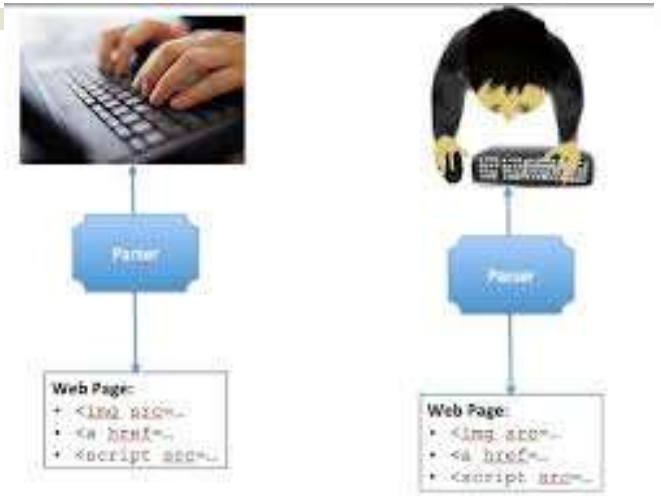
Programiranje



Nastava: prof.dr.sc. Dražena Gašpar

Datum: 24.04.2018.

Domaća zadaća za 24.04.2018.

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

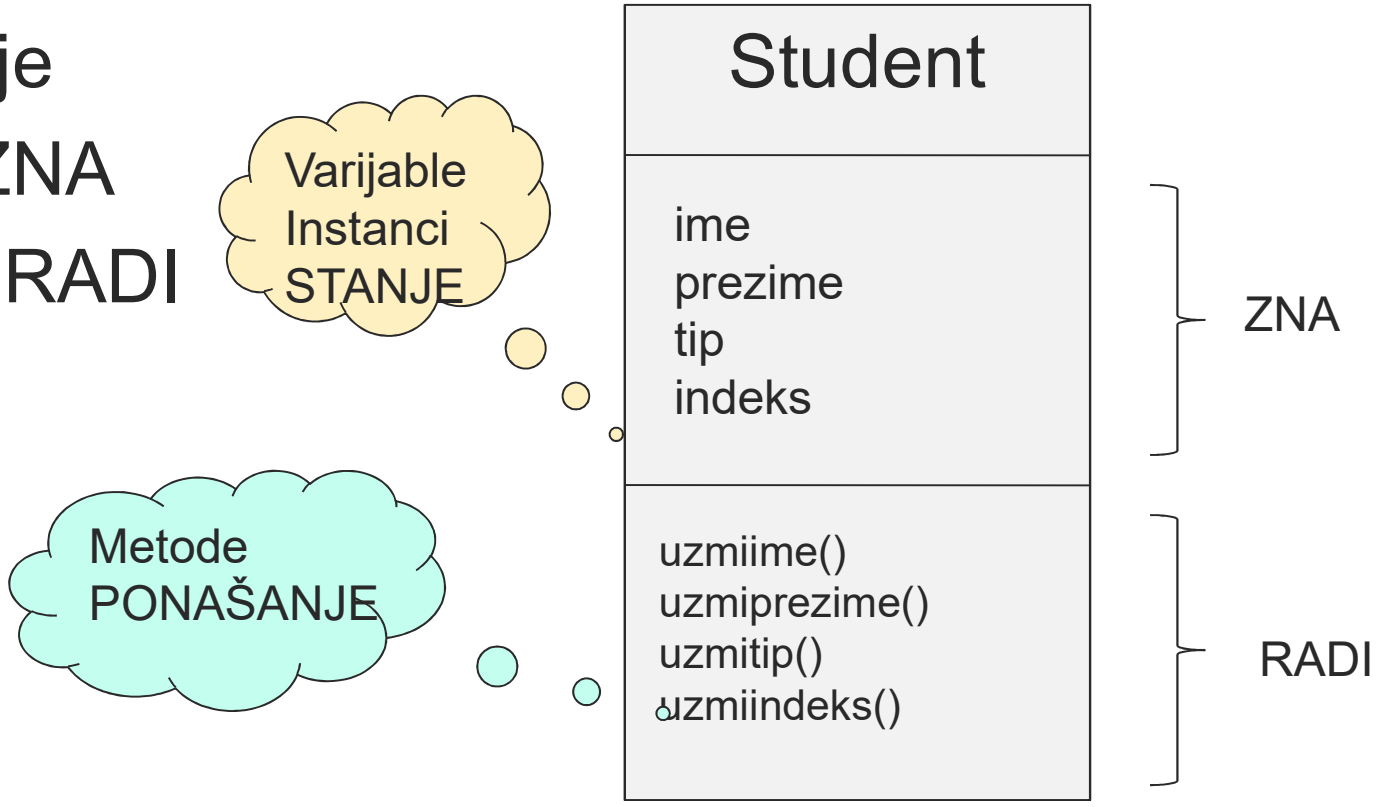
Metode

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



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:

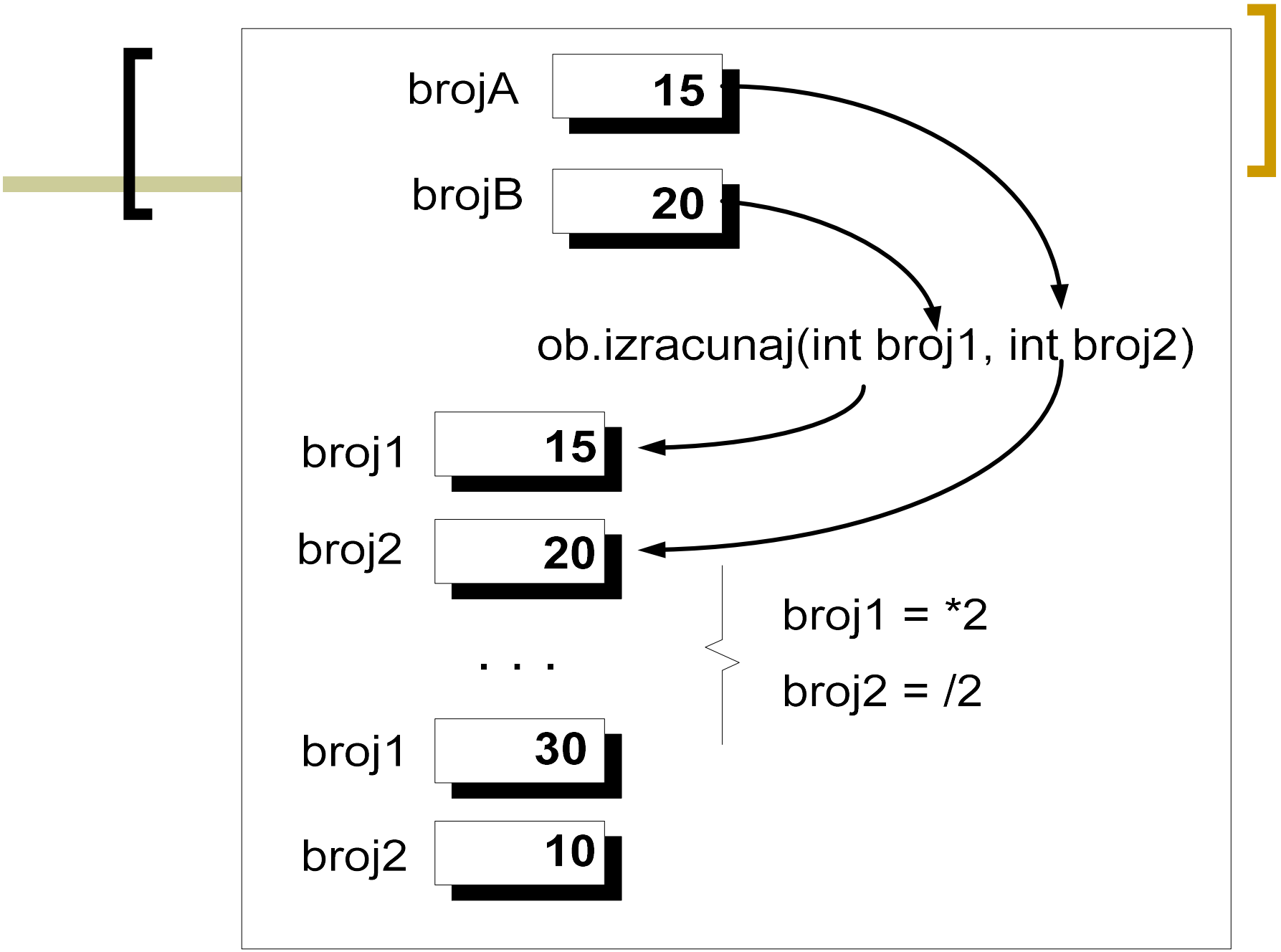
1. **“x” se inicijalizira s kopijom vrijednosti “postotak” (x= 10)**
2. **“x” se utrostruči (x=30), ali “postotak” je i dalje 10**
3. **Metoda završava i parametarska varijabla “x” više nije u uporabi.**

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:

