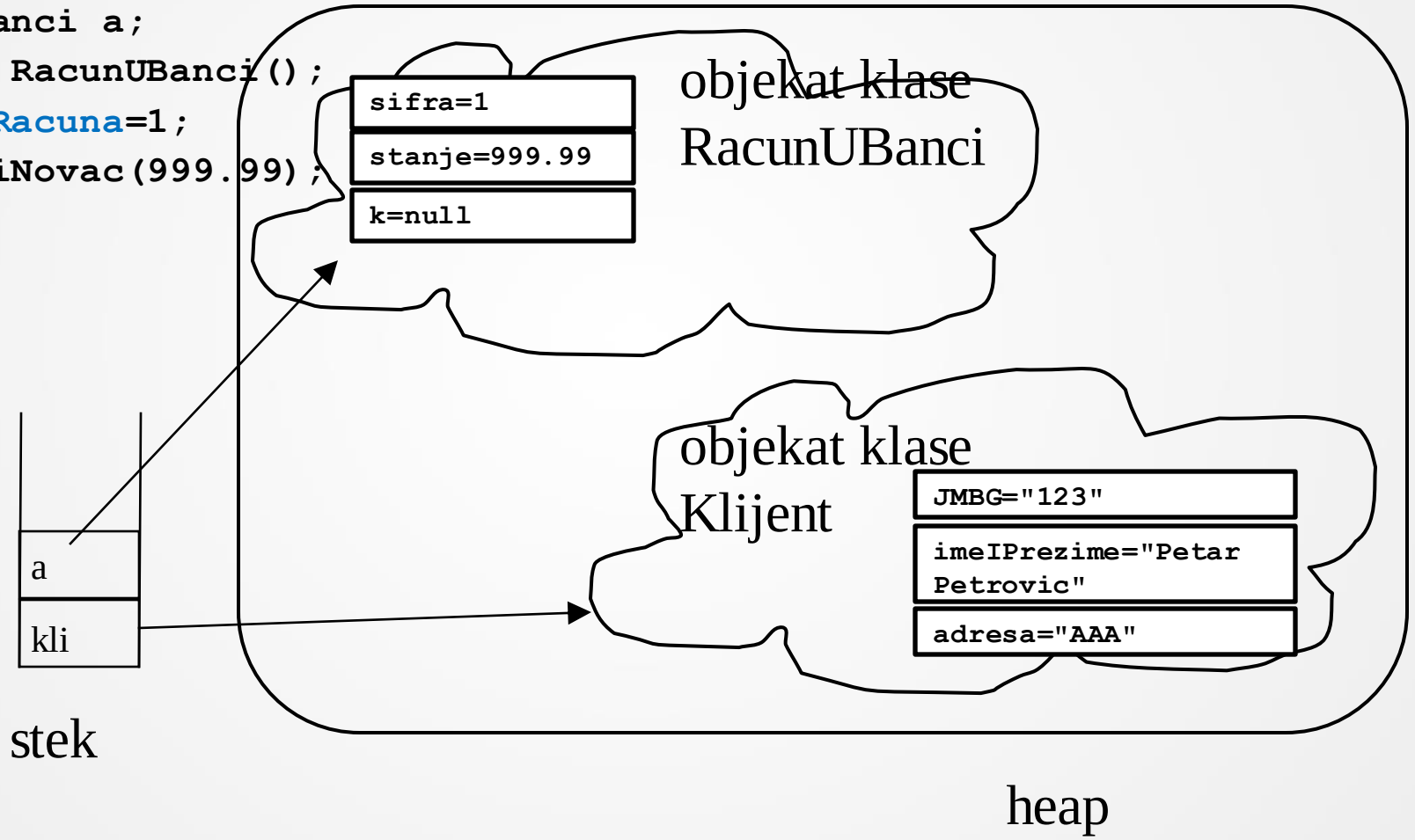
```
Klijent kli = new Klijent();
kli.JMBG="123";
kli.imeIPrezime="Petar Petrovic";
kli.adresa="AAA";
```



Rad sa referencama i Heap memorjom

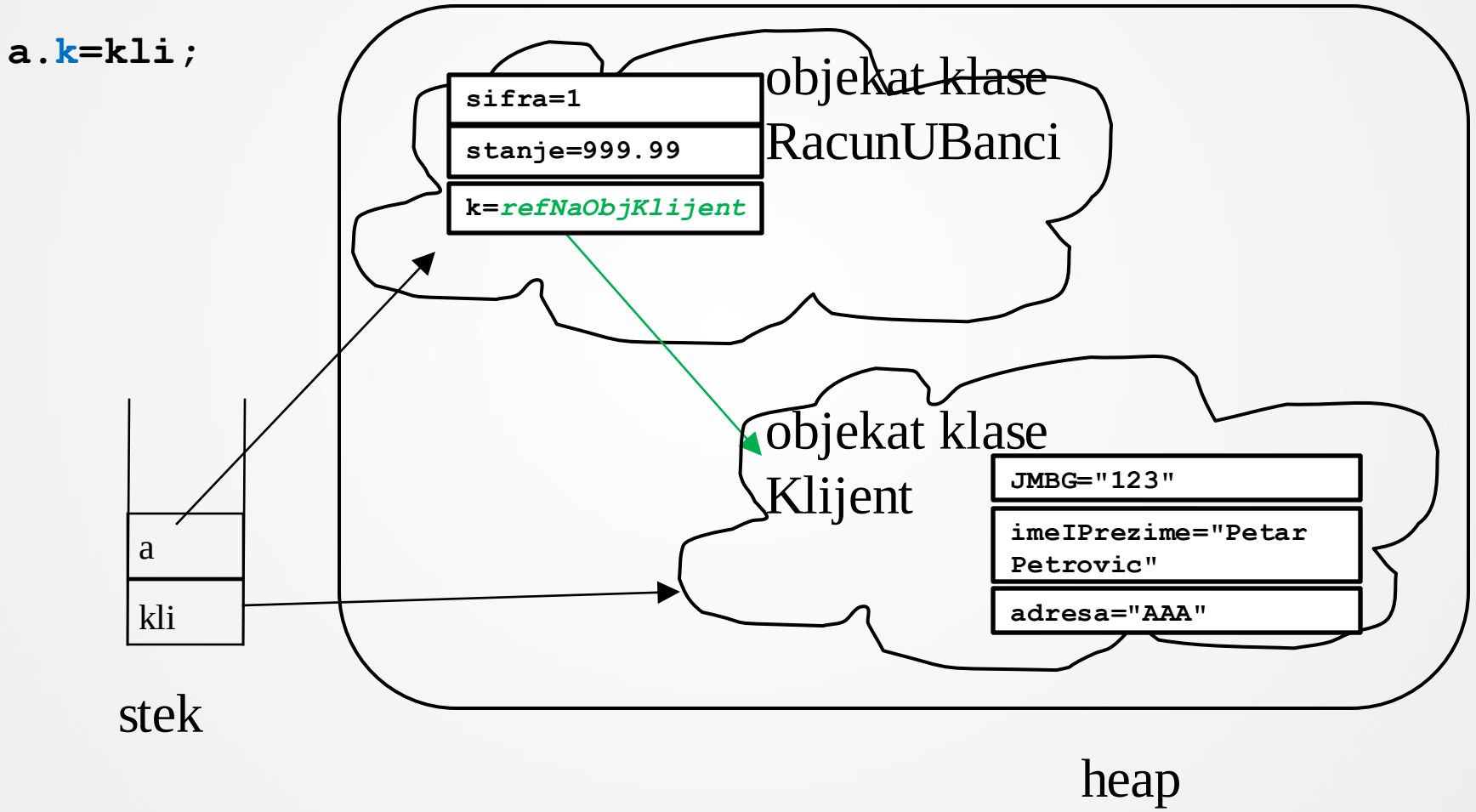
- Kod izvršen do ← 2

```
RacunUBanci a;
a = new RacunUBanci();
a.sifraRacuna=1;
a.uplatiNovac(999.99);
```



