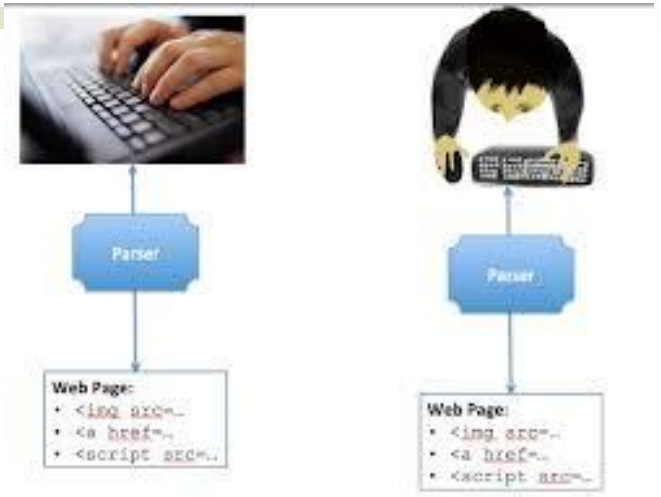
# Programiranje




Nastava: prof.dr.sc. Dražena Gašpar

Datum: 21.04.2015.

# KLASE

- **Pisanje JAVA programa podrazumijeva pisanje kolekcije definicija klasa.**
- **Definicija klase opisuje instancu klase i specificira podatke kojima upravlja objekt koji je instanca te klase, kao i funkcionalnost (upite i naredbe) koje objekt podržava.**
- **Definicija klase podrazumijeva slijedeće:**
  - **definiranje varijabli instance**
  - **definiranje varijabli klase**
  - **definiranje upita i naredbi – objedinjeno pod zajedničkim nazivom metode.**

# KLASA

**Varijabla instance** (engl. instance variable) je imenovani memorijski prostor koji se koristi za pohranjivanje podataka objekta i dodjeljuje se objektu u trenutku njegovog kreiranja.

Svaka instanca klase (svaki objekt klase) ima svoj vlastiti primjerak te varijable.

# KLASA

**Varijabla klase** (engl. class variable) podrazumijeva da klasa ima samo jedan primjerak te varijable koji sve instance te klase mogu koristiti.

Varijabla klase je element klase koji se koristi neovisno o instancama (objektima) klase.

# Objekt

Memorijski prostor za pohranjivanje podataka o objektu dodjeljuje se u trenutku kreiranja objekta i naziva se varijabla instance (engl. *instances variables*).

Varijabla instance se dodjeljuje za svako svojstvo objekta i sadrži vrijednost tog svojstva.

Varijabla instance je varijabla koja je stalni dio objekta, a memorijski prostor za nju se dodjeljuje pri kreiranju objekta

# Objekti klase

**Za kreiranje instanci klase u JAVA jeziku, odnosno objekata klase koriste se:**

- **operator new i**
- **konstruktor.**

**Primjeri:**

```
String znakovniNiz = new String();
Random r = new Random();
Brojač prviBrojač = new Brojač(10);
```

**…. Program Student1.java**

# Objekti klase

Pri kreiranju objekta potrebno je deklarirati varijablu čiji tip odgovara klasi, što nije ništa drugo do kreiranje novog tipa podatka koji se može koristiti za deklariranje objekata tog tipa.

Tako deklarirana varijabla ne definira objekt već samo pokazuje (referencira) na njega.

Stvarna, fizička kopija objekta se dodjeljuje toj varijabli pomoću operatora new. Operator new dinamički (tj. u trenutku izvršavanja programa) dodjeljuje memoriju za objekt i programu vraća referencu na njega. Ova referenca predstavlja adresu objekta stvorenog operatorom new u memoriji.

# Objekti klase

B I T N O

u varijablu se smješta referenca na objekt, a ne objekt. Iz ovoga proizlazi da JAVA svim objektima klasa memoriju mora dodjeljivati dinamički.

# Objekti klase

**Primjer:**

**Brojač prviBrojač = new Brojač(10);**

**Može se rastaviti u dva koraka:**

```
Brojač  prviBrojač;        // deklariranje reference na objekt
prviBrojač = new Brojač(10); // dodjeljivanje memorije
                             objektu Brojač
```

# Objekti klase

NE POSTOJI
Varijabla tipa
OBJEKT
Ovo je referenca

## Opći oblik new operatora je:

```
varijabla = new NazivKlase();
```

Gdje je varijabla tip klase koja se pravi, a NazivKlase označava klasu čiji se objekt (instanca) pravi.

Operator new kreira novu instanca za klasu i dodjeljuje joj memoriju.

Slijedeći korak je poziv specijalne metode za inicijaliziranje objekta i postavljanje odgovarajućih početnih (inicijalnih) vrijednosti. Ta specijalna metoda se naziva konstruktor i ona kreira i inicijalizira nove instance klase. Obično se konstruktor izričito definira unutar definicije klase. Međutim, ako se konstruktor ne naznači eksplicitno, JAVA automatski osigurava tzv. podrazumijevani (engl. default) konstruktor.

# Objekti – 3 osnovna koraka

1 3 2

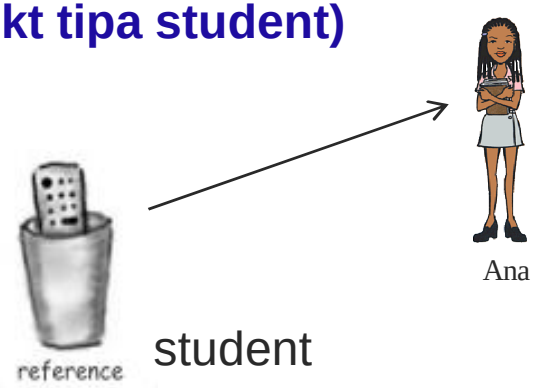
```
student Ana = new student();
```

**1. Deklariranje varijable koja sadrži referencu**
**(JVM dodjeljuje memoriju za referentnu varijablu)**


student

**2. Kreiranje objekta**
**(JVM dodjeljuje memoriju za novi objekt tipa student)**

**3. Povezivanje objekta i reference**
**(dodjeljuje referencu na objekt**
**Referentnoj varijabli)**




student

# Objekti klase

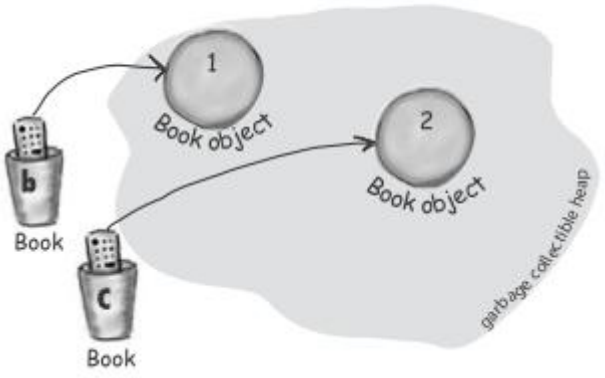
**Kod dodjeljivanja referenci na objekte varijablama, odnosno kada se jedna referentna varijabla dodjeljuje drugoj, tada se ne pravi kopija objekta, već se samo kopira referenca na objekt**:

```
Brojač prviBrojač = new Brojač(10);
Brojač drugiBrojač = prviBrojač;
```