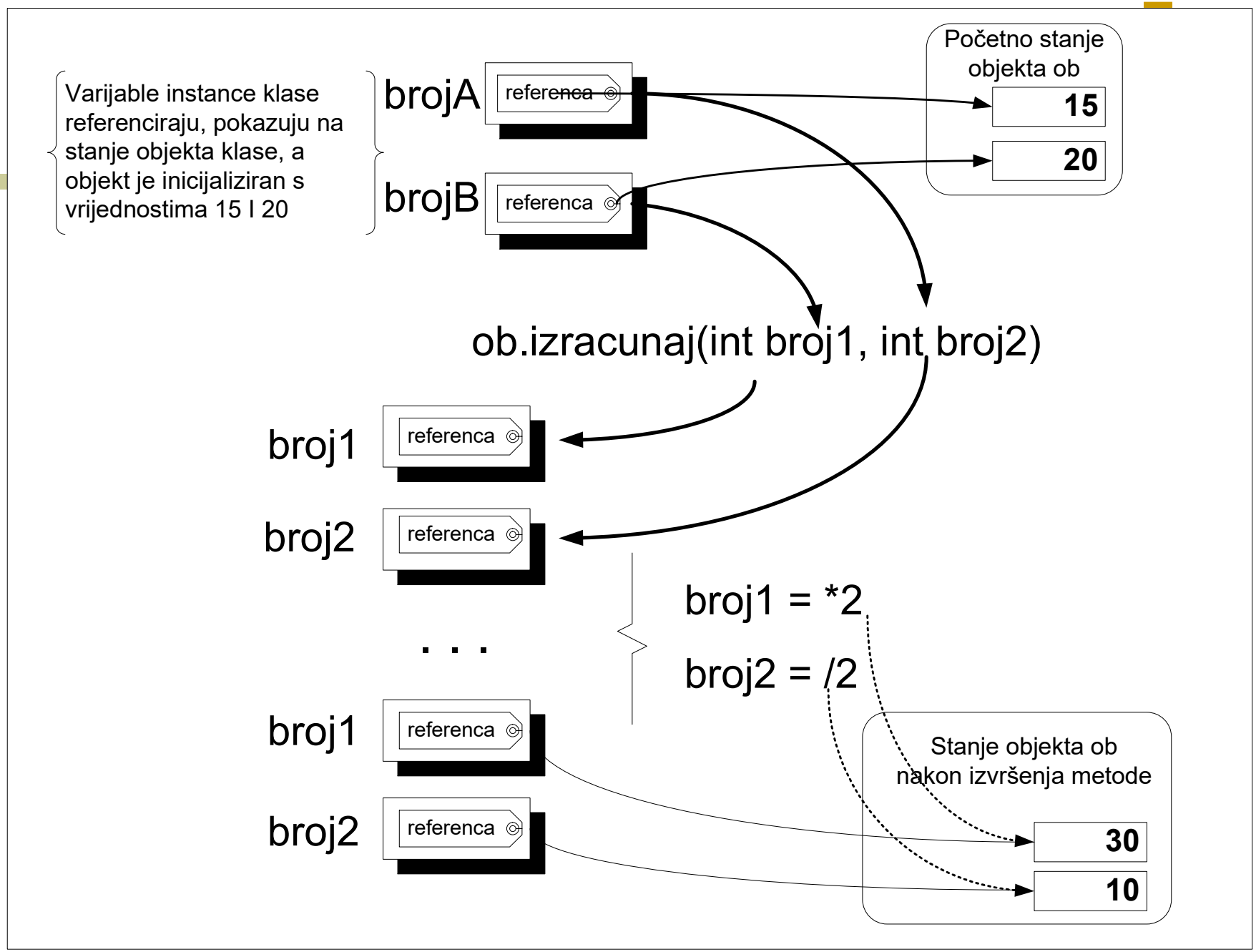
1. **“x” se inicijalizira kopijom vrijednosti “Mate”, tj. referencom na objekt**
2. **Metoda povećajPlaću se primjenjuje na tu referencu objekta. Objekt “Djelatnik” na kojega i “x” i “Mate” pokazuju (referenciraju) dobiva plaću uvećanu za 300%.**
3. **Metoda završava i parametarska varijabla “x” nije 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);
  }
}
```

HR 9:20 10.4.2014.


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

lako 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**

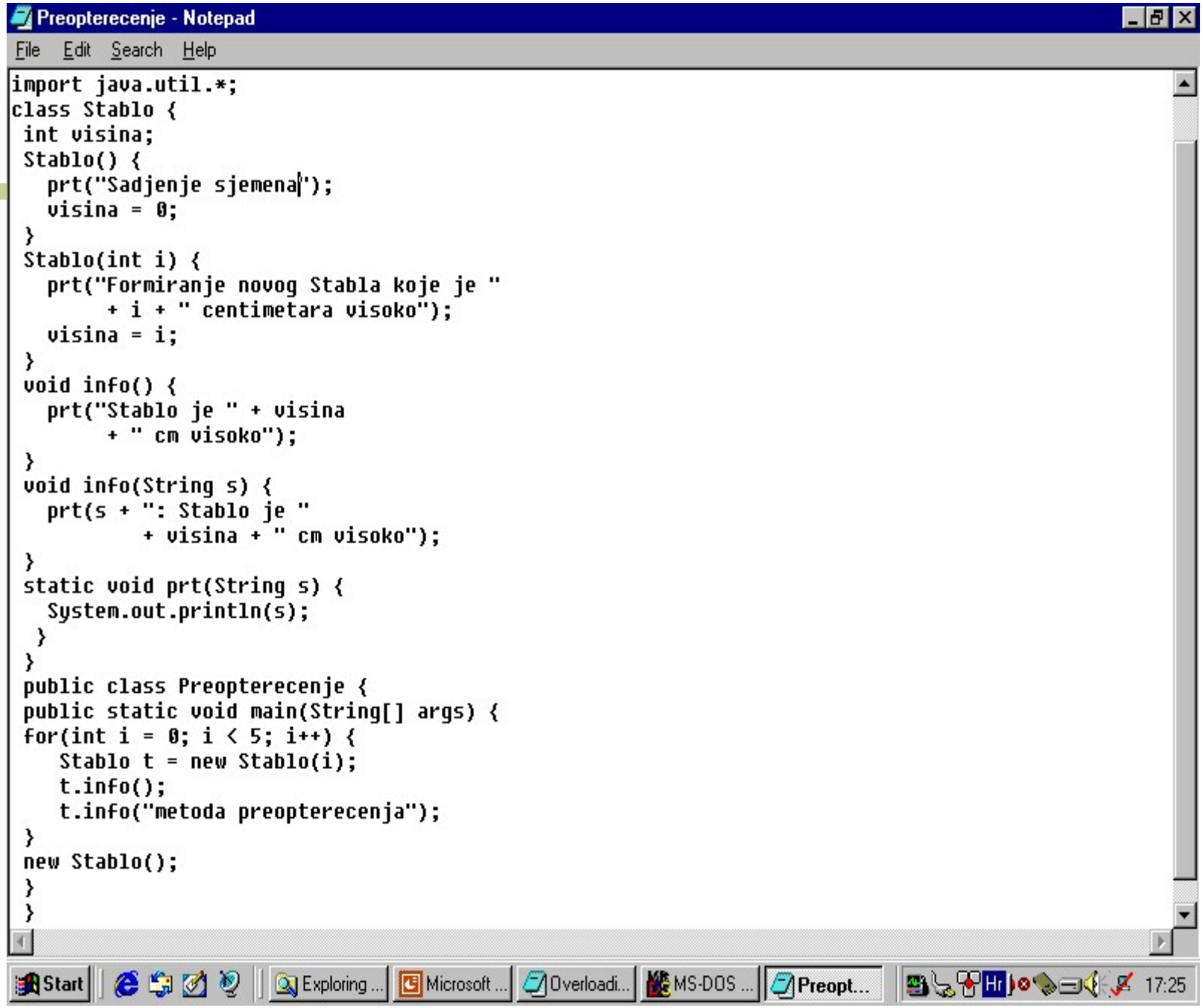
Preopterećivanje metoda (method overloading)

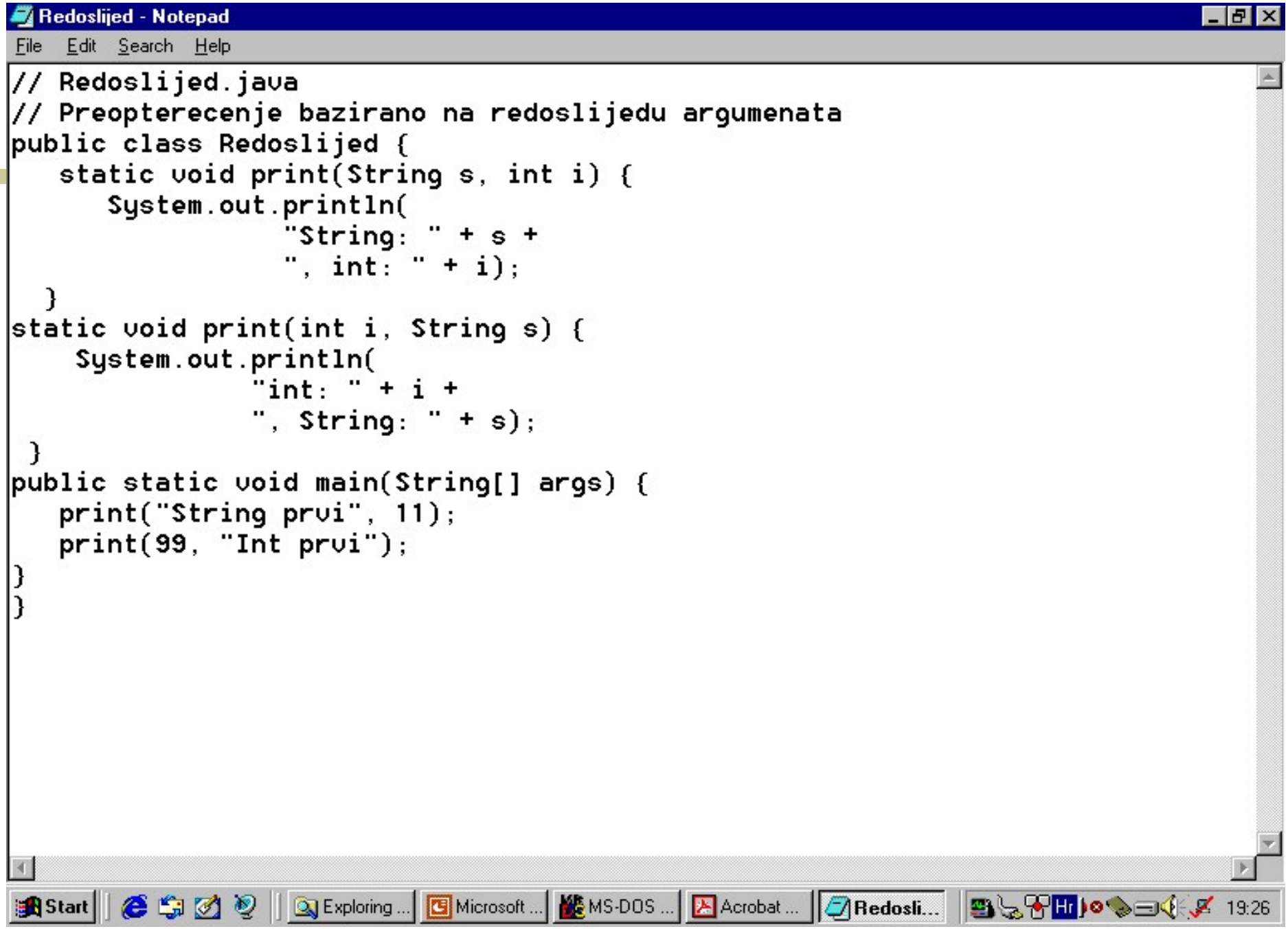
RAZLIKOVANJE

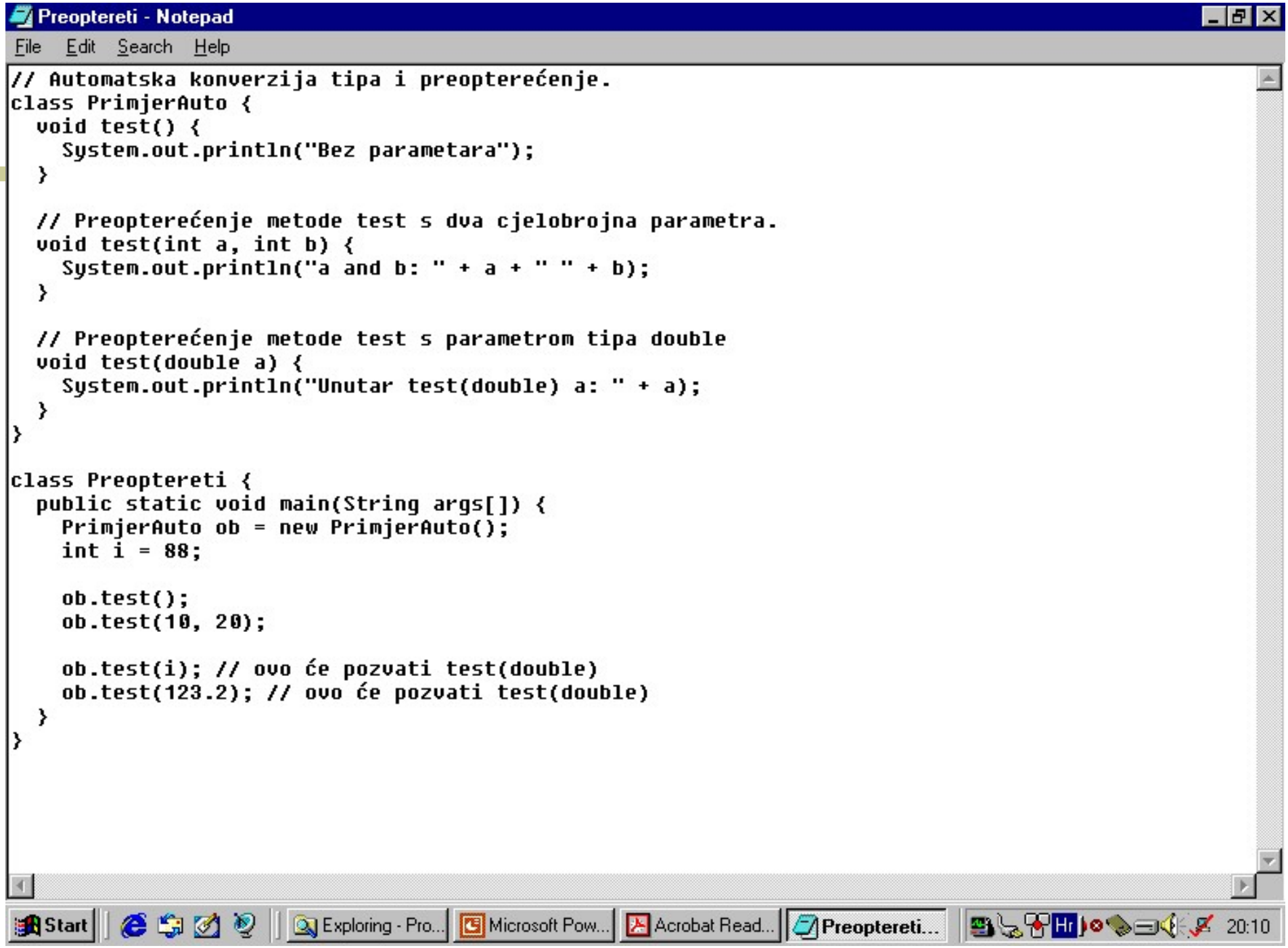
- **po tipu i/ili**
- **po broju argumenata**
- **po redoslijedu (ne preporučuje se)**

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

Zadatak

Formirati polje dimenzije [3] naziva Izvanredni

Popuniti s tri objekta tipa student

Klasi student dodati školarinu

Dodati metodu kojoj se parametarski šalje postotak uvećanja školarine

Program:Student6.java

Nasljeđivanje

Tri metafore nasljeđivanja:

- ***biološka*: dijete nasljeđuje gene od oba svoja roditelja (dijeli neke crte fizičke/karakterne i sposobnosti roditelja)**