Rad sa referencama i Heap memorjom

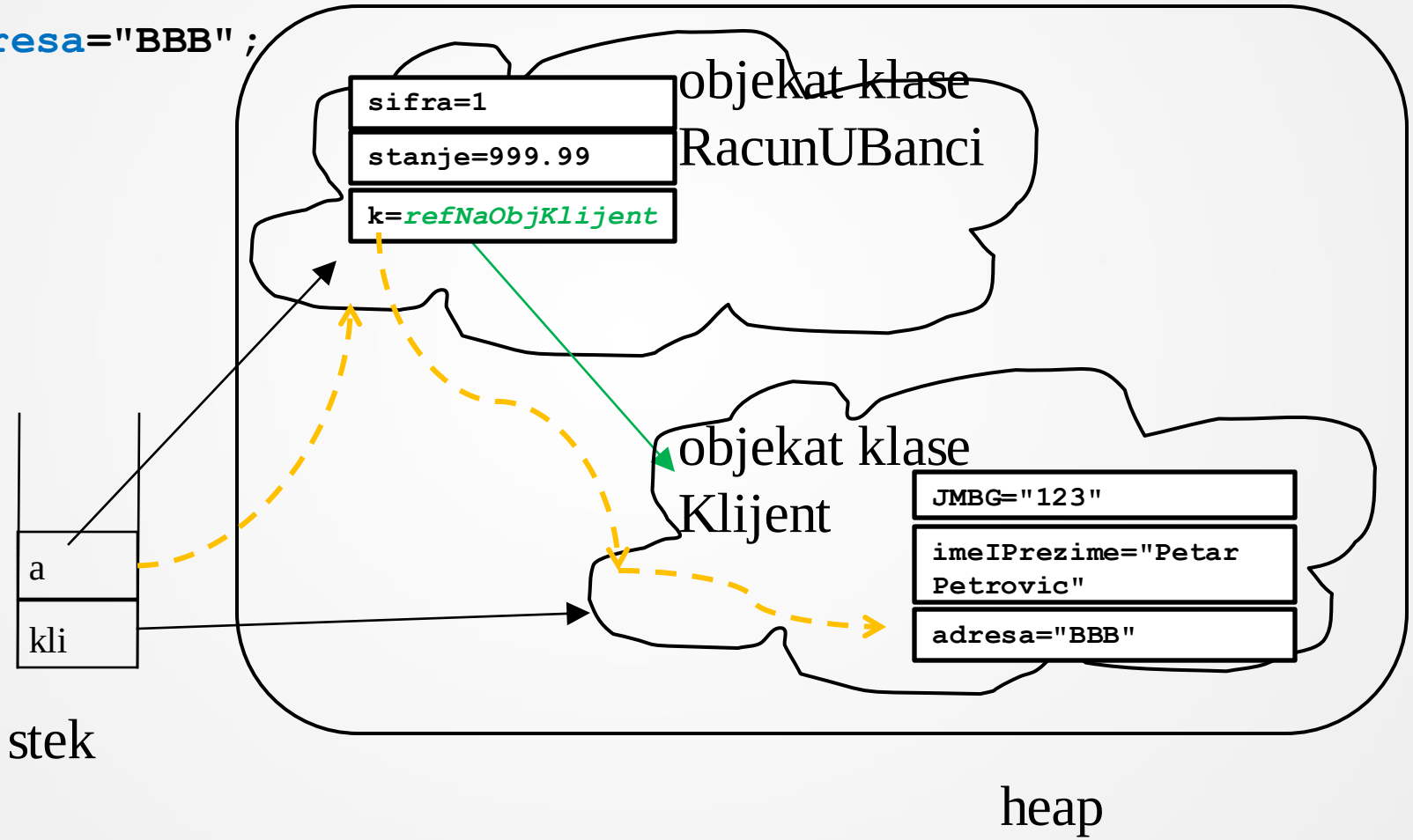
- Kod izvršen do ← 3



Rad sa referencama i Heap memorjom

- Kod izvršen do


```
a.k.adresa="BBB";
```

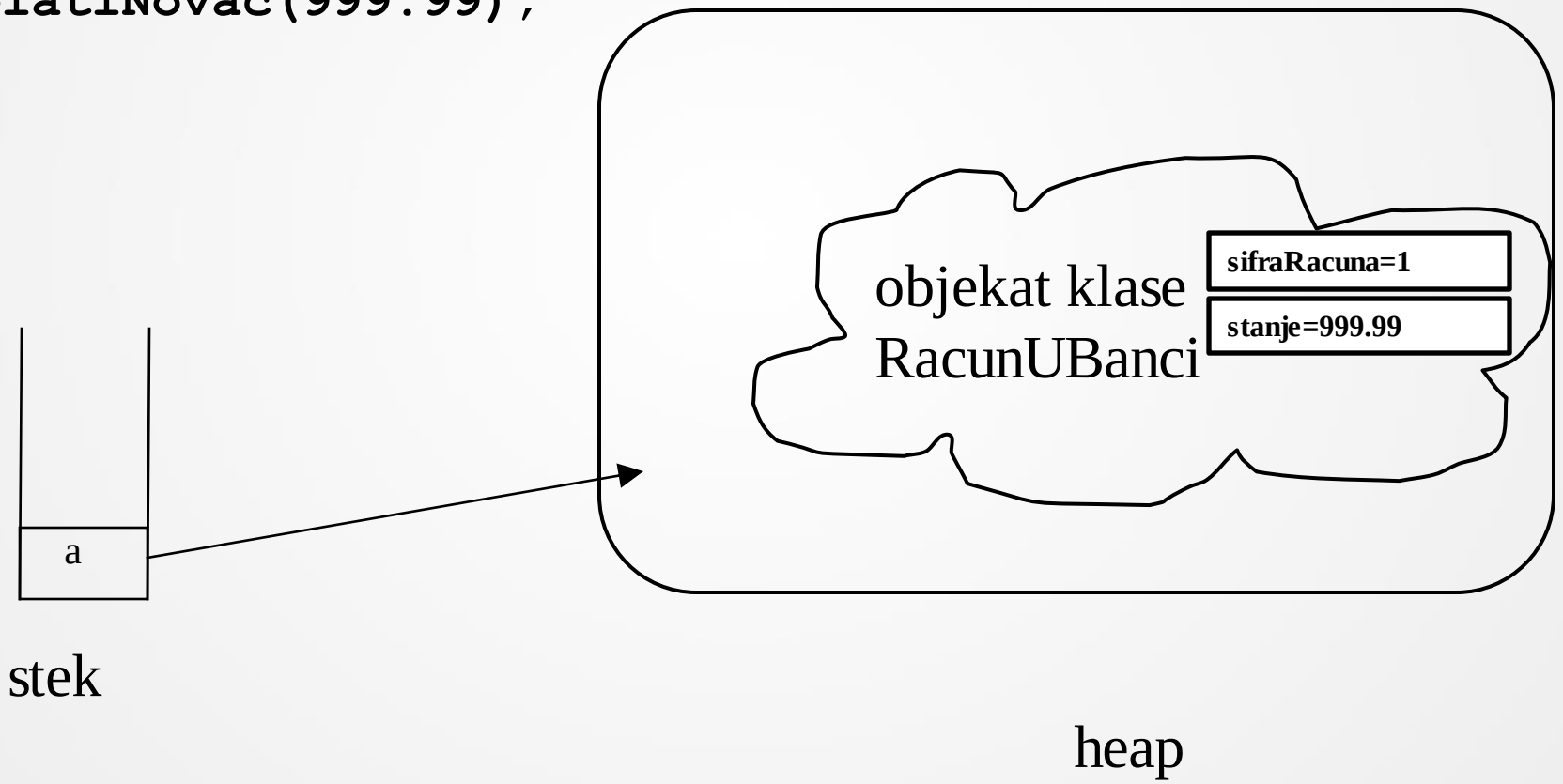


Referenca na objekat kao parametar metode

```
void metodaT(RacunUBanci aF) {
  aF.skiniNovac(99.99);
}
void main(args[]) {
...
a = new RacunUBanci();
a.sifraRacuna=1;
a.uplatiNovac(999.99);
metodaT(a);
...
```

Referenca na objekat kao parametar metode

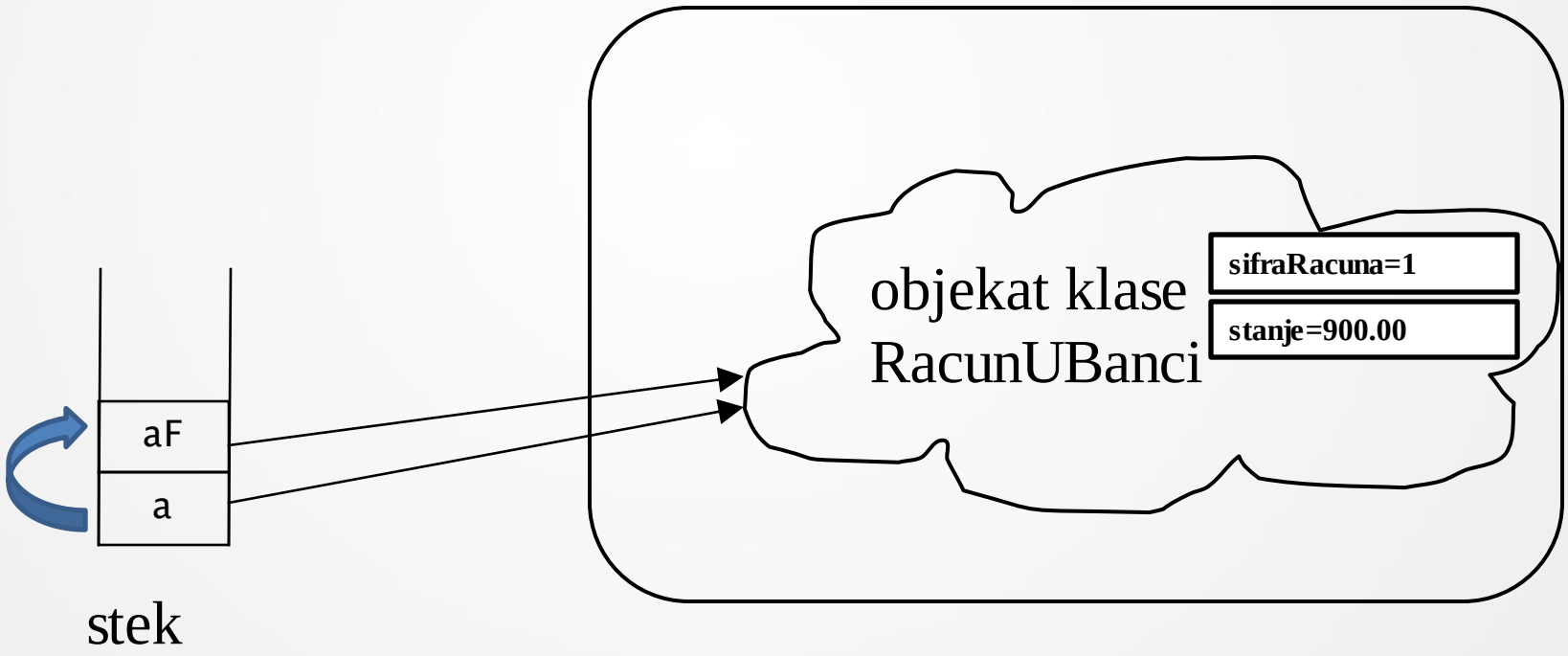
```
...
a = new RacunUBanci();
a.sifraRacuna=1;
a.uplatiNovac(999.99);
```