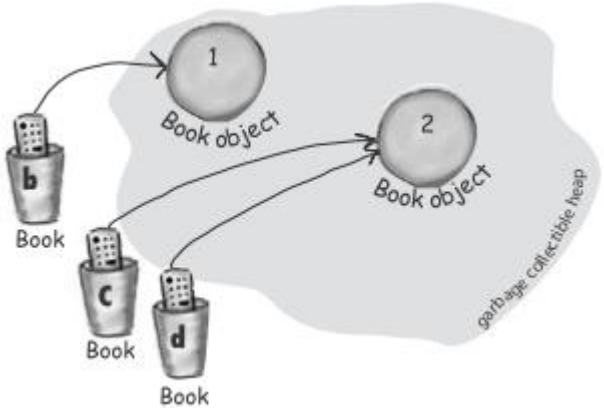
# Objekti klase

**Book b = new Book();**

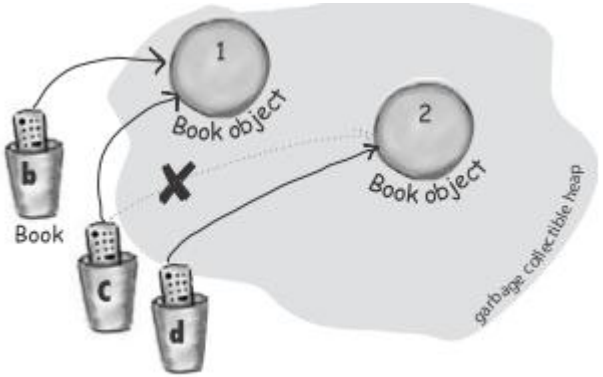
**Book c = new Book();**




**Book d= c;**




**c=b;**



# Objekti klase

Da bi se moglo raditi s objektima potrebno je

1. Napraviti (konstruirati) objekt
2. Definirati početno (inicijalno) stanje
3. Pridružiti metode objektu

JAVA rabi konstruktore za pravljenje (konstruiranje) i inicijalizaciju objekata.

# Inicijalni konstruktor

```
import java.util.Scanner;
public class Student2
{
 public static void main(String[] args)
 {
  Scanner unos = new Scanner(System.in);
  student Ana = new student();
  System.out.print("Ime studenta ");
  Ana.ime = unos.next();
  System.out.print("Prezime studenta ");
  Ana.prezime = unos.next();
  System.out.print("Status ");
  Ana.tip = unos.next();
  System.out.print("Broj indeksa ");
  Ana.indeks = unos.next();
  System.out.println(Ana.ime+" "+Ana.prezime+" "+Ana.tip+" "+Ana.indeks);
} }

class student
 {
   String ime;
   String prezime;
   String tip;
   String indeks;
   }
```

**Program: Student2.java**

# Konstruktor

Pseudo metoda koja kreira objekt. To su instance metoda s UVIJEK istim imenom kao i njihove klase.

Zadaća konstruktora jeste inicijaliziranje objekta u procesu njegovog stvaranja, tako da se nakon naredbe “new” ima odmah spreman objekt za uporabu.

# Metoda

**Metoda je naziv dan dijelu (bloku) programskog koda koji se izvršava kao odgovor na specifičnu poruku, gdje je poruka formalna komunikacija poslana od jednog objekta drugom s ciljem izvršenja određenog servisa (usluge).**

# Metoda

Dva su tipa poruka na koje objekt može odgovoriti:

- Zahtjev za podacima koji se naziva upit (engl.*query*)
- Zahtjev za promjenom stanja koji se naziva naredba (engl. *command*).

Skup upita i naredbi na koje će dani objekt odgovoriti naziva se sposobnost (mogućnost) objekta i određuje prigodom dizajniranja objekta, a u objektnoj paradigmi se implementira pomoću metode.

# Metoda

Skup svih sposobnosti (funkcionalnosti) objekta, onako kako se vidi od strane klijenta naziva se *specifikacija*

Pojam *implementacija* koristi se za opis interne strukture objekta koja ustvari čini sposobnost tj. funkcionalnost objekta

Funkcionalnost se naziva još i sučelje (engl. *interface*), ali se taj termin ne koristi da se ne bi pomiješao s Java sučeljem.

# Metode

**Opći oblik deklariranja metode je:**

```
modifikator pristupa  tip naziv(lista parametara)
{
        // tijelo metode
    return izraz
}
```

# Metode

**Lista parametara sadrži niz parova sastavljen od tipa podatka i imena parametra, razdvojenih zarezima.**

**Parametri su varijable koje prihvaćaju vrijednosti argumenata proslijeđenih metodi u trenutku njenog poziva.**

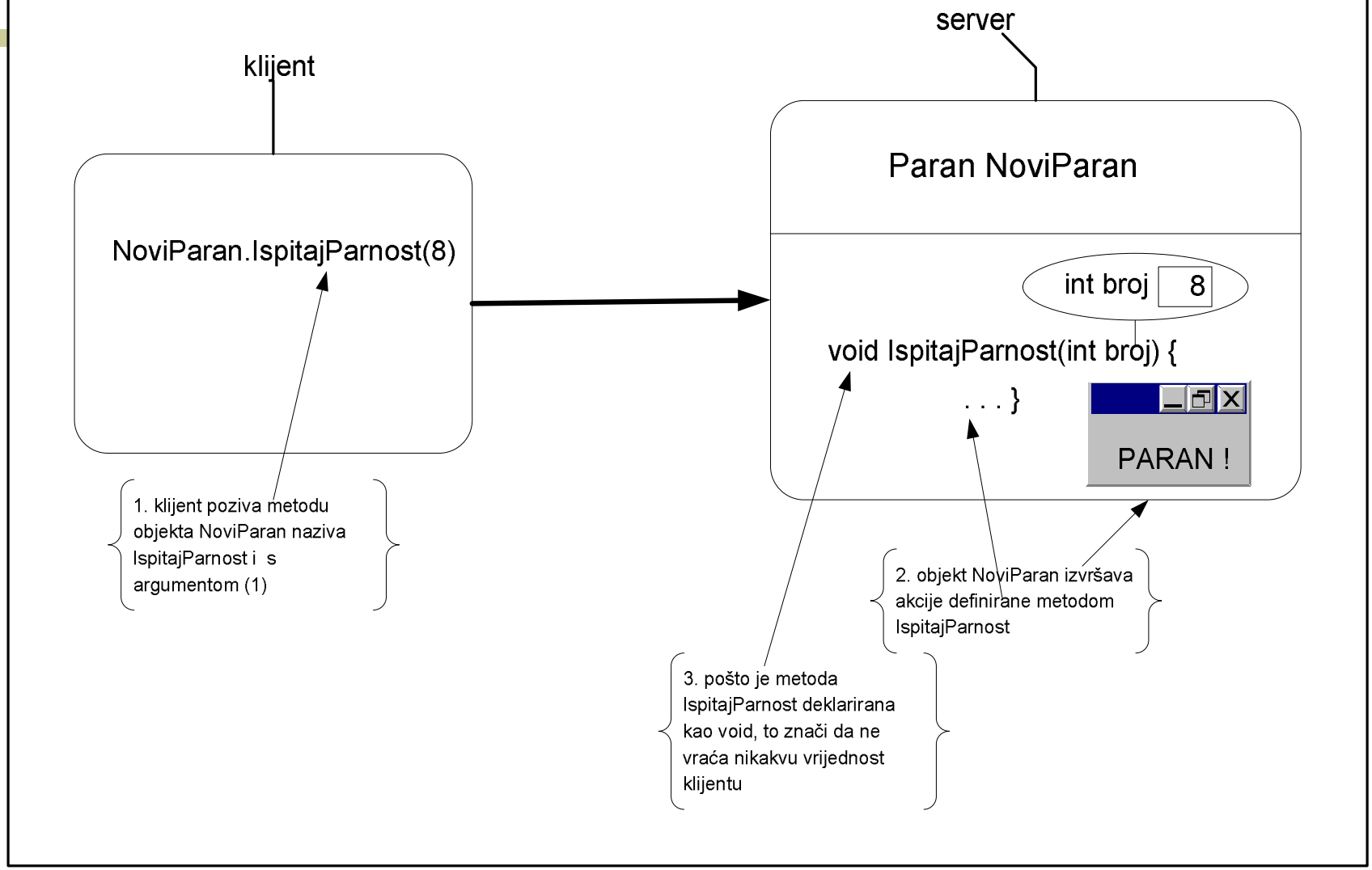
**Ako metoda nema parametara, lista parametara je prazna.**