- ***biološka*: kreiranje globalne strukture na osnovu relacije "je vrsta od" (taksonomija)**
- ***ekonomska*: srodnici nasljeđuju resurse svojih roditelja.**

Nasljeđivanje

“nadređeno-podređena relacije između klasa u kojoj podređena (dijete) klasa ima isto ponašanje (odgovornost) kao nadređena (roditelj) klasa plus najmanje još jedno dodatno“

West, D. „Object thinking“, Microsoft Press, USA, 2004

Nasljeđivanje

Nasljeđivanje je mehanizam pomoću kojega se osigurava da jedna klasa (proširena [engl. extended] ili podklasa[engl. subclass]) automatski preuzima (posjeduje) sve osobine i ponašanja, osim onih definiranih kao privatni (engl. private), od druge klase (nadklase[engl. superclass] ili „roditelj“ klase[engl. parent class]).

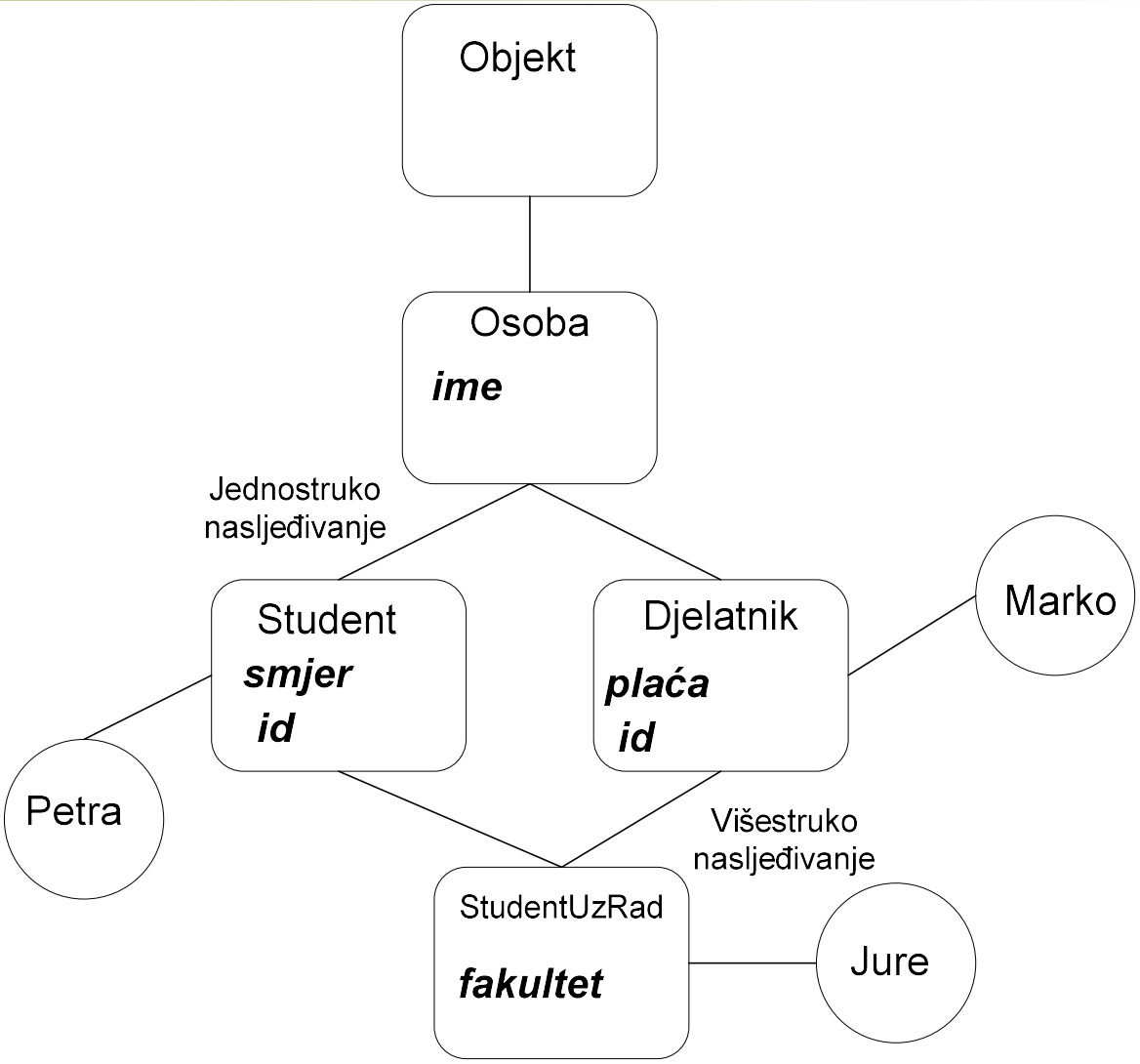
Nasljeđivanje omogućuje pravljenje opće klase koja definira zajedničke osobine i ponašanja za skup srodnih klasa.

Nasljeđivanje

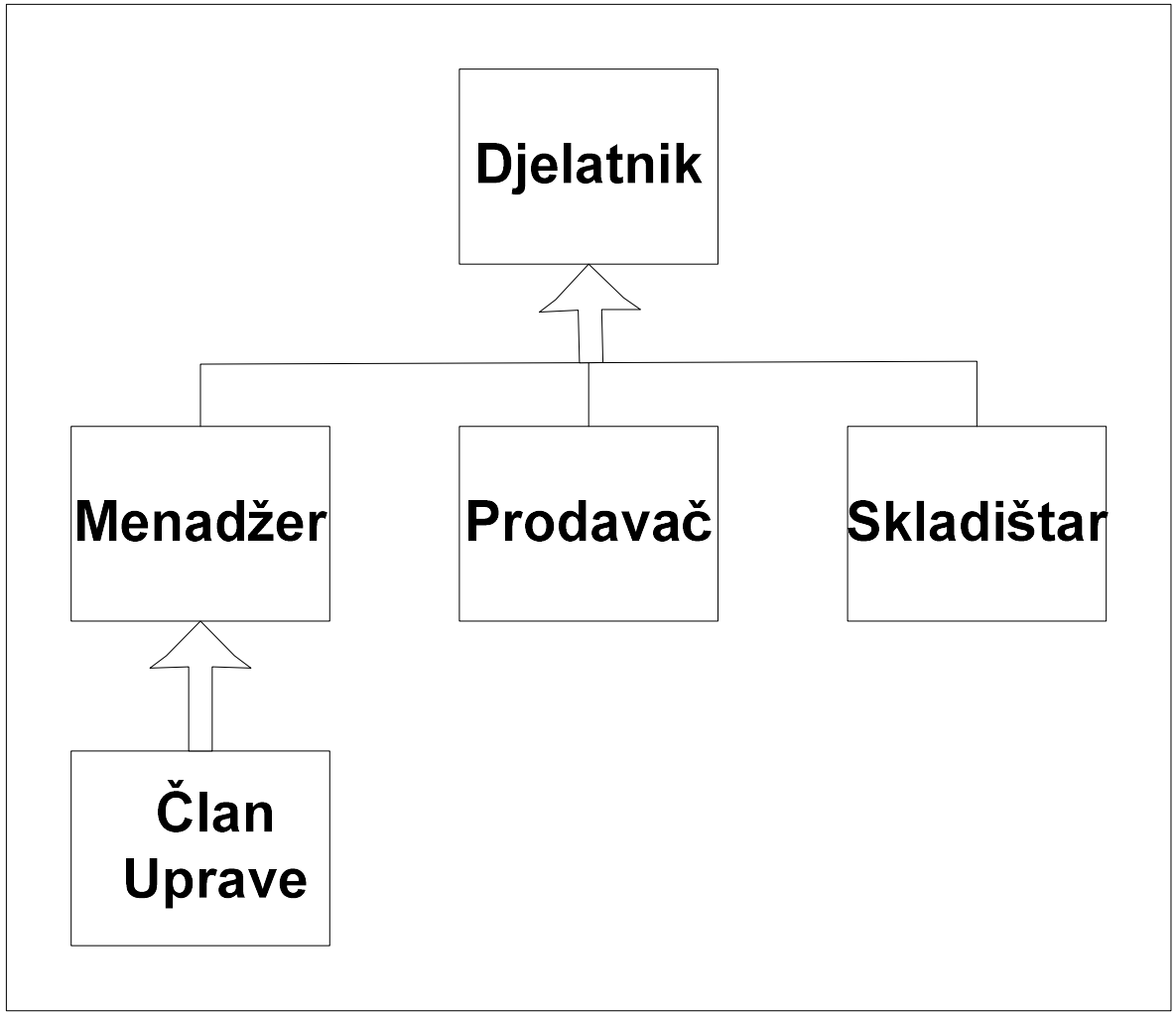
2 tipa nasljeđivanja:

- Jednostruko
- Višestruko

Nasljeđivanje



Nasljeđivanje



Nasljeđivanje

- **omogućava izradu hijerarhija klasa**
- **omogućava stvaranje jedne općenite klase koja definira neka od osnovnih ponašanja objekta, a konkretni objekti mogu nasljeđivati njena svojstva i nadopunjavati ih**
- **NADKLASA**
- **PODKLASA**