Referenca na objekat kao parametar metode

```
void metodaT(RacunUBanci aF) {
  aF.skiniNovac(99.99);
}
...
metodaT(a);
...
```

- Prenos je strogo po vrednosti, u promenljivu aF kopira se se vrednost reference iz a.



Literal Null i This

- Ako želimo da inicijalizujemo referencu tako da ona ne ukazuje ni na jedan objekat, onda takvoj promenljivoj dodeljujemo **null** vrednost, odn. **null** literal:

```
RacunUBanci a = null;
a.k = null;
```

- Ključna reč **this** je referenca na objekat nad kojim je pozvana metoda. Literal **this** se često koristi unutar metoda modelovanih klasa u Javi.

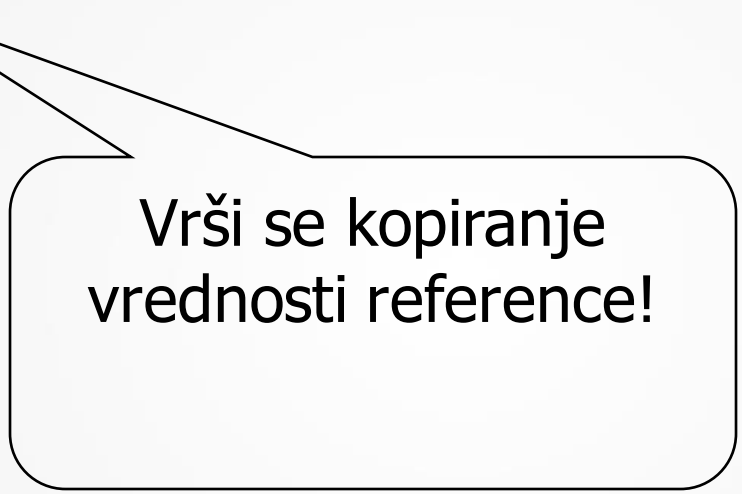
```
/** Atribut koji opisuje stanje*/
double stanje;
/** Postavlja vrednost atributa */
void postaviNovoStanje(double stanje) {
  this.stanje = stanje;
}
```

Method overloading

- U klasi može da postoji više metoda sa istim imenom
- Razlikuju se po parametrima (broju i/ili tipu ulaznih parametara)
- Metode se nikada ne razlikuju po povratnoj vrednosti

Operator dodele vrednosti

```
RacunUBanci a = new RacunUBanci();
RacunUBanci b = new RacunUBanci();
b = a;
```



Primer modelovanja 1 klase

- PODSEĆANJE SMERNICA ZA MODELOVANJE:
  - Ako u tekstu zadatka identifikujemo varijaciju reči više znamo da možemo imati atribut koji će biti Set/Lista/Mapa i koji će sadržati reference ka više objekata.
  - Ako u tekstu zadatka identifikujemo reči jedan znamo da možemo imati atribut koji će sadržati referencu ka jednom objektu.
- Implementirati aplikaciju koja omogućuje rad sa entitetima Studentske Službe. Entiteti su studenti.
- Student ima svoj *id* (identifikator), *indeks*, *ime*, *prezime* i *grad*.
  - Student može da pohađa više predmeta, a ne mora da pohađa ni jedan.

Primer01

Nizovi i Objekti

```
int a[] = new int[5]; //sve vrednosti u nizu su 0
int a[] = { 1, 2, 3, 4, 5 };
```

- Kako kod niza objekta, vrednosti elemenata mogu biti **null** ili referenca na objekat, tako pri ispisu svih elemenata niza moramo biti obazrivi i voditi računa da ne dođe do **NullPointerException** greške.

```
RacunUBanci[] racuniKlijenta= new RacunUBanci[5];
//sve vrednosti u nizu su null

//dodaj objekte u niz
racuniKlijenta[0] = new RacunUBanci();
racuniKlijenta[1] = new RacunUBanci();
```

…

ili

```
for(int i = 0; i < racuniKlijenta.length; i++)
    racuniKlijenta[i] = new RacunUBanci();
```

Nizovi i Objekti

- Može i jednostavnije kreiranje i popunjavanje:

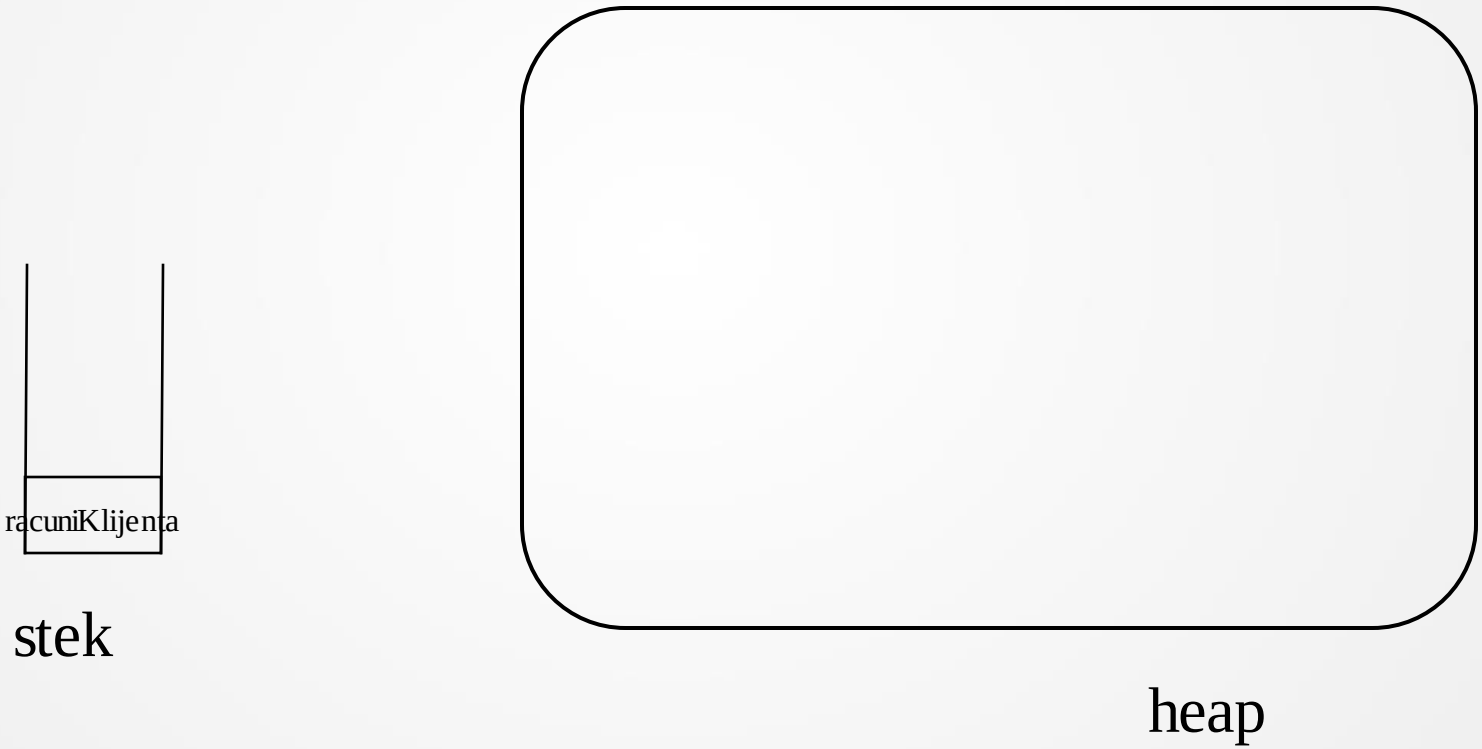
```
RacunUBanci[] racuniKlijenta = {
  new RacunUBanci(),
  new RacunUBanci(),
  new RacunUBanci()};
```

ili:

```
RacunUBanci[] racuniKlijenta;
...
racuniKlijenta = new RacunUBanci[] {
  new RacunUBanci(),
  new RacunUBanci(),
  new RacunUBanci()
};
```