# Metode

Način rada objekata s metodama opisuje se općenito kao:

NazivObjekta.NazivMetode(argumenti);

*Poziva se NazivMetode objekta NazivObjekta, odnosno metoda NazivMetode koja se odnosi na objekt NazivObjekta, s argumentima*


server
klijent
Paran NoviParan
NoviParan.IspitajParnost(8)
int broj 8
void IspitajParnost(int broj) {
. . . }
PARAN !
1. klijent poziva metodu objekta NoviParan naziva IspitajParnost i s argumentom (1)
2. objekt NoviParan izvršava akcije definirane metodom IspitajParnost
3. pošto je metoda IspitajParnost deklarirana kao void, to znači da ne vraća nikakvu vrijednost klijentu

# Metode

```
class Paran {
        void ispitajParnost (int broj)
{
          System.out.println("Broj je "+broj);
                  if (broj % 2 = = 0)
          System.out.println("PARAN !");
    }
      public static void main (String args[]) {
                  Paran NoviParan = new Paran();
                  NoviParan.ispitajParnost(1);
                  NoviParan.ispitajParnost(8);
                  NoviParan.ispitajParnost(93);
                  NoviParan.ispitajParnost(1432);
 }}
```

**Zadatak: doraditi program tako da korisnik izravno unosi broj i onda se provjerava parnost. (Paran.java)**

# Primjer kreiranja objekta

**Program: Student4.java**

```
import java.util.Scanner;

public class Student4
{ public static void main(String[] args)
 { Scanner unos = new Scanner(System.in);
   System.out.print("Ime studenta ");
   String imes = unos.next();
   .....
 student student1 = new
 student(imes,prezimes,tips,indekss);
  System.out.println("Student1 "+student1.uzmiime()+"
"+student1.uzmiprezime()+" "+student1.uzmitip()+"
"+student1.uzmiindeks());  }}
```

# Primjer kreiranja objekta

**Program: Student4.java**

```
class student
 { public student(String i, String p, String t, String n)
  { ime = i;
    prezime = p;
    tip = t;
    indeks = n;
  }
    public String uzmiime()
  { return ime; }
  public String uzmiprezime()
  { return prezime; }
  public String uzmitip()
  {return tip; }
  public String uzmiindeks()
  {return indeks; }
  private String ime, prezime, tip, indeks;   }
```

# “Static” varijabla

⇒Ako se varijabla definira kao “static” to znači da postoji samo jedna takva za klasu, inače svaki objekt ima svoju kopiju svih varijabli instanci.

```
class djelatnik {
        ....
        private int id;
        private static int nextId = 1;
   }
```

# “Static” varijabla

```
Public void setId() {
        id = nextId;
        nextId++;
    }

Mate.setId();
Mate.id = …;
djelatnik.nextId++;
```

# “Static” varijabla

**Program: Student4.java**

**Nadograditi s automatskim izračunom šifre za studenta i prikazom sljedeće slobodne šifre.**

**Rezultat: Student5.java**

# “Static” varijabla

**Static varijable su rijetke, za razliku od static konstanti**

```
public class Matematika  {
        …
 public static final double
PI=3.14159265358979;
      ….
   }

Matematika.PI   // poziv konstante
```

# Kreiranje objekata

**ZAPAMTITI**

- **Konstruktor ima isto ime kao i klasa**
- **Klasa može imati više konstruktora**
- **Konstruktor može imati nijedan ili jedan i više parametara**
- **Konstruktor ne vraća vrijednost**
- **Konstruktor se uvijek poziva s operatorom “new”**

# Kreiranje objekata

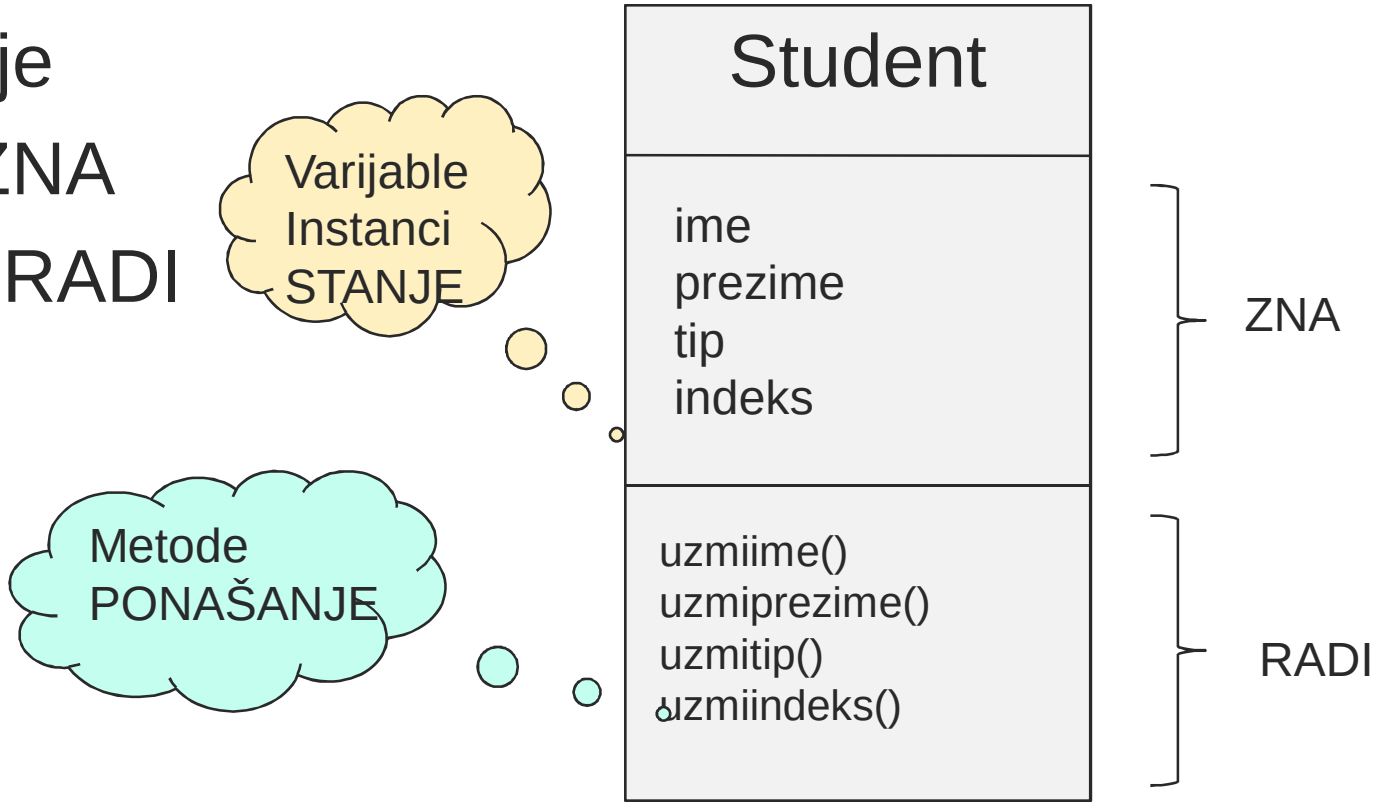
**Metode**

**VAŽNO:**

**Svaka od metoda može pristupiti privatnim varijablama instanci putem njihovog imena. To je zbog toga što su varijable instanci uvijek vidljive za metode njihove vlastite klase.**