```
class imepodklase extends imenadklase {
// tijelo klase }
```

Nasljeđivanje

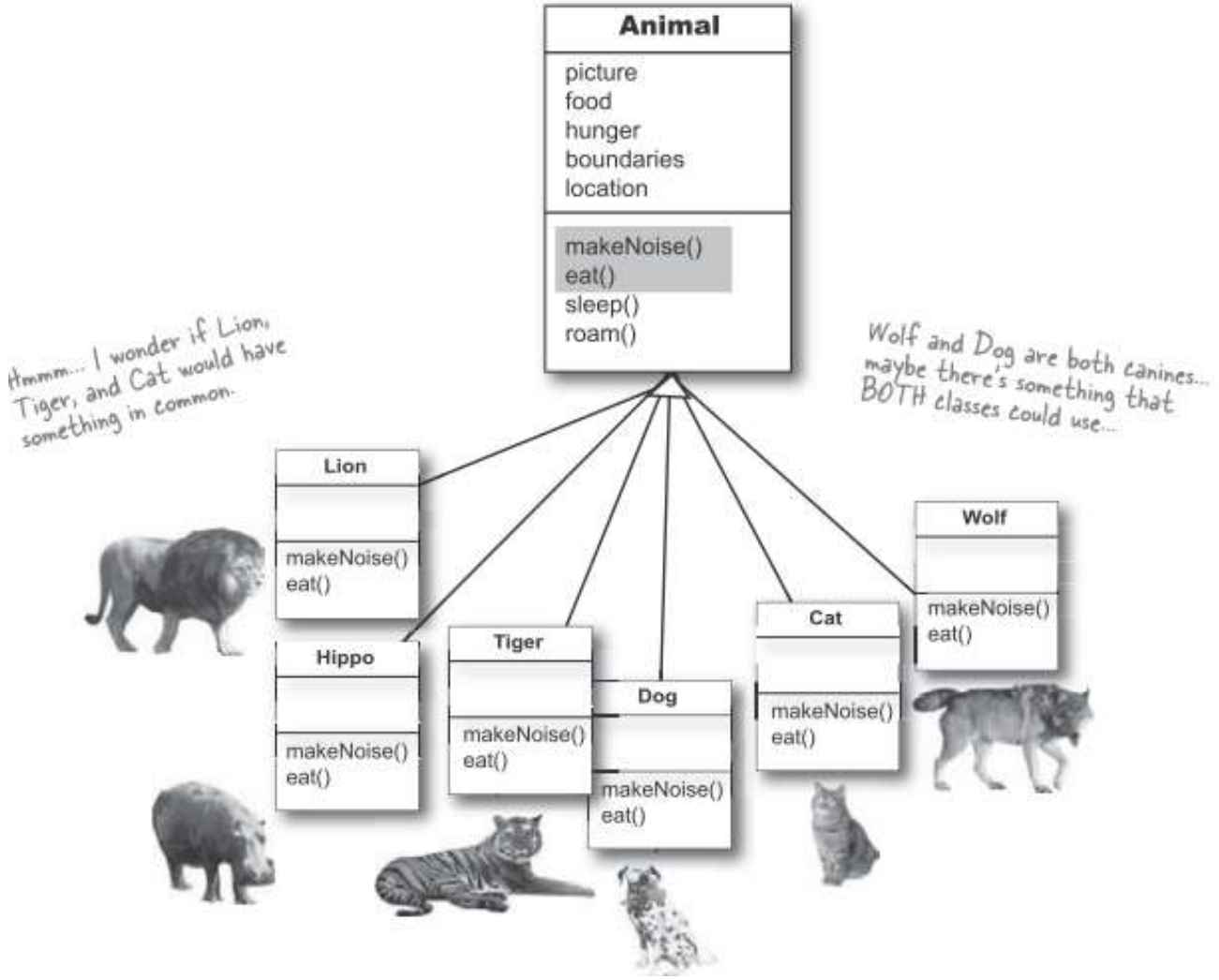
Osnovna pravila:

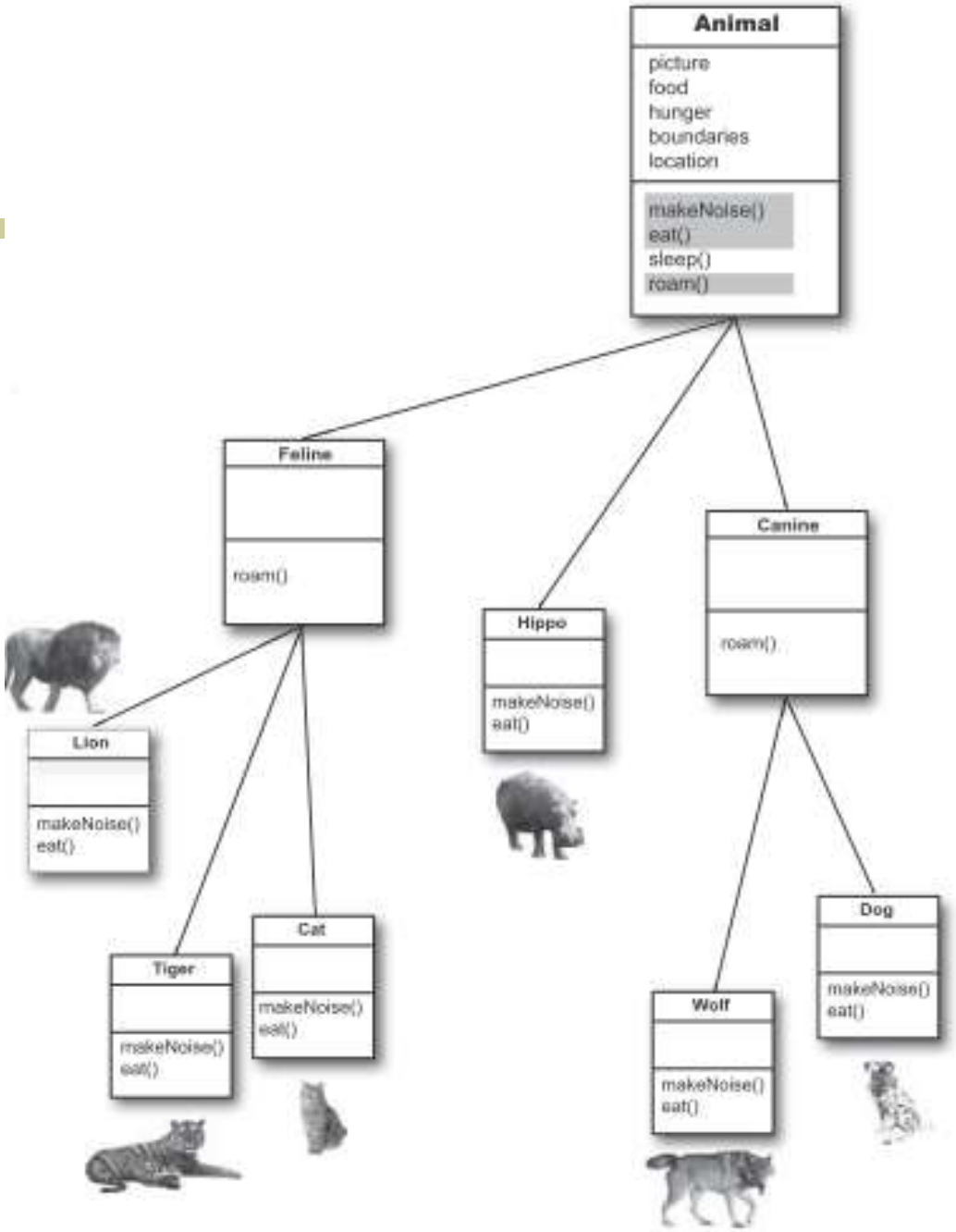
PODKLASA može imati samo jednu NADKLASU !!

NADKLASA može imati više PODKLASA !!

Nijedna klasa ne može biti vlastita NADKLASA.

Nasljeđivanje




Animal
picture
food
hunger
boundaries
location
makeNoise()
eat()
sleep()
roam()
Feline
roam()
Hippo
makeNoise()
eat()
Canine
roam()
Lion
makeNoise()
eat()
Tiger
makeNoise()
eat()
Cat
makeNoise()
eat()
Wolf
makeNoise()
eat()
Dog
makeNoise()
eat()

Poziv metoda

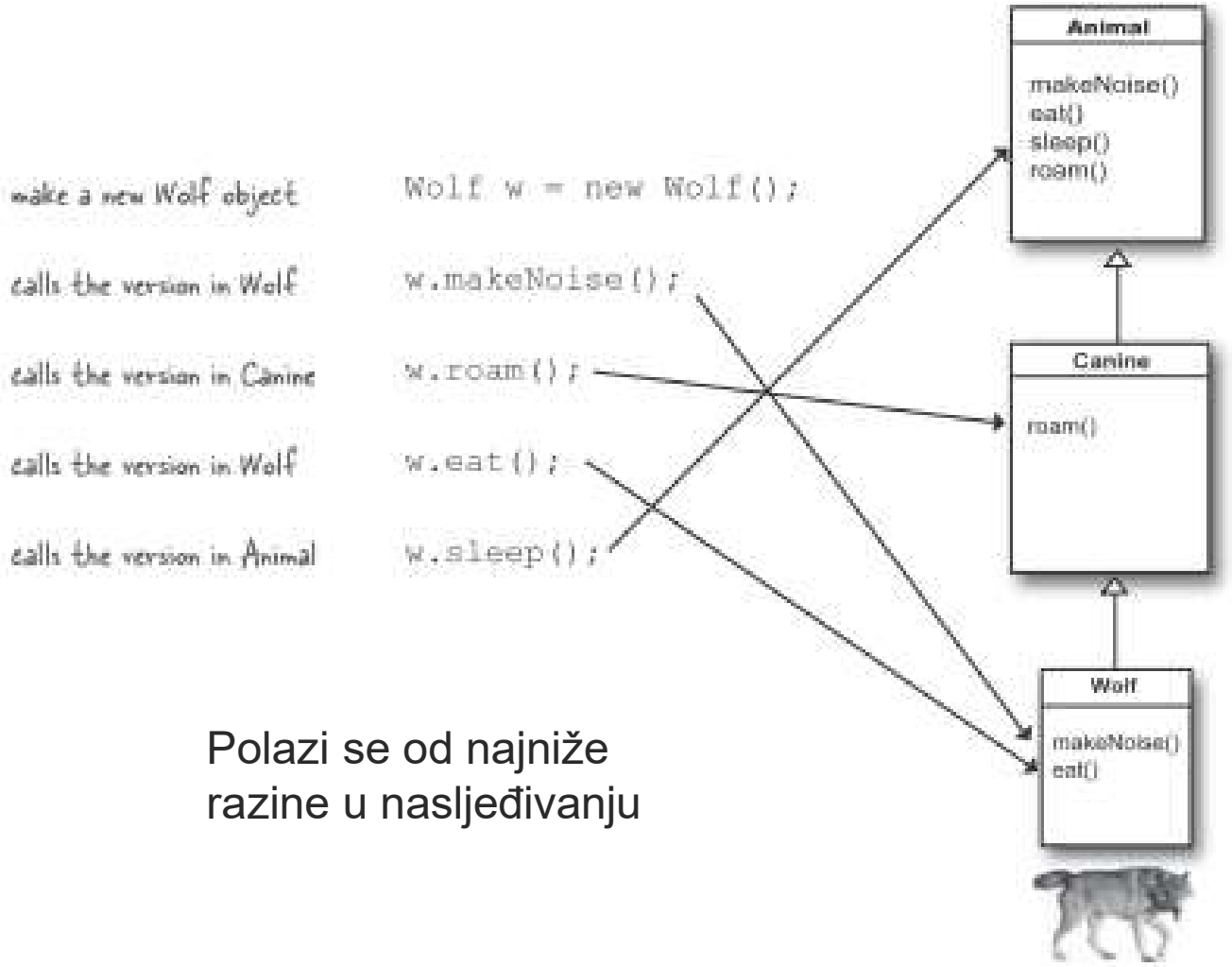
make a new Wolf object — `Wolf w = new Wolf();`

calls the version in Wolf — `w.makeNoise();`

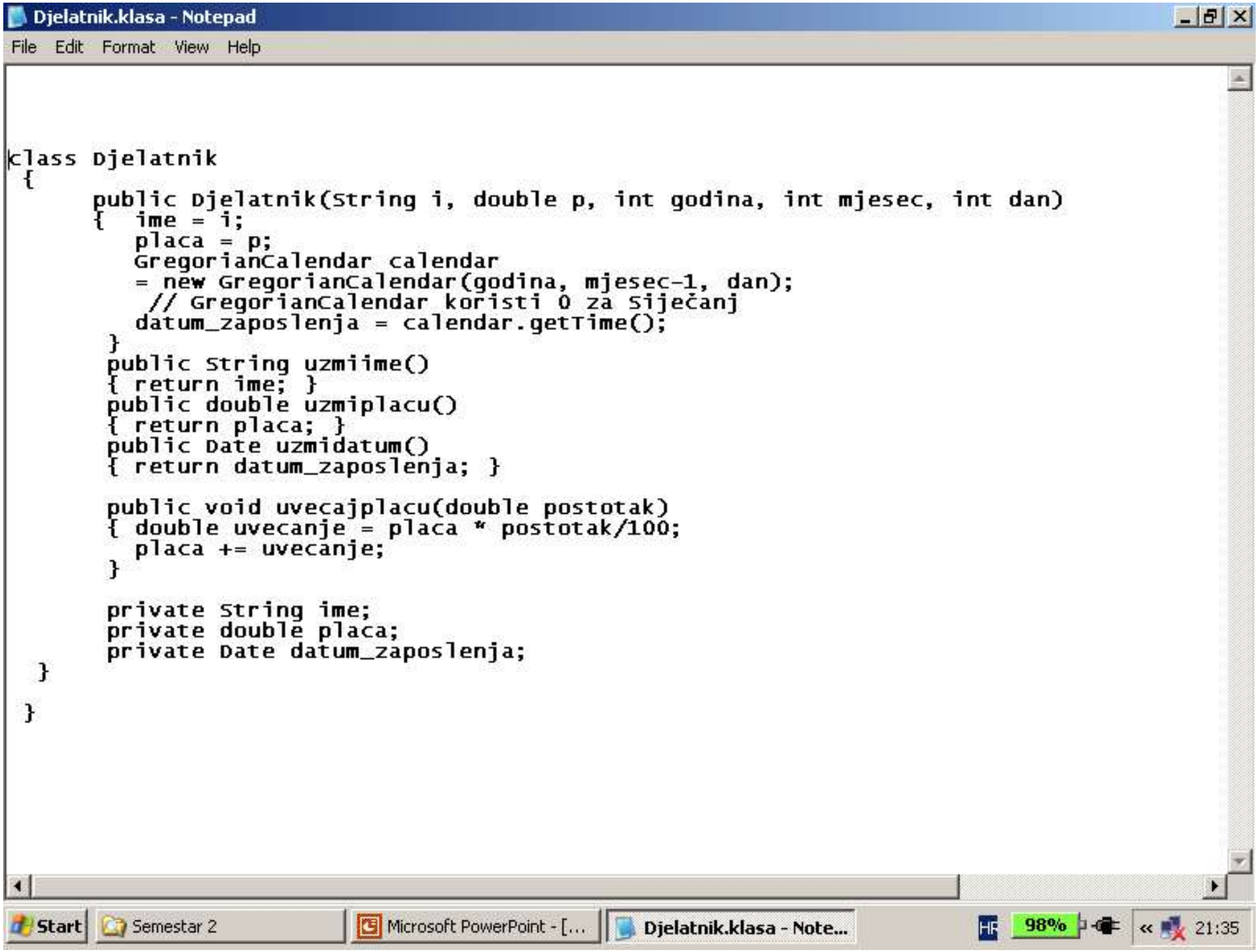
calls the version in Canine — `w.roam();`

calls the version in Wolf — `w.eat();`

calls the version in Animal — `w.sleep();`



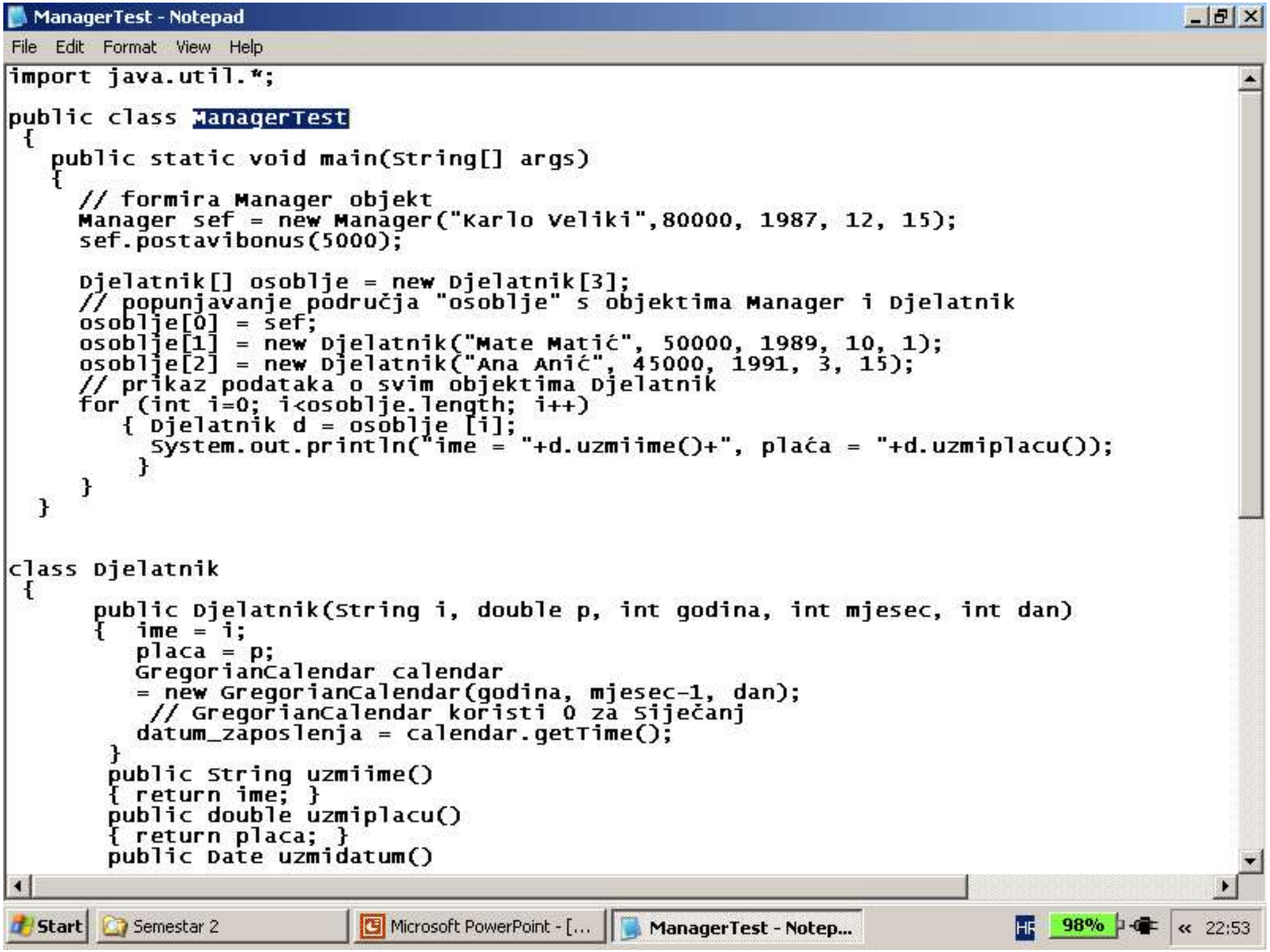
Polazi se od najniže razine u nasljeđivanju


Djelatnik.klasa - Notepad
File Edit Format View Help
class Djelatnik
{
public Djelatnik(String i, double p, int godina, int mjesec, int dan)
{ ime = i;
placa = p;
GregorianCalendar calendar
= new GregorianCalendar(godina, mjesec-1, dan);
// GregorianCalendar koristi 0 za Siječanj
datum_zaposlenja = calendar.getTime();
}
public String uzmiime()
{ return ime; }
public double uzmiplacu()
{ return placa; }
public Date uzmidatum()
{ return datum_zaposlenja; }
public void uvecajplacu(double postotak)
{ double uvecanje = placa * postotak/100;
placa += uvecanje;
}
private String ime;
private double placa;
private Date datum_zaposlenja;
}
}
Start
Semestar 2
Microsoft PowerPoint - [...
Djelatnik.klasa - Note...
HR
98%
21:35

Nasljeđivanje

- Menadžeri jesu radnici, ali imaju bonus na plaću