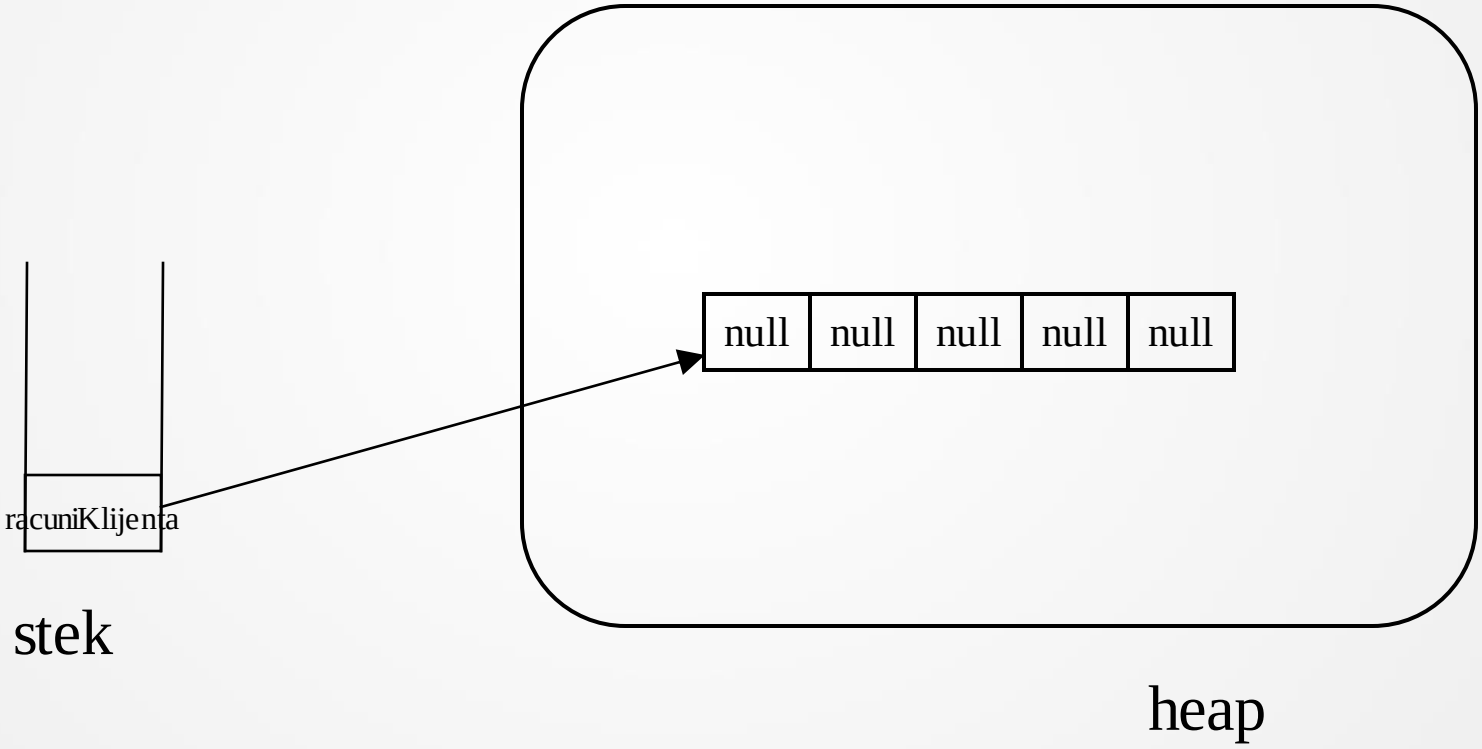
Nizovi i Objekti

```
RacunUBanci[] racuniKlijenta;
```



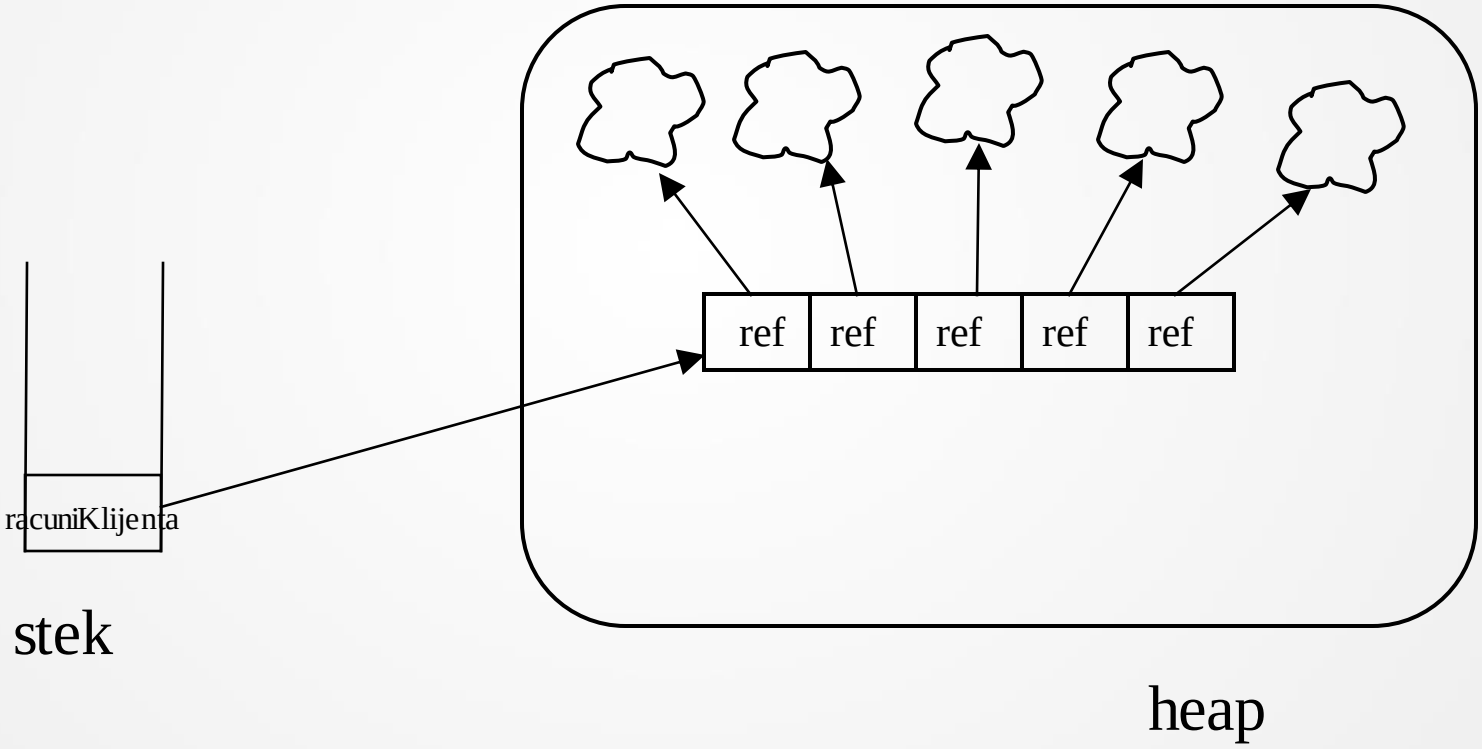
Nizovi i Objekti

```
racuniKlijenta = new RacunUBanci[5];
```



Nizovi i Objekti

```
for(int i = 0; i < racuniKlijenta.length; i++)
 racuniKlijenta[i] = new RacunUBanci();
//šta bi bilo na slici ako for petlja ide od 0 do 2?
```



Liste i Objekti modelovanih entiteta

- Za razliku od niza objekata koji je fiksne dužine, čiji elementi mogu sadržati **null** vrednosti, lista je inicijalno prazna i ona se proširuje tek kada se u listu doda neka reference na objekat.

```
ArrayList<Predmet> lista = new ArrayList<Predmet>();
Predmet fiz =  new Predmet();
fiz.id = 2;
fiz.naziv = "Fizika";
lista.add(fiz);

Predmet mat =  new Predmet();
mat.id = 1;
mat.naziv = "Matematika";
lista.add(mat);

Predmet inf =  new Predmet();
inf.id = 3;
inf.naziv = "Informatika";
lista.set(0, inf);
```

Primer modelovanja više klase

- PODSEĆANJE SMERNICA ZA MODELOVANJE:
  - Ako u tekstu zadatka identifikujemo varijaciju reči više znamo da možemo imati atribut koji će biti Set/Lista/Mapa i koji će sadržati reference ka više objekata.
  - Ako u tekstu zadatka identifikujemo reči jedan znamo da možemo imati atribut koji će sadržati referencu ka jednom objektu.
- Implementirati aplikaciju koja omogućuje rad sa entitetima Studentske Službe. Entiteti su studenti i premeti.
  - Student ima svoj id (identifikator), indeks, ime, prezime i grad.
    - Student može da pohađa više predmeta, a ne mora da pohađa ni jedan.
  - Predmet ima svoj id (identifikator) i naziv.
    - Predmet može da pohađa više studenata, a ne mora da pohađa ni jedan.