# Metode

Klasa opisuje
što Objekt ZNA
i što Objekt RADI



# Zadatak

Klasi student dodati školarinu

Dodati metodu kojoj se parametarski šalje postotak uvećanja školarine

**Program:Student6.java**

# Analiza metode (Student6.java)

```
public void povecajskolarina(double postotak)
  {
    double povecanje = skolarina*postotak/100;
    skolarina += povecanje;
  }
```

POZIV metode:

**Izvanredni[i].povecajskolarina(10);**

**IZVRŠENJE metode:**

**double povecanje = Izvanredni[i].skolarina * 10 /100;**

**Izvanredni[i].skolarina += povecanje;**

# Analiza metode

**Metoda povecajskolarina ima 2 parametra:**

- **Implicitni**
- **Eksplicitni**

**Implicitni parametar je objekt tipa “student” (Izvanredni[i]) koji se pojavljuje prije imena metode.**

**Ključna riječ “this” odnosi se na implicitni parametar.**

**Eksplicitni parametar je onaj što se eksplicitno navodi u deklaraciji metode.**

# Analiza metode

Eksplicitni parametar

```
public void povecajskolarina(double postotak)
  {
    double povecanje = this.skolarina*postotak/100;
    this.skolarina += povecanje;
  }
```

Implicitni parametar

# Metode

**Svaka JAVA metoda ima dva parametra:**

**- eskplicitni (vidljivi) i**

**- implicitni (nevidljivi ili prikriveni).**

**Eksplicitni parametar je lista parametara koja se navodi u zagradi iza naziva metode i koja služi za pohranjivanje argumenata koje objekt šalje metodi.**

**Implicitni parametar metode se ne navodi i nije vidljiv iz deklaracije metode. To je objekt u pozivu metode, odnosno NazivObjekta koji dolazi ispred operatora točka (.) i NazivaMetode.**

**Ključna riječ koja se odnosi na implicitni parametar metode je**

**this**

**i ona se odnosi na trenutni (tekući) objekt koji je pozvao tu metodu.**

# Ključna riječ: **this**

**Ako postoje dva objekta istog tipa npr. a i b, kako se može pozvati metoda f() za oba:**

**Primjer:**

```
class Banana { void f(int i) { /* … */}}
Banana a = new Banana (), b = new Banana();
a.f(1);
b.f(2);
```

**Kompilator odradi "pozadinski" posao i pošalje nevidljivi prvi argument metodi f koji ukazuje na objekt s kojim se radi (Banana.f(a,1); i Banana.f(b,2);**

**Nema identifikatora za to – samo this koji se može koristiti samo unutar metode.**

Pravokutnik.java – Blok za pisanje

Datoteka Uređivanje Formatiranje Prikaz Pomoć

```
import java.util.Scanner;
class Pravokutnik {
        int x1, y1 = 0;
        Pravokutnik napraviPravokutnik(int x1, int y1) {
                this.x1 = x1;
                this.y1 = y1;
                return this; }

        void ispisiPravokutnik() {
             System.out.print("Dimenzije napravljenog pravokutnika:  <"+x1+", "+y1+" >");
              }

        public static void main (String args[]) {
                Pravokutnik NoviPravokutnik = new Pravokutnik();
                Scanner unos = new Scanner(System.in);
                System.out.print("Unesite dimenziju1: ");
                String ubroj = unos.next();
                int dim1 = Integer.parseInt(ubroj);
                System.out.print("Unesite dimenziju2: ");
                ubroj = unos.next();
                int dim2 = Integer.parseInt(ubroj);

                NoviPravokutnik.napraviPravokutnik(dim1,dim2);
                NoviPravokutnik.ispisiPravokutnik();
}}
```

Što će biti ako nema this??

HR 8:36 10.4.2014.

# Metode

Klasa *Pravokutnik* ima deklarirane varijable instanci *x1*, *y1*.

Metoda klase *Pravokutnik* naziva *napraviPravokutnik* ima definirane parametre identičnog naziva tj. *x1*, *y1*.

Ključna riječ this uvijek referencira (pokazuje) na objekt za koji je pozvana metoda, to znači da se this može koristiti unutar bilo koje metode da bi se ukazalo na tekući objekt.

this omogućava izravno obraćanje (referenciranje) objektu, može se iskoristiti i za rješavanje eventualnog sukoba imena između varijabli instanci i lokalnih varijabli

Java dopušta postojanje lokalnih varijabli i parametara metoda s identičnim nazivima kao što su nazivi varijabli instanci klase. Kada lokalna varijabla ima isto ime kao i varijabla instance, kaže se da lokalna varijabla *skriva* varijablu instance**.**

**this ispred naziva varijable referencira na objekt *NoviPravokutnik* za koji je pozvana metoda *napraviPravokutnik***

# Metode

Ključna riječ this se u metodama često koristi za naglašavanje razlike između varijabli instanci i lokalnih varijabli metode.

Varijable instanci predstavljaju osobine, karakteristike objekta i da njihova vrijednost čini dio stanja objekta, kao i da se definiraju izvan definicija metoda. Varijablu instance inicijalizira konstuktor, a memorijski prostor joj se dodjeljuje u trenutku kreiranja objekta i postoji onoliko dugo koliko i objekt. Ovom tipu varijable može pristupiti bilo koja metoda klase.

Lokalne varijable se definiraju unutar metoda i postoje samo ako se metode izvršavaju. Inicijalizacija ovih varijabli mora se uraditi prije njihove prve uporabe, inače kompilator javlja grešku. Lokalnim varijablama se može pristupiti samo iz metode u kojoj su definirane i one imaju određeno značenje samo tijekom izvršavanja te metode. Ove varijable najčešće sadrže neke međuvrijednosti potrebne tijekom izvršavanja metode, ali njihova vrijednost nije dio stanja objekta

# Prosljeđivanje argumenata

**U osnovi 2 načina:**

- **po vrijednosti (call-by-value)**

  **vrijednost argumenta se kopira u formalni parametar metode, promjene izvedene na parametru u metodi nemaju utjecaja na argument korišten za njegovo pozivanje**

- **po referenci (call-by-reference)**

  **parametru se prosljeđuje referenca na argument, a ne njegova vrijednost, a koristi se unutar metode za pristupanje stvarnom argumentu zadanom u pozivu.**

# Prosljeđivanje argumenata
## po vrijednosti (call-by-value)

- **za primitivne tipove parametara (brojeve, Boolean i sl.)**

**Metoda prima kopiju svih vrijednosti parametara i ona ne može promijeniti njihov sadržaj**

**Primjer:**

```
public static void utrostruči (double x)
{ x = 3 * x; }
Poziv:
….
double postotak = 10;
utrostruči(postotak);
```