```
class Manager extends Djelatnik
  { // dodavanje metoda i polja //
   }
```


ManagerTest - Notepad
File Edit Format View Help
import java.util.*;
public class ManagerTest
{
public static void main(String[] args)
{
// formira Manager objekt
Manager sef = new Manager("Karlo Veliki",80000, 1987, 12, 15);
sef.postavibonus(5000);
Djelatnik[] osoblje = new Djelatnik[3];
// popunjavanje područja "osoblje" s objektima Manager i Djelatnik
osoblje[0] = sef;
osoblje[1] = new Djelatnik("Mate Matić", 50000, 1989, 10, 1);
osoblje[2] = new Djelatnik("Ana Anić", 45000, 1991, 3, 15);
// prikaz podataka o svim objektima Djelatnik
for (int i=0; i<osoblje.length; i++)
{ Djelatnik d = osoblje [i];
System.out.println("ime = "+d.uzmiime()+", plaća = "+d.uzmiplacu());
}
}
}
class Djelatnik
{
public Djelatnik(String i, double p, int godina, int mjesec, int dan)
{ ime = i;
placa = p;
GregorianCalendar calendar
= new GregorianCalendar(godina, mjesec-1, dan);
// GregorianCalendar koristi 0 za Siječanj
datum_zaposlenja = calendar.getTime();
}
public String uzmiime()
{ return ime; }
public double uzmiplacu()
{ return placa; }
public Date uzmidatum()
Start
Semestar 2
Microsoft PowerPoint - [...
ManagerTest - Notep...
HR
98%
« 22:53

ManagerTest - Notepad

File Edit Format View Help