Primer02

Modifikatori pristupa

- Ponekad je potrebno obezbediti kontrolisan pristup atributima, kako za čitanje, tako i za pisanje. Tada se koriste se modifikatori pristupa
  - **public** – vidljiv za sve klase
  - **protected** – vidljiv samo za klase naslednice i klase iz istog paketa
  - **private** – vidljiv samo unutar svoje klase
  - **nespecificiran** (package-private) – vidljiv samo za klase iz istog paketa (direktorijuma, foldera)

- Modifikatori pristupa se navode ispred definicija klasa, metoda i atributa.

Modifikatori pristupa

- Kada atributima i metodama odredimo i napišemo modifikatore pristupa, dobija se klasa kojoj je omogućen **kontrolisani** pristup iz dugih klasa i programa.
- Klase mogu međusobno da komuniciraju bez znanja o međusobnoj implementaciji njihovih metoda, npr. da li pozvana javna metoda u svom telu poziva i neke druge zaštićene metode ili da ona koristi neke zaštićene atribute, itd.
- **Detalji implementacije su skriveni**, tj. **enkapsulirani** unutar klase. Druga klasa vidi prvu klasu samo kroz metode kojima ona može da pristupi!!!

Primer03

Get i Set metode

- Kada je potrebno obezbediti kontrolisan pristup atributima (čitanje i izmena vrednosti) koriste se *getters* i *setters* metode.

```
public class Student {
  private String ime;

  public String getIme() {
    return ime;
  }
  public void setIme(String ime) {
    this.ime = ime;
  }
}
```

Get i Set metode

- Kombinacija nevidljivog atributa i vidljivih get i set metoda naziva se svojstvo (*property*).
- Ovim je omogućeno da se čitanje vrednosti svojstva samo sprovodi kroz njegov *getter*, a izmena samo kroz *setter*.
- Ako izostavimo *setter*, dobijamo *read only* svojstvo.

Inicijalizacija Objekta

- Ako želimo posebnu akciju prilikom kreiranja objekta neke klase, napravićemo konstruktor.
- Konstruktor je posebna metoda koja konstruiše objekat klase.
- Konstruktor se automatski poziva prilikom kreiranja objekta.
- Ukoliko ne postoji napisan ni jedan konstruktor, kompajler će sam generisati podrzumevani (*default*-ni), koji ništa ne radi.

```
RacunUBanci a = new RacunUBanci();
```

- Ima obavezno isto ime kao i klasa i nema povratni tip

Uništavanje Objekta

- U Javi ne postoji metoda destrukor za uništavanje objekata kao u C++.
- Možemo napisati posebnu metodu ***finalize()***, koja se poziva neposredno pre oslobađanja memorije koju je objekat zauzimao, ali opet nemamo garanciju da će metoda ikada biti pozvana.

Konstruktori klase

- Ako ne napravimo konstruktor, kompajler će sam napraviti default konstruktor, koji ništa ne radi.
- Taj konstruktor se neće kreirati na nivou izvornog koda, već na nivou bajt-koda (prevedenog koda).
- U konstruktoru inicijalizujemo atribute koji bi trebalo da su inicijalizovani.
- Konstruktor može primati i parametre.
- Kako je konstruktor metoda klase možemo da napravimo proizvoljan broj konstruktora sve dok se oni razlikuju po broju i tipu parametara.
- Ukoliko se u klasi napiše barem jedan konstruktor, tada podrazumevani konstruktor više ne postoji.

Konstruktori klase

```
class RacunUBanci {
   int    sifraRacuna;
   double stanje;
   Klijent k;

   RacunUBanci(){} //konstrukor bez parametara

   RacunUBanci(int sifraRacuna, double stanje){
      this.sifraRacuna=sifraRacuna;
      this.stanje=stanje;
   } //konstrukor sa 2 parametara

   RacunUBanci(int sifraRacuna, double stanje, Klijent k){
      this.sifraRacuna=sifraRacuna;
      this.stanje=stanje;
      this.k=k;
   } //konstrukor sa 3 parametara
  ...
}
```

Objekti – pozivanje konstruktora

- Primer kreiranje Objekta pozivanjem rezličitih konstruktora

```
class Test {
    public static void main(String args[]) {
      RacunUBanci a = new RacunUBanci();
      RacunUBanci b = new RacunUBanci(2, 0.0);
      Klijent kli = new Klijent();
      RacunUBanci c = new RacunUBanci(3, 999.99, kli);
    }
}
```

Primer04

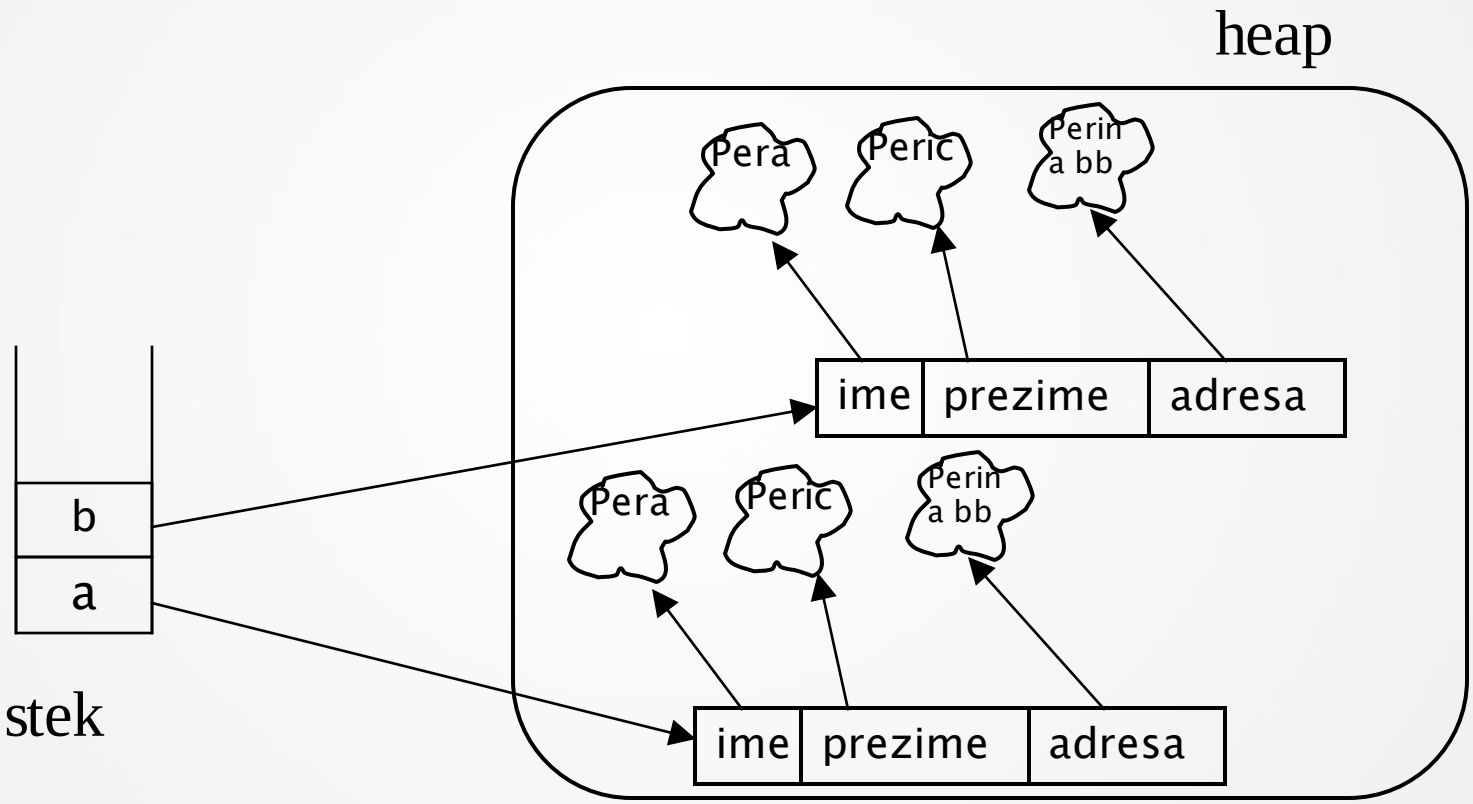
Deep vs. shallow copy

- Ako operatorom '=' zapravo prenosimo vrenost reference, kako onda da napravimo kopiju nekog objekta?
  - ako napravimo metodu:

    ```
    KopijaObjekta deepCopy() {...}
    ```

    koja će praviti kopiju objekta, onda moramo da za svaki atribut uradimo deep copy
  - operator '=' kod primitivnih tipova radi deep copy
    - kod referenci ne radi deep copy objekta, već kopira reference