# Prosljeđivanje argumenata
## po vrijednosti (call-by-value)

**Redoslijed događanja:**

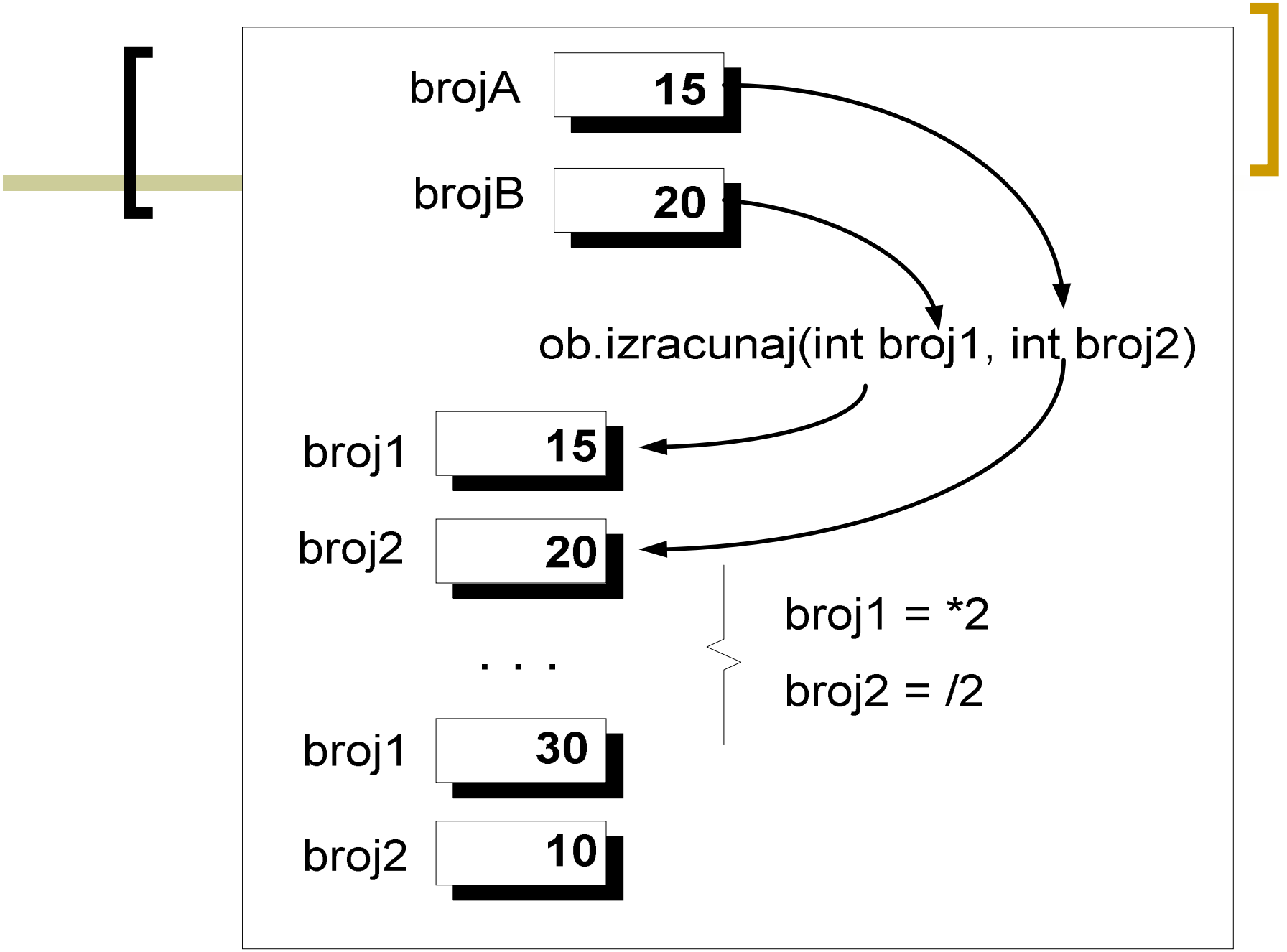
1. **“x” se inicijalizira s kopijom vrijednosti “postotak” (x= 10)**
2. **“x” se utrostruči (x=30), ali “postotak” je i dalje 10**
3. **Metoda završava i parametarska varijabla “x” više nije u uporabi.**

PozivPoVrijednosti.java – Blok za pisanje

Datoteka Uređivanje Formatiranje Prikaz Pomoć

```
// Jednostavni tipovi se prosljeđuju po vrijednosti.
import java.util.Scanner;

class Test {
  void racunaj(int a, int b) {
    a *= 2;
    b /= 2;
  System.out.println("a i b iz metode: " +
                     a + " " + b);
  }
}

class PozivPoVrijednosti {
  public static void main(String args[]) {
      Test ob = new Test();
      int a, b;

      Scanner unos = new Scanner(System.in);
      System.out.print("Unesite a ");
      String ubroj = unos.next();
      a = Integer.parseInt(ubroj);
      System.out.print("Unesite b ");
      ubroj = unos.next();
      b = Integer.parseInt(ubroj);

    System.out.println("a i b prije poziva metode: " +
                     a + " " + b);

    ob.racunaj(a, b);

    System.out.println("a i b nakon poziva metode: " +
                     a + " " + b);
  }
}
```

HR 8:57 10.4.2014.


brojA
15
brojB
20
ob.izracunaj(int broj1, int broj2)
broj1
15
broj2
20
broj1 = *2
broj2 = /2
. . .
broj1
30
broj2
10

# Prosljeđivanje argumenata
# po referenci (call-by-reference)

- **za prosljeđivanje objekata**

**Parametri koji predstavljaju objekte u osnovi sadrže samo referencu (put do) objekta.**

**Kada se ovo proslijedi metodi, parametar koji primi referencu ukazuje na isti objekt.**

**Primjer:**

```
public static void utrostruči(Djelatnik x)
{ x.povećajPlaću(300); }
Poziv:
Djelatnik Mate = new Djelatnik(…):
utrostruči(Mate);
```