```
        { return datum_zaposlenja; }

        public void uvecajplacu(double postotak)
        { double uvecanje = placa * postotak/100;
          placa += uvecanje;
        }

        private String ime;
        private double placa;
        private Date datum_zaposlenja;

 }

class Manager extends Djelatnik
 {
   public Manager(String i, double p, int godina, int mjesec, int dan)
   {
     super(i, p, godina, mjesec, dan);
     bonus = 0;
   }
   public double uzmiplacu()
        {
          double baznaplaca = super.uzmiplacu();
          return baznaplaca+bonus;
        }

   public void postavibonus(double b)
        { bonus = b;}
          private double bonus;
 }
```

Start | Semestar 2 | Microsoft PowerPoint - [... | ManagerTest - Notep... | HR | 98% | « 22:56

Nasljeđivanje

```
class Manager extends Djelatnik
 { ....
    public void postavibonus(double b)
   { bonus = b;
    }
   private double bonus;
  }
```

Primjena:

```
Manager sef = ….;
sef.postavibonus(5000);
```

Nasljeđivanje

Na objekt Djelatnik ne može se primijeniti postavibonus metoda jer nije definirana u klasi Djelatnik.

Na objekt Manager mogu se primijeniti metode uzmiime, uzmidatum iako nisu definirane u klasi Manager, ali one se automatski nasljeđuju od superklase (nadklase).

Polja: ime, plaća i datum_zaposlenja također se nasljeđuju od superklase.

Objekt Manager ima 4 polja: ime, placa, datum_zaposlenja i bonus.

=> Metoda uzmiplacu treba vratiti zbroj place i bonusa, što znači da postojeća metoda iz klase Djelatnik ne odgovara potrebama Managera.

Nadjačavanje (redefiniranje) metoda
engl. override

Definira se nova metoda koja "nadjačava" metodu iz nadklase.

```
class Manager  extends Djelatnik
 { ….
   public double uzmiplacu()
    {…. }
    ….
 }
```

Nadjačavanje (redefiniranje) metoda
engl. override

Implementiranje:

a) Vrati zbroj plaće i bonusa

```
public double uzmiplacu()
    {
     return placa + bonus;  // NE RADI !!??
    }
```

Metoda uzmiplacu iz klase Manager nema izravan pristup privatnim poljima nadklase.

Nadjačavanje (redefiniranje) metoda
engl. override

Implementiranje:

b) Koristi public sučelje tj. public metodu uzmiplacu klase Djelatnik

```
public double uzmiplacu()
    {
    double baznaplaca=uzmiplacu(); // NE RADI !!??
    return baznaplaca + bonus;
    }
```

Poziv metode uzmiplacu poziva tu metodu iz klase Manager što za posljedicu ima beskonačan poziv iste metode i dovodi do rušenja programa.

Nadjačavanje (redefiniranje) metoda
engl. override

Implementiranje:

c) Mora se naglasiti da se zove metoda uzmiplacu klase Djelatnik

```
public double uzmiplacu()
    {
    double baznaplaca=super.uzmiplacu();  RADI !!??
    return baznaplaca + bonus;
    }
```

Nasljeđivanje i kontrola pristupa

PRAVILO

Član klase koji je deklariran kao privatan, ostati će privatan za svoju klasu, što znači da neće biti dostupan izvan te klase, čak ni za podklasu !!!

Nadjačavanje (redefiniranje) metoda
engl. override

Kada metoda podklase ima isto ime i tip kao neka metoda nadklase, tada metoda podklase ima prednost nad metodom nadklase – tj. nadjačava je.

Kada se unutar podklase pozove tako nadjačana metoda, uvijek će se izvršiti njena verzija koja je definirana u podklasi.

Nadjačavanje (redefiniranje) metoda

Ako se želi pristupiti verziji redefinirane metode

u nadklasi koristi se operator super.

Metoda se nadjačava samo ako su imena i tipovi metoda identični.

Operator super

2 načina uporabe:

A. **za pozivanje konstruktora klase**

B. **za pristupanje članu nadklase koji je skriven članom podklase.**

Pozivanje konstruktora