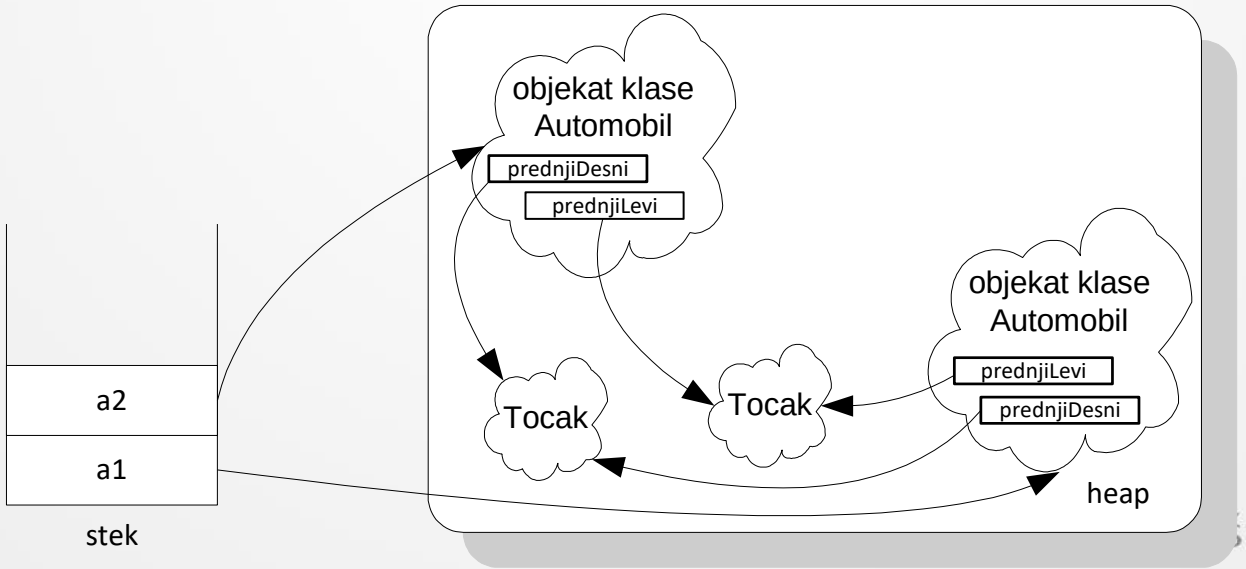
Deep vs. shallow copy




```
Osoba a = new Osoba("Pera", "Peric", "Perina bb");
Osoba b = a.deepCopy();
```

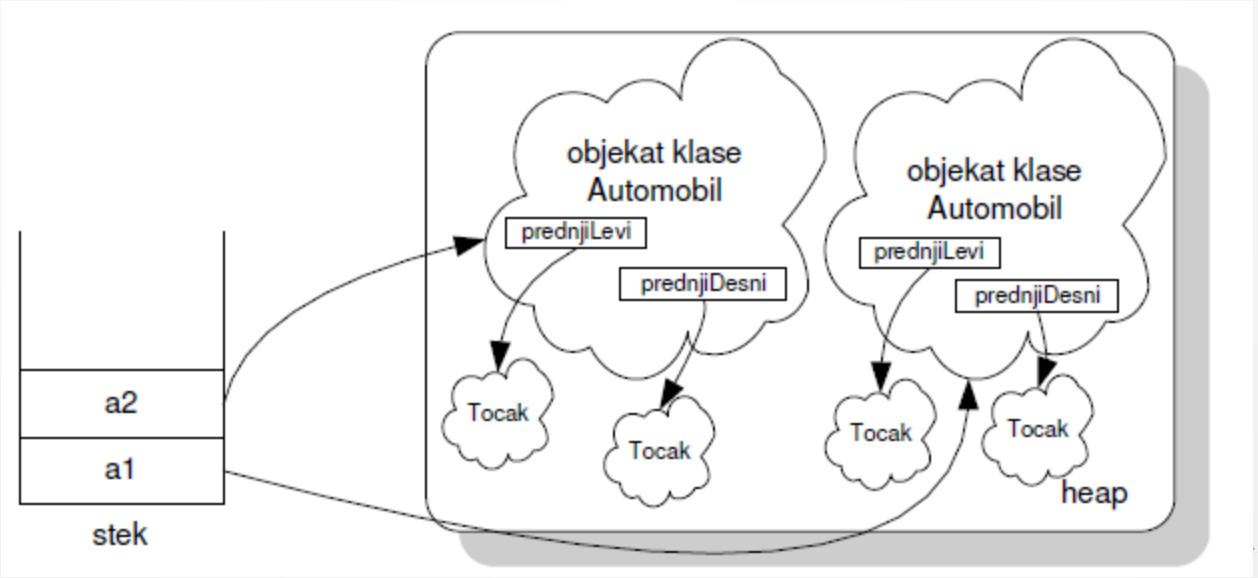
Plitka kopija

```
Automobil shallowCopy () {
  Automobil ret = new Automobil();
  ret.prednjiLevi = this.prednjiLevi;
  ret.prednjiDesni = this.prednjiDesni;
  ret.zadnjiLevi = this.zadnjiLevi;
  ret.zadnjiDesni = this.zadnjiDesni;
}
```



Duboka kopija

```
Automobil deepCopy () {
  Automobil ret = new Automobil();
  ret.prednjiLevi = this.prednjiLevi.deepCopy();
  ret.prednjiDesni = this.prednjiDesni.deepCopy();
  ret.zadnjiLevi = this.zadnjiLevi.deepCopy();
  ret.zadnjiDesni = this.zadnjiDesni.deepCopy();
}
```



Klasa Object

- Sve Java klase direktno ili indirektno nasleđuju klasu Object
- Klasa Object definiše osnovne metode koje imaju svi objekti u Javi a to su
  - equals(o) –metoda za poređenje dva objekta,
  - toString()– metoda za prebacivanje objekta u string reprezentaciju ,
  - hashCode()– metoda za računanje hash vrednosti objekta (koristi se za HashMap i HashSet),
  - getClass()– metoda koja vraća objekat klase Class koja sadrži osnovne metapodatke o klasi (ne o objektu, nego baš o klasi)
    - Mogu se pronaći informacije koje metode i atributi postoje u klasi, kao i koji konstruktori, podaci o statičkim atributima i metodama.
    - Refleksija – inspekcija koje metode ima neki objekat i pozivanje tih metoda nad objektom

Klasa Object

- clone() – metoda za kloniranje objekta, odnosno njegovu kompletnu kopiju ,
- finalize() – metodu za čišćenje određenih resursa pre uništenja objekta ,
- wait() – metoda za rad sa procesorskim nitima–čekaj,
- notify() – metoda za rad sa procesorskim nitima–obavesti prvu koja čeka,
- notifyAll() – metoda za rad sa procesorskim nitima–obavesti sve koje čekaju.

Klasa Object

- To znači da za svaku klasu možemo da redefinišemo pomenute metode

```
public class Student {
...
 public boolean equals(Object obj) {
   if (obj instanceof Student && obj != null)
        return id.equals((Student)obj.getId());
   else
        return false;
 }
 public String toString() {
   return "Student sa id " + id + " čije je ime i
        prezime " + ime + " " + prezime + " ima indeks "
        + indeks + " i zivi u gradu " + grad;
 }
}
```

Klasa Object

- Ukolio identifikator klase predstavlja kombinacija više objekata, tada se vrednost tih objekata mora proveriti u metodi equals

```
public class IspitnaPrijava{
...
  public boolean equals(Object obj) {
     if (obj instanceof IspitnaPrijava && obj != null){
            IspitnaPrijava iP = (IspitnaPrijava)obj;
            return (st.equals(iP.getSt()) && pr.equals(iP.getPr()) &&
                                          ir.equals(iP.getIr()))
     }
     return false;
  }
...
}
```

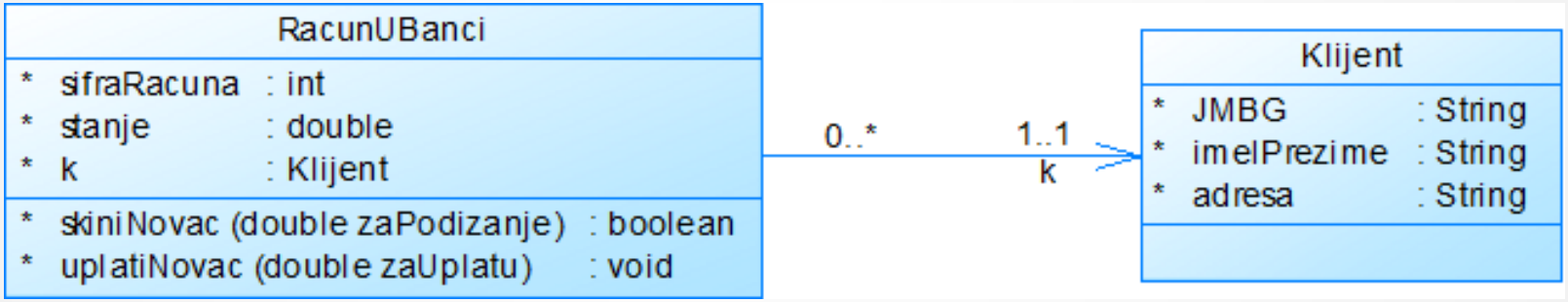
Primer05

Relacije između objekata

- Relacija može biti asocijacija, agregacija i kompozicija.
- Java (kao i ostali objektno orijentisani jezici) ne pruža nikakve specijalne jezičke konstrukcije za označavanje pomenutih relacija.
- *Unified Modeling Language* (UML) podržava simbole kojima se mogu definisati navedene relacije između objekata. UML je opšte namenski jezik za razvoj i modelovanje u oblasti softverskog inženjerstva, čijom se upotrebom postiže standardizovan način za vizualizaciju dizajna softverskog sistema.
- UML dijagram klasa (*Class Diagram*) se koristi za modelovanje klasa u aplikaciji.
- Sledeći dijagrami su kreirani upotrebom alata *SAP Power Designer 16*.
- Sve navedene relacije su implementirane u UML korišćenjem simbola strelica koja povezuje modelovane klase za objekte.
- U UML smer strelice određuje smer sadržanja informacija (koji objekat sadrži informacije o drugom objektu/listi objekta).