# Prosljeđivanje argumenata
# po referenci (call-by-reference)

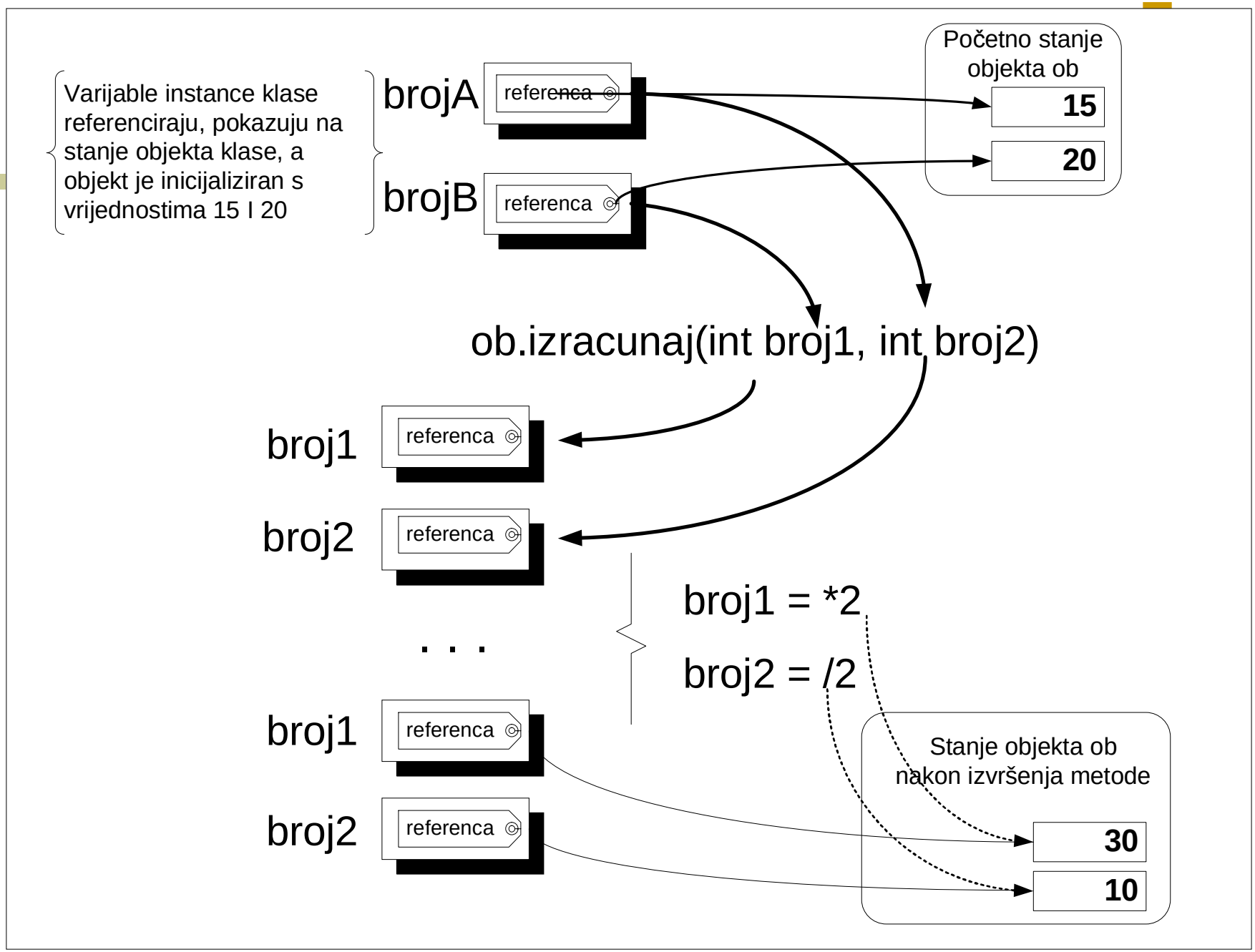
**Redoslijed događanja:**

1. **“x” se inicijalizira kopijom vrijednosti “Mate”, tj. referencom na objekt**
2. **Metoda povećajPlaću se primjenjuje na tu referencu objekta. Objekt “Djelatnik” na kojega i “x” i “Mate” pokazuju (referenciraju) dobiva plaću uvećanu za 300%.**
3. **Metoda završava i parametarska varijabla “x” nije više u uporabi, ali objektna varijabla “Mate” i dalje pokazuje na objekt čija je plaća utrostručena.**

PozivPoReferenci.java – Blok za pisanje

Datoteka Uređivanje Formatiranje Prikaz Pomoć

```
class Test {
  int a, b;

  Test(int i, int j) {
    a = i;
    b = j;
     }

  // proslijedi objekt
  void racunaj(Test ob) {
    ob.a *=  2;
    ob.b /= 2;
    System.out.println("a i b unutar metode: " +
                       ob.a + " " + ob.b);
  }
}
class PozivPoReferenci {
  public static void main(String args[]) {

      int a, b;

      Scanner unos = new Scanner(System.in);
      System.out.print("Unesite a ");
      String ubroj = unos.next();
      a = Integer.parseInt(ubroj);
      System.out.print("Unesite b ");
      ubroj = unos.next();
      b = Integer.parseInt(ubroj);

      Test ob = new Test(a,b);

    System.out.println("ob.a i ob.b prije poziva: " +
                       ob.a + " " + ob.b);

    ob.racunaj(ob);

    System.out.println("ob.a i ob.b nakon poziva: " +
                       ob.a + " " + ob.b);
  }
}
```


Varijable instance klase referenciraju, pokazuju na stanje objekta klase, a objekt je inicijaliziran s vrijednostima 15 I 20
brojA
referenca
brojB
referenca
Početno stanje objekta ob
15
20
ob.izracunaj(int broj1, int broj2)
broj1
referenca
broj2
referenca
. . .
broj1 = *2
broj2 = /2
broj1
referenca
broj2
referenca
Stanje objekta ob nakon izvršenja metode
30
10

# Privatne metode

**Iako su metode većinom javne (public) mogu se definirati i kao privatne što znači da se mogu pozvati samo iz drugih metoda iste klase.**

**Kada se metoda proglasi privatnom ona ne mora biti raspoloživa ako dođe do promjene implementiranja.**

**BITNO: dokle god je metoda privatna programer može biti siguran da se neće nikada koristiti izvan operacije te klase.**

**Kada se odlučiti za privatnu metodu:**

- **ako metoda nije bitna za korisnika klase**
- **Ako se metoda ne može jednostavno podržati ako dođe do promjene u implementiranju klase (specifični protokoli i sl.)**

# Preopterećivanje konstruktora i metoda (constructor/method overloading)

**Bitna osobina programskih jezika jeste uporaba imena.**

**Problem: preslikavanje nijansi svojstvenih ljudskim jezicima u programski jezik**
**(homonim - jedna riječ više značenja).**

**Primjeri: ženska kosa, kosa crta, alatka kosa …**
**oprati košulju, oprati auto, oprati psa …**
**(Čekaj me dok dođem / čekaj me dok ne dođem)**
**Programski jezik: opratiKošulju košulju**
**opratiAuto auto**
**opratiPsa psa**

# Preopterećivanje konstruktora i metoda (constructor/method overloading)

**Programski jezici zahtijevaju postojanje JEDINSTVENIH IDENTIFIKATORA za svaku funkciju.**

**JAVA**

**Konstruktori zahtijevaju mogućnost “preopterećenja” metoda.**

**RAZLOG: ime konstruktora je određeno imenom klase, što znači da može biti samo jedno. Ali, postoji potreba kreiranja objekta na više načina.**

# Preopterećenje konstruktora

```
public class Student7
{
 public static void main(String[] args)
 {
        student[] StudentR = new student[3];

        StudentR[0] = new student("Ana","Anić",1000);
        StudentR[1] = new student(1500);
        StudentR[2] = new student();


        for(int i=0; i<StudentR.length; i++)
        {
          student s = StudentR[i];
          System.out.println("Student "+s.uzmiime()+" "+s.uzmiprezime()+" "+"školarina
        "+s.uzmiskolarina());
       }
 }
}
```

```
class student
 { //prvi konstruktor
   public student(String i, String p, double s)
   { ime = i;
     prezime = p;
     skolarina = s;   }
   // drugi konstruktor
   public student(double s)
   {     this("konstruktor","2",s);    }
   // treći konstruktor - inicijalni
   public student()
    { /* inicijalne vrijednosti */ }
   public String uzmiime()
   { return ime; }
   public String uzmiprezime()
   { return prezime; }
   public String uzmitip()
   {return tip; }
   public double uzmiskolarina()
   {return skolarina; }
 private String ime, prezime, tip;
 private double skolarina;
   }
```

# Preopterećenje konstruktora

```
class Mini extends Car {

    Color color;

    public Mini() {
        this(Color.Red);
    }

    public Mini(Color c) {
        super("Mini");
        color = c;
        // more initialization
    }

    public Mini(int size) {
        this(Color.Red);
        super(size);

    }
}
```

The no-arg constructor supplies a default Color and calls the overloaded Real Constructor (the one that calls super()).