```
public Manager
(String i, double p, int godina, int mjesec, int dan)
 {
   super(n,s,godina,mjesec,dan);
   bonus=0;
 }
```

Ovdje je super umjesto “pozovi konstruktor klase Djelatnik s parametrima i, p, godina, mjesec i dan”

Kako Manager konstruktor ne može pristupiti privatnim poljima klase Djelatnik, mora ih inicijalizirati preko konstruktora.

Pozivanje konstruktora

U hijerarhiji klasa konstruktori se pozivaju po redoslijedu izvođenja, od nadklase ka podklasi.

IS – A relacija

Canine extends Animal

Wolf extends Canine

Wolf extends Animal

Canine IS-A Animal

Wolf IS-A Canine

Wolf IS-A Animal

Pas JE životinja
Vuk JE pas
Vuk JE životinja

HAS – A relacija

Bathroom HAS-A Tub and Tub HAS-A Bubbles.
But nobody inherits from (extends) anybody else.

Polimorfizam

Poziv

d.uzmiplacu()

Zove “korektnu” metodu

Činjenica da objektna varijabla (d) može referirati na više stvarnih tipova naziva se polimorfizam.

Automatski odabir odgovarajuće metode na runtime-u naziva se dinamičko razrješavanje metoda.

Nasljeđivanje - polimorfizam

Promjenjiva nadklase može referencirati objekt podklase.

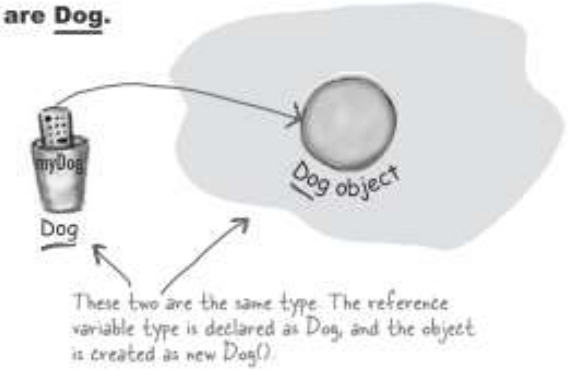
tj.

Referentnoj promjenjivoj nadklase može se dodijeliti referenca na bilo koju podklasu izvedenu iz te nadklase.

Polimorfizam

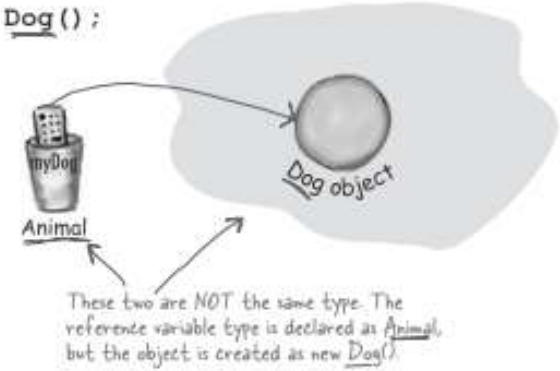
The important point is that the reference type AND the object type are the same.

In this example, both are Dog.

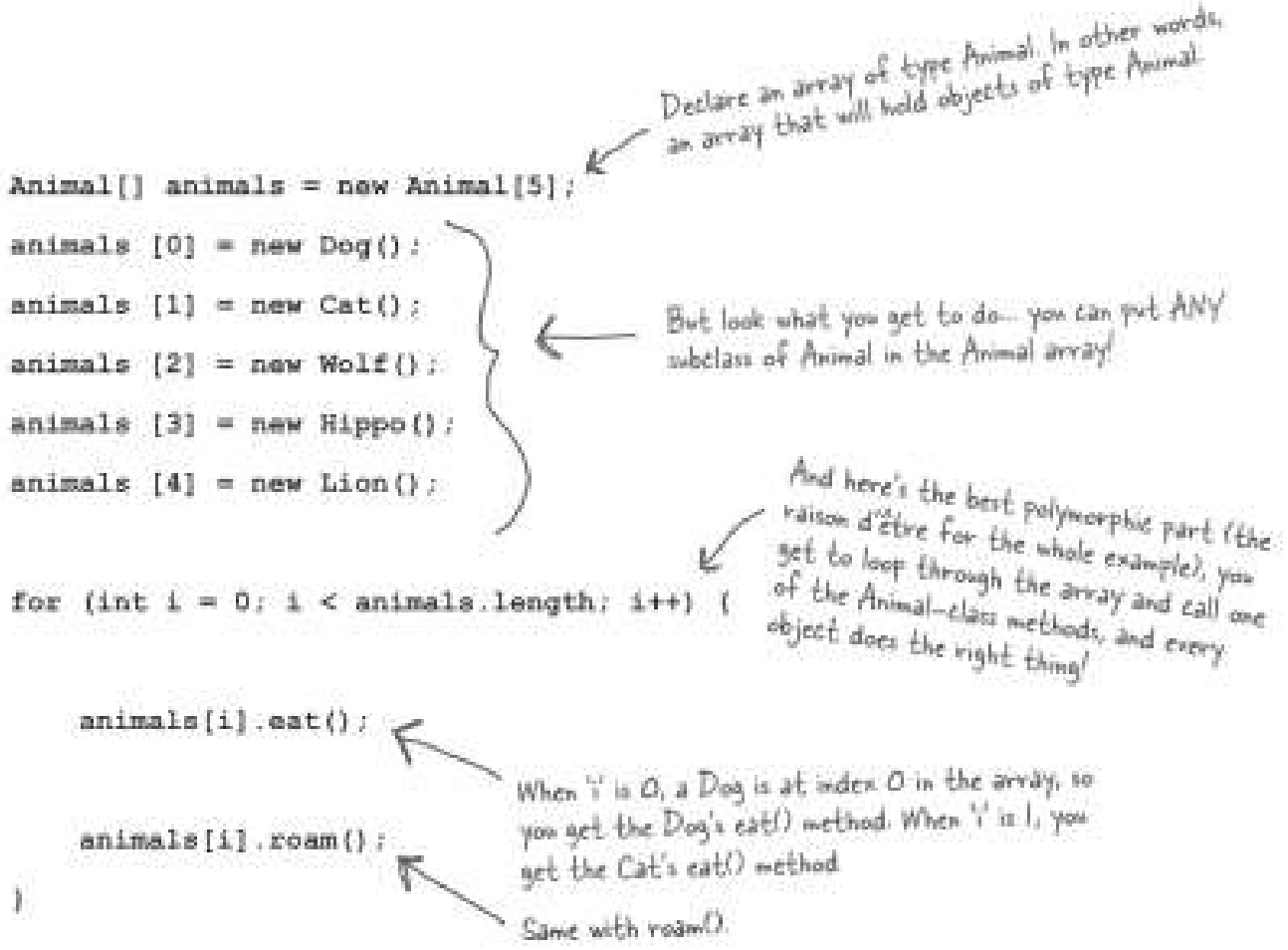


But with polymorphism, the reference and the object can be *different*.

```
Animal myDog = new Dog();
```



Polimorfizam



Nasljeđivanje - polimorfizam

U JAVI – objekt varijable su polimorfne.

Npr. varijabla tipa Djelatnik može referencirati na objekt tipa Djelatnik ili na objekt bilo koje podklase klase Djelatnik (Manager, Tajnica …)

```
Djelatnik[] osoblje = new Djelatnik[3];
Manager sef = new Manager(….);
osoblje[0] = sef;
```

Varijable osoblje[0] i sef odnose se na isti objekt.

osoblje[0] je za kompilator samo objekt Djelatnik

Može se pozvati sef.postavibonus(5000); ali ne i

```
osoblje[0]. postavibonus(5000); // GREŠKA
```

Nasljeđivanje - polimorfizam

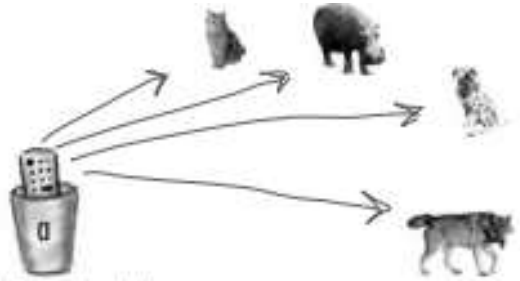
Deklarirani tip za osoblje[0] je Djelatnik, a metoda postavibonus nije metoda klase Djelatnik.

Također, ne može se dodijeliti referenca nadklase varijabli podklase:

Manager m = osoblje[i]; // GREŠKA

Nisu svi djelatnici manageri.

Polimorfizam argumenata



```
class Vet {
   public void giveShot(Animal a) {
      // do horrible things to the Animal at
      // the other end of the 'a' parameter
      a.makeNoise();
   }
}
```

The Animal parameter can take ANY Animal type as the argument. And when the Vet is done giving the shot, it tells the Animal to makeNoise(), and whatever Animal is really out there on the heap, that's whose makeNoise() method will run.