Relacije između objekata

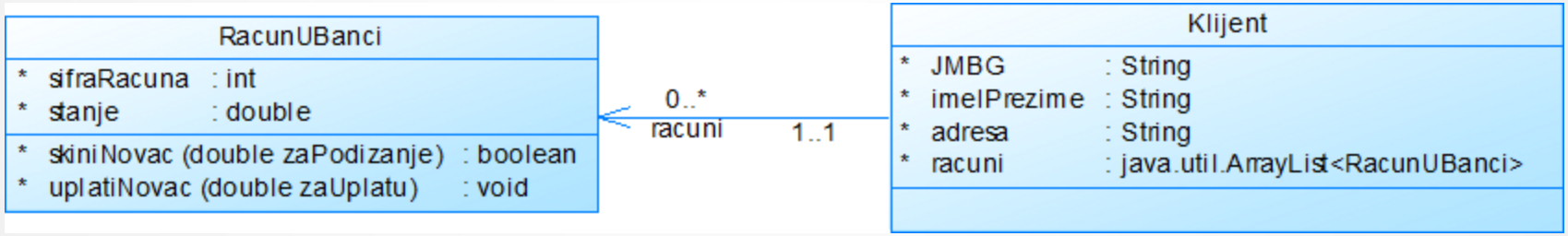
- Kada vrh strelice pokazuje kardinalitet učešća 1 objekta (0..1 ili 1..1), tada će se u klasi sa suprotne strane strelice definisati atribut klase koji će sadržati referencu ka 1 objektu klase na koju je usmerena strelica.



- Ako postoji labela (tekstualna oznaka) kod kardinaliteta tada naziv labele postaje naziv atributa klase.

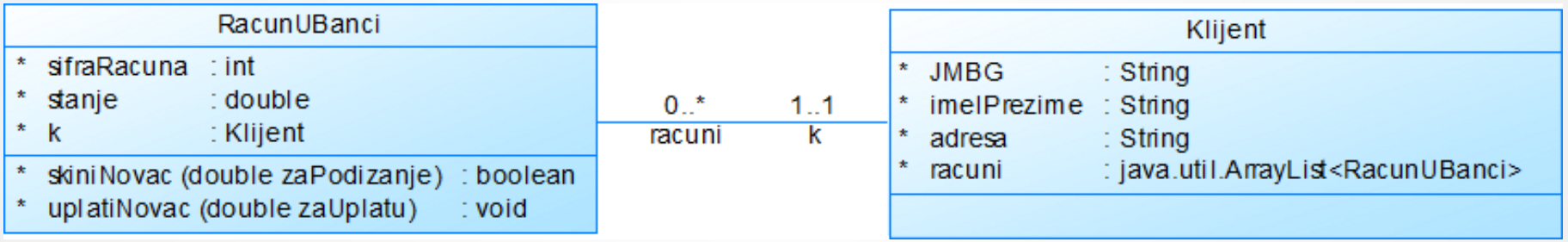
Relacije između objekata

- Kada vrh strelice pokazuje kardinalitet učešća samo više objekta (0..* ili 1..*), tada će se u klasi sa suprotne strane strelice definisati atribut klase koji će sadržati referencu ka kolekciji (*Listi/Mapi/Setu*) koja sadrži reference više objekta klase na koju je usmerena strelica.



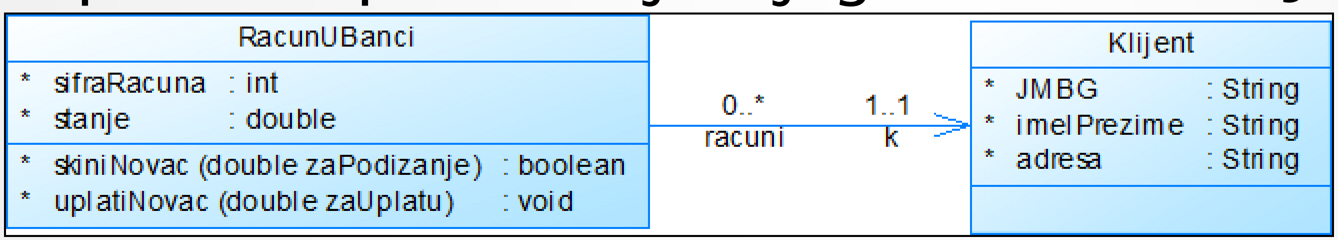
Relacije između objekata

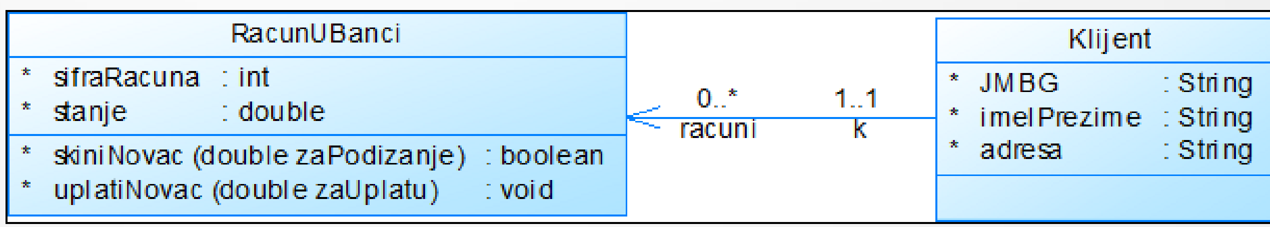
- Kada ne postoji vrh strelice (obična linija) tada obe klase mogu da sadrže atribute klase sa referencama na objekte suprotnih klasa koje se definišu po prethodno navedenim pravilima.

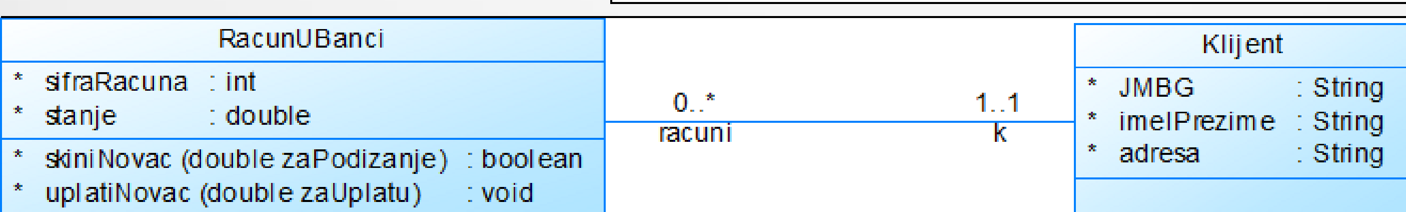


Relacije između objekata

- Kod većine UML alata, pa i kod *Power Designer*-a, prilikom definisanja određene relacije između klasa, neophodno je samo spojiti klase strelicom/crticom i definisati odgovarajuće kardinalitete. U klasama nije obavezno navesti atribute klasa koji predstavljaju relaciju, jer će njih alat automatski sam generisati prilikom prevođenja dijagrama klasa u Java kod.