This is The Real Constructor that does The Real Work of initializing the object (including the call to super())

Won't work!! Can't have super() and this() in the same constructor, because they each must be the first statement!

# Ključna riječ: **this**

**Ako postoji nekoliko konstruktora za klasu, postoji potreba za pozivom jednog konstruktora iz drugog kako bi se izbjegao dupli kod.**

- ⇒**uporaba this operatora**
- ⇒**drugačije nego kod standardne uporabe**
- ⇒**znači eksplicitni poziv konstruktoru koji odgovara listi argumenata, što osigurava izravan način za poziv drugih konstruktora**
- ⇒**THIS mora biti prvi izraz u konstruktoru**

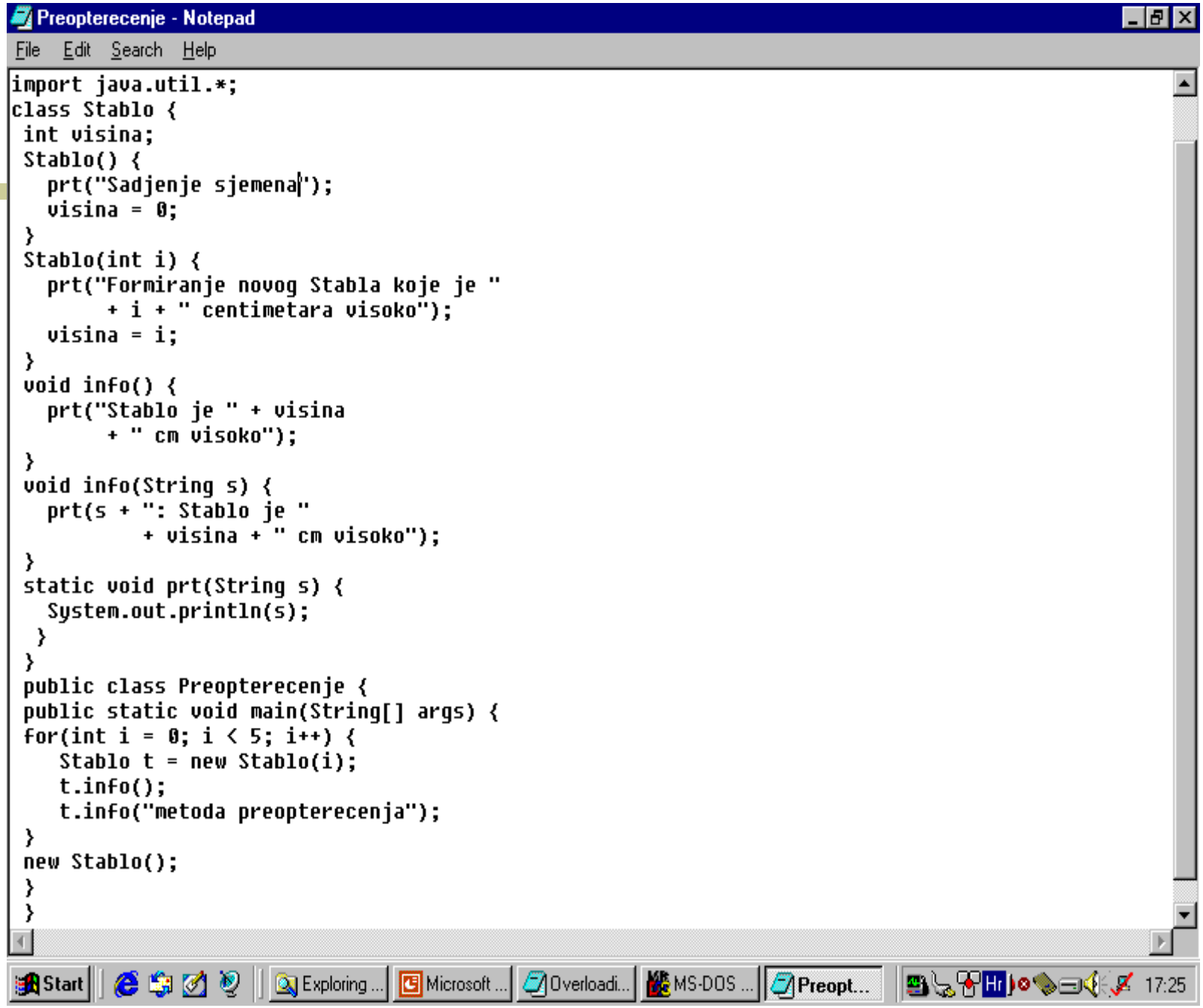
# Preopterećivanje metoda (method overloading)