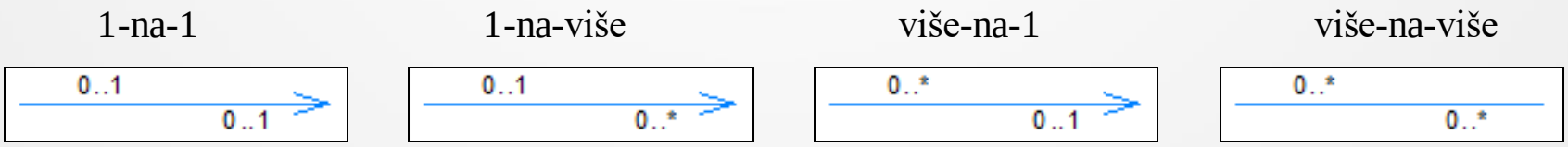






Relacije između objekata

- Asocijacija
  - **Govori da su 2 objekta u vezi**
  - Asocijacija je strukturna veza kojom se određuje da li je objekt jedne klase povezan s objektom druge ili iste klase
  - **Saradnja na nivou klasa**, se prikazuje pomoću asocijacije
  - Relacije između klasa mogu biti: 1-na-1 (**1** Pacijent ima **1** Zdravstveni karton), 1-na-više (**1** Klijent poseduje **više** Racuna), više-na-1 (**više** Poglavlja napisano je od stane **1** Autora), više-na-više (**više** Studenta može da pohađa **više** Predmeta)
  - predstavljena je kao obična strelica ili obična linija

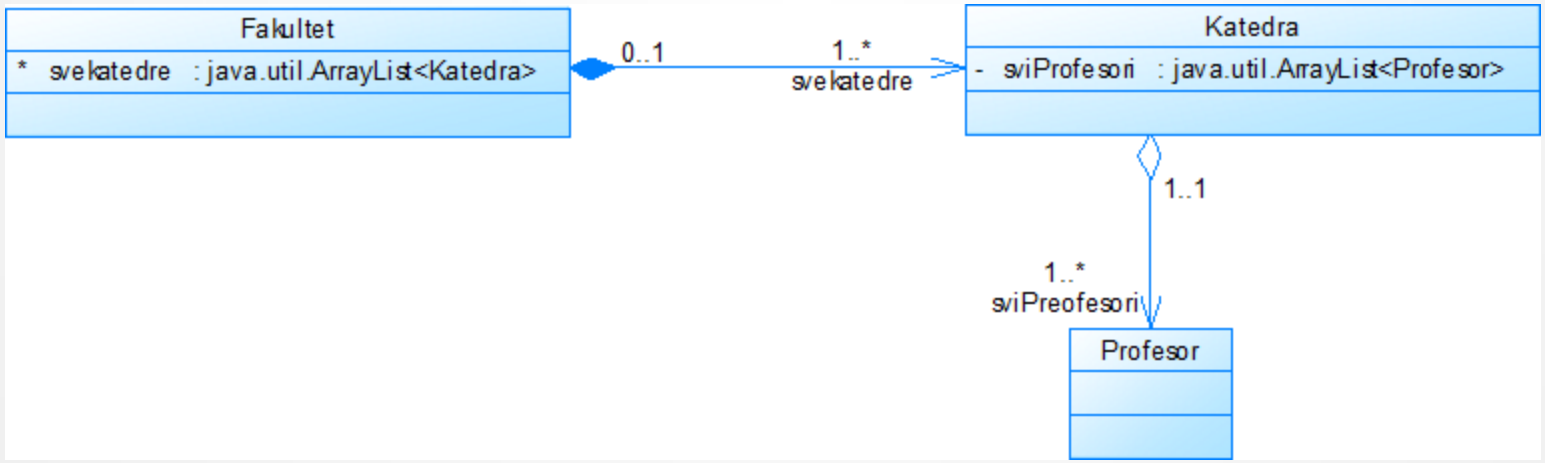


Relacije između objekata

- Agregacija – veza sadržavanja
  - Vrsta asocijacije, samo određenija
  - direktna asocijacija između objekata, konkretnije definiše odnos između objekata
  - **jedna strana igra ulogu celine a druga ulogu dela** – celina sadrži delove (“has a")
  - Kod agregacije postojanje objekata koji su u relaciji agregacije je nezavisno jedan od drugog (u aplikaciji ukoliko se obriše vlasnički objekat tada neće doći do brisanja sadržanog objekta)
  - Predstavljena je kao strelica ili linija koja na početku ima simbol prazan romb

Relacije između objekata

- Agregacija – veza sadržavanja
  - Katedri pripada/sadrži više Profesora.
  - Ako obrišemo određenu Katedru, tada ne brišemo sve Profesore koji pripadaju toj Katedri.
  - Brisanje određenog Profesora neće dovesti do brisanja Katedre (samo izmena podataka)

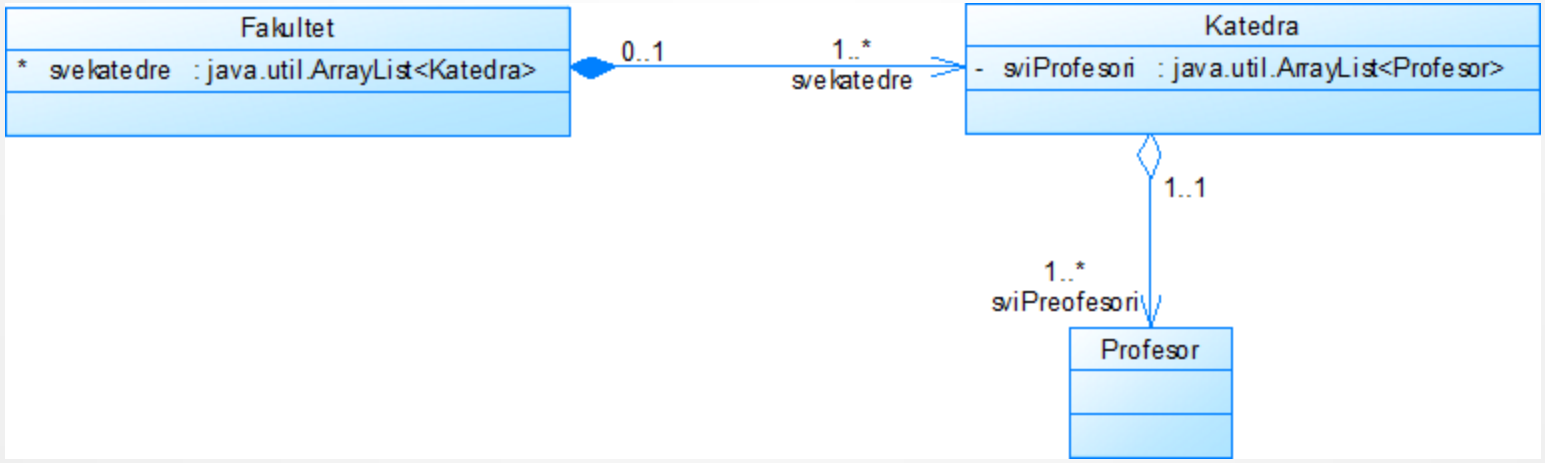


Relacije između objekata

- Kompozicija
  - Kompozicija se specijalni oblik agregacije
  - Stroga agregacija između objekata
  - **celina odgovorna za životni vek dela**
  - prvi objekat sadrži drugi, ali drugi ne može postojati bez prvog (u aplikaciji ukoliko se obriše vlasnički objekat tada će doći do brisanja i sadržanog objekta)
  - predstavljena je kao strelica ili linija koja na početku ima simbol popunjen romb

Relacije između objekata

- Kompozicija
  - Fakulteti ima svoje Katedre.
  - Ako obrišemo određeni Fakultet, tada moramo da obrišemo sve Katedre sa tog Fakulteta.
  - Brisanje određenog Katedre neće dovesti do brisanja Fakulteta (samo izmena podataka)



Immutable Objects

- *"An object is considered immutable if its state cannot change after it is constructe"* – Java 1.8. dokumentacija
- Immutable Objects je objekat kome se definiše vrednost u trenutku njegovog kreiranja. Za njega ne postoje metode, ni načini kako da se ta vrednost dodatno promeni.
- Primer su objekti klase String i objekti Wraper klasa za primitivne tipove
- Izmena postojećeg Vs kreiranje novog – programeri su često u nedoumici da li je bolje menjati postojeće objekte ili kreirati nove
  - Izmena objekata nekad je više procesorski zahtevna nego kreiranje novog objekta
  - Immutable object imaju smanjeni overhead pri uklanjanju iz memorije sa *Garbage Collector*–om, nego što je to brisanje ostalih objekata

Immutable Objects

- Prednosti korišćenja Immutable Objects su:
  - Saznanje da se njihovi podaci ne mogu promenuti – bolja enkapsulacija (ne mora brinuti o kontrolisanoj izmeni vrednosti)
    - Možemo ih korisiti slobodno, bez obaveze da strahujemo da li metoda kojoj smo prosledili objekat može na nekakav način da utiče na njegovu izmenu na način koji je zabranjen
  - sprečavanje nastanka inkonzistentnog stanja objekata kada se objekat deli između više programskih niti (multithreaded environment)
    - automatski su *thread-safe* i nemaju problema oko sinhronizacije podataka (*synchronization*)
  - jednostvani za modelovanje, testiranje i upotrebu
    - nema potrebe za njih praviti *copy constructor*
    - nema potrebe za njih praviti metodu *clone*
    - posle kreiranja objekta nema potrebe za validiranjem stanja njihovih podataka
      - nikada se njihovi podaci neće naći u nevalidnom stanju
  - predstavljaju dobre ključeve za kolekcije mape

Immutable Objects

- Korisnik može sam kreirati klasu čiji bi objekti bili immutable. Smernice su:
  - zabraniti nasleđivanje klase (postaviti *final* ispred naziva klase)
  - atributi klase moraju biti *final* i *private*
  - ne kreirati *set* metode, niti bilo koje metode koje mogu da promene stanje atributa objekta
  - ako je atribut klase referenca na objekat tada se mora izvršiti defanzivno kopiranje atributa tj. kreiranje kopije objekta u metodema u kojima se preuzima vrednost atributa tj. npr. u getMetodi.
    - npr. imamo atribut **`Date datum`** kojem preko reference možemo menjati vrednosti
    - u get metodi bi pozvali kod

```
return new Date(datum.getTime())
```

primer06