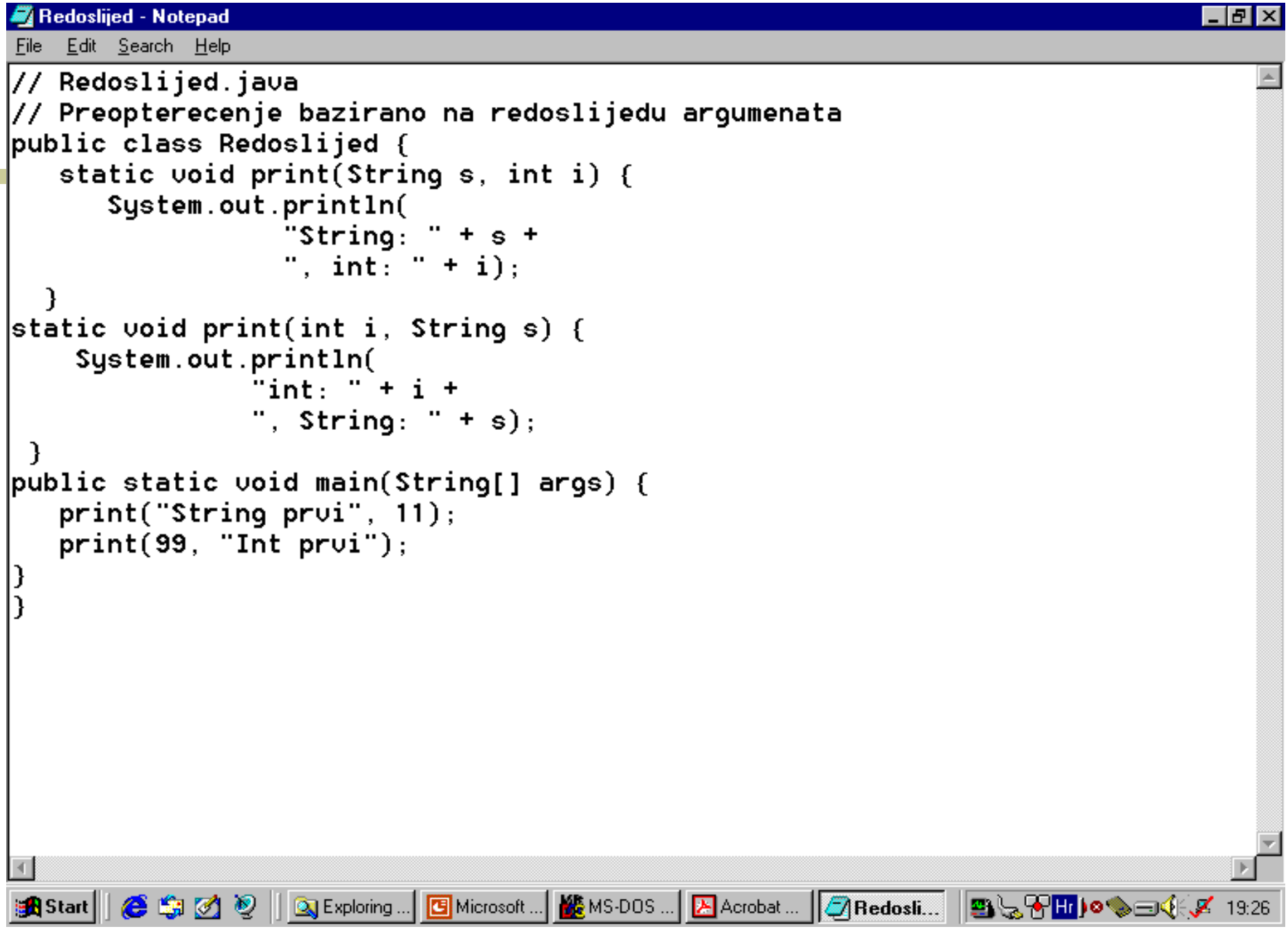
**RAZLIKOVANJE**

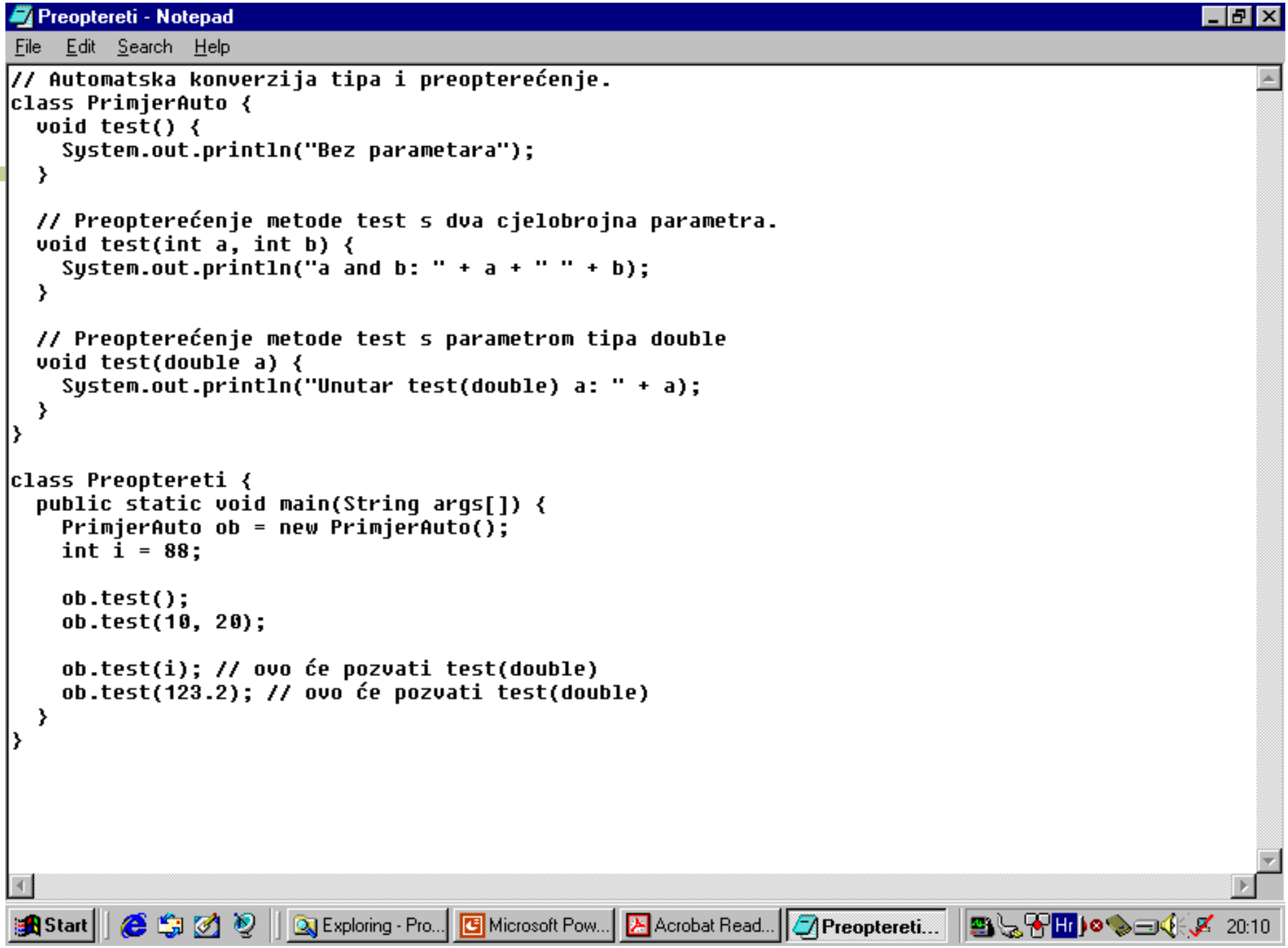
- **po tipu  i/ili**
- **po broju argumenata**
- **po redoslijedu (ne preporučuje se)**

**Povratni tip podataka nije dovoljan za razlikovanje metoda.**

**Slaganje ne mora biti potpuno točno – tu igra ulogu i automatska konverzija podataka.**


Preopterecenje - Notepad
File Edit Search Help
import java.util.*;
class Stablo {
int visina;
Stablo() {
prt("Sadjenje sjemena");
visina = 0;
}
Stablo(int i) {
prt("Formiranje novog Stabla koje je "
+ i + " centimetara visoko");
visina = i;
}
void info() {
prt("Stablo je " + visina
+ " cm visoko");
}
void info(String s) {
prt(s + ": Stablo je "
+ visina + " cm visoko");
}
static void prt(String s) {
System.out.println(s);
}
}
public class Preopterecenje {
public static void main(String[] args) {
for(int i = 0; i < 5; i++) {
Stablo t = new Stablo(i);
t.info();
t.info("metoda preopterecenja");
}
new Stablo();
}
}
Start
Exploring ...
Microsoft ...
Overloadi...
MS-DOS ...
Preopt...
17:25


Redoslijed - Notepad
File Edit Search Help
// Redoslijed.java
// Preopterecenje bazirano na redoslijedu argumenata
public class Redoslijed {
static void print(String s, int i) {
System.out.println(
"String: " + s +
", int: " + i);
}
static void print(int i, String s) {
System.out.println(
"int: " + i +
", String: " + s);
}
public static void main(String[] args) {
print("String prvi", 11);
print(99, "Int prvi");
}
}
Start
Exploring ...
Microsoft ...
MS-DOS ...
Acrobat ...
Redosli...
19:26


Preoptereti - Notepad
File Edit Search Help
// Automatska konverzija tipa i preopterećenje.
class PrimjerAuto {
void test() {
System.out.println("Bez parametara");
}
// Preopterećenje metode test s dva cjelobrojna parametra.
void test(int a, int b) {
System.out.println("a and b: " + a + " " + b);
}
// Preopterećenje metode test s parametrom tipa double
void test(double a) {
System.out.println("Unutar test(double) a: " + a);
}
}
class Preoptereti {
public static void main(String args[]) {
PrimjerAuto ob = new PrimjerAuto();
int i = 88;
ob.test();
ob.test(10, 20);
ob.test(i); // ovo će pozvati test(double)
ob.test(123.2); // ovo će pozvati test(double)
}
}
Start
Exploring - Pro...
Microsoft Pow...
Acrobat Read...
Preoptereti...
20:10

# PITANJA