```
class PetOwner {
   public void start() {
      Vet v = new Vet();
      Dog d = new Dog();
      Hippo h = new Hippo();
      v.giveShot(d);
      v.giveShot(h);
   }
}
```

The Vet's giveShot() method can take any Animal you give it. As long as the object you pass in as the argument is a subclass of Animal, it will work.

Dog's makeNoise() runs

Hippo's makeNoise() runs

Uporaba apstraktnih klasa i metoda

Klasa:

modifikator abstract class Ime_klase {…}

Metoda:

modifikator abstract tip ime(lista parametara);

Uporaba apstraktnih klasa

Ne traži implementiranje metoda.

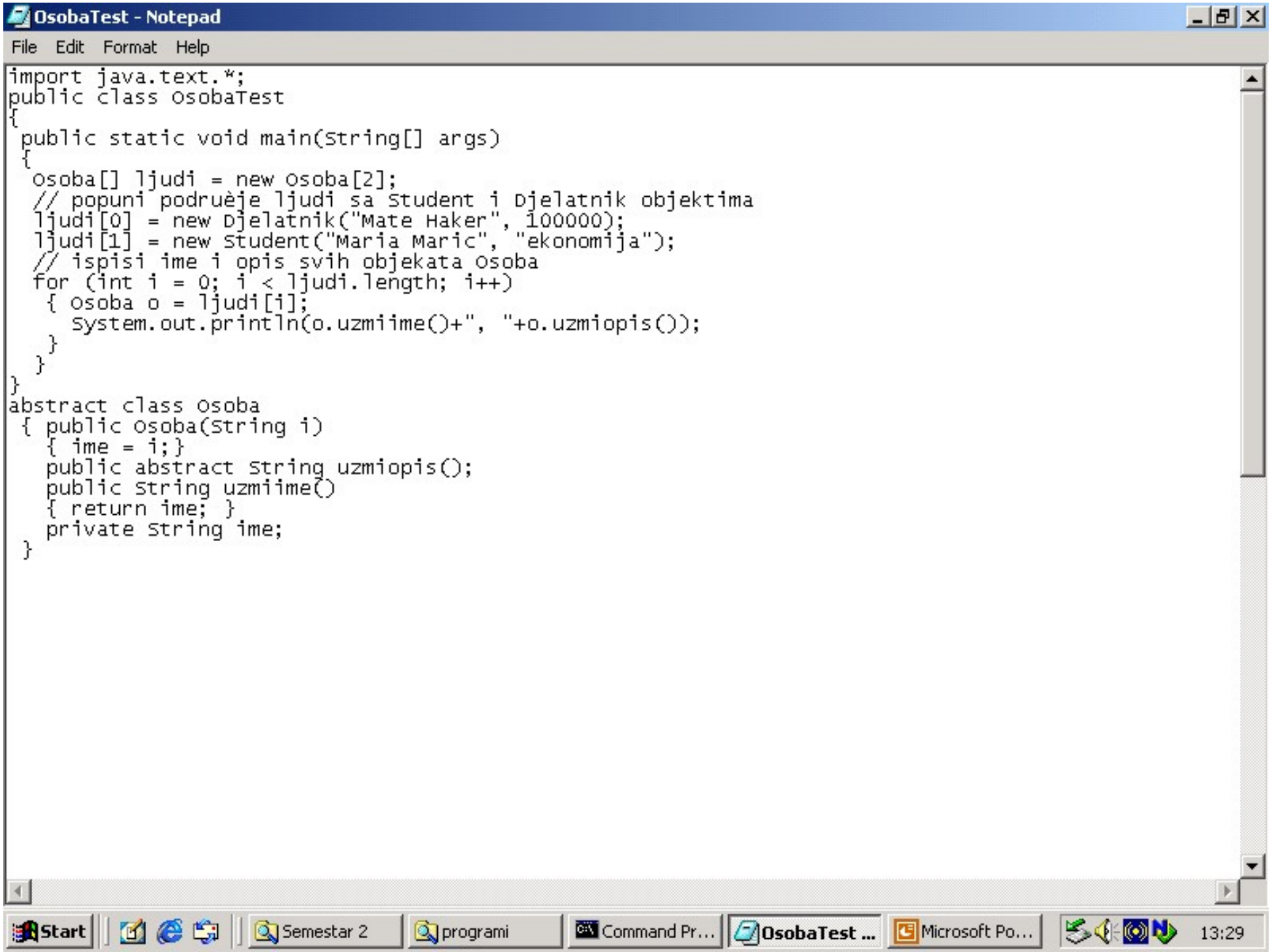
Klasa može biti definirana kao apstraktna iako nema apstraktnih metoda.

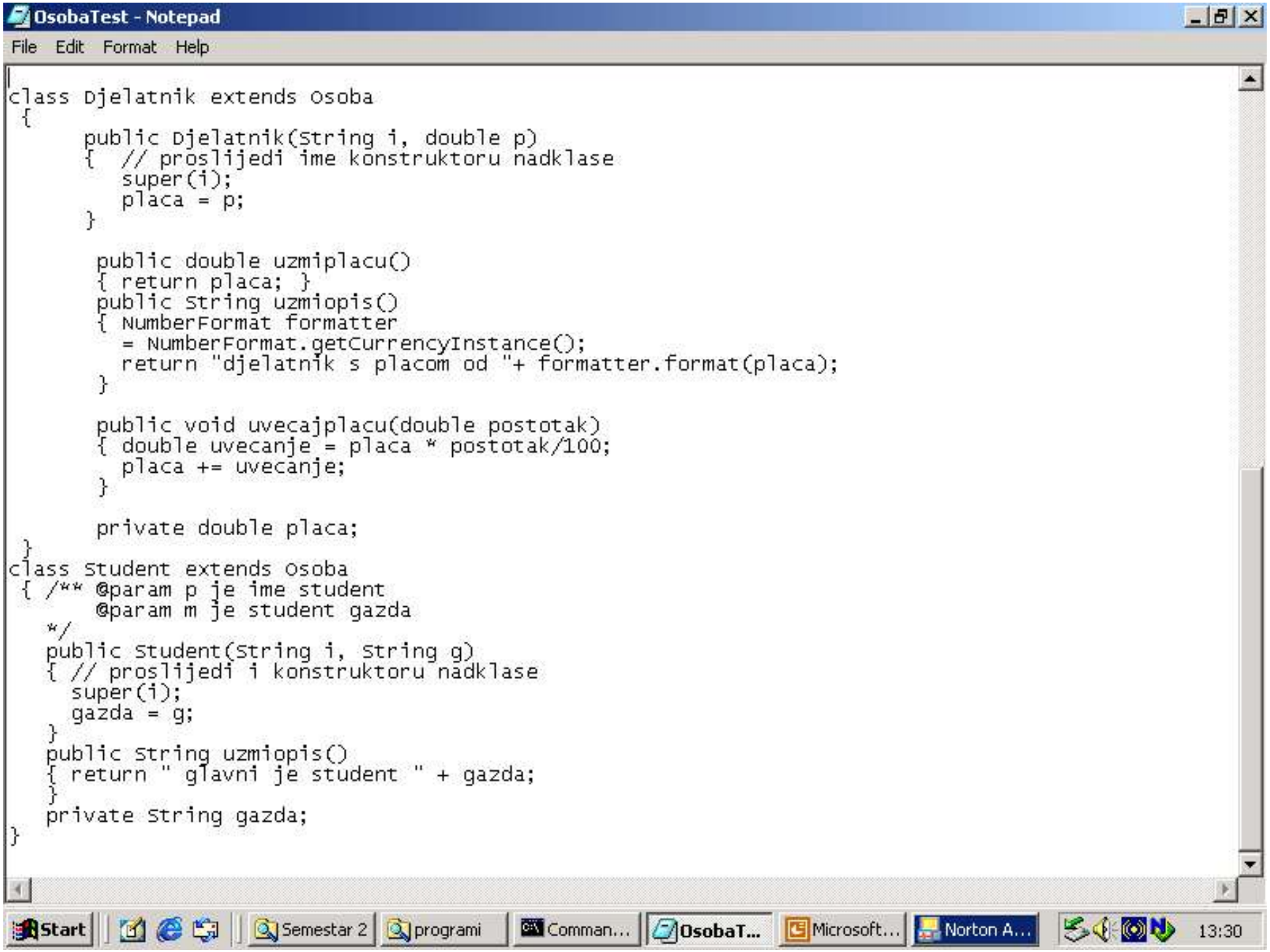
Apstraktne klase nemaju instancu, tj. ako je klasa prijavljena kao apstraktna niti jedan objekt te klase ne može biti kreiran.

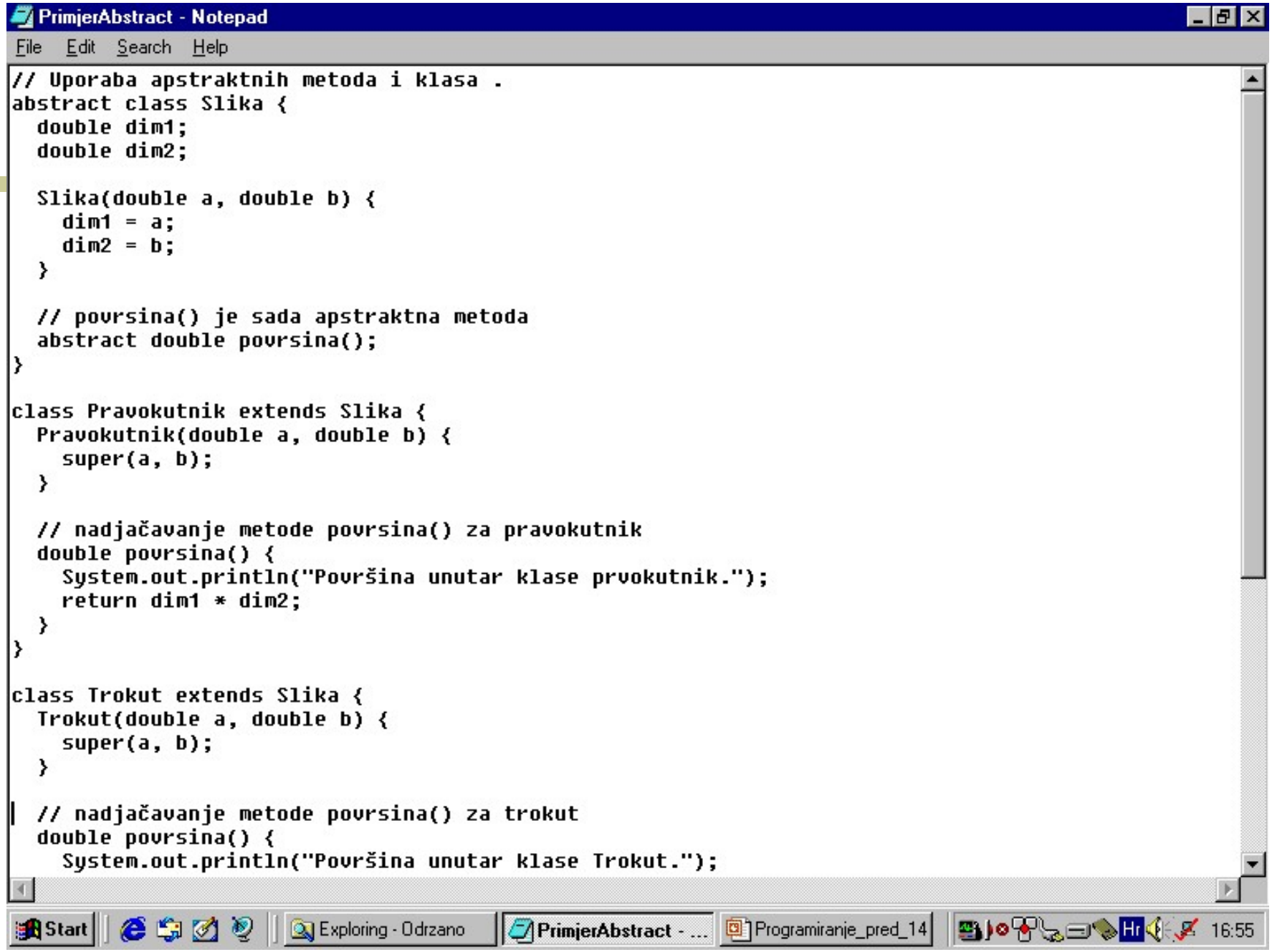
Npr. izraz new Osoba(“xx yy”) je greška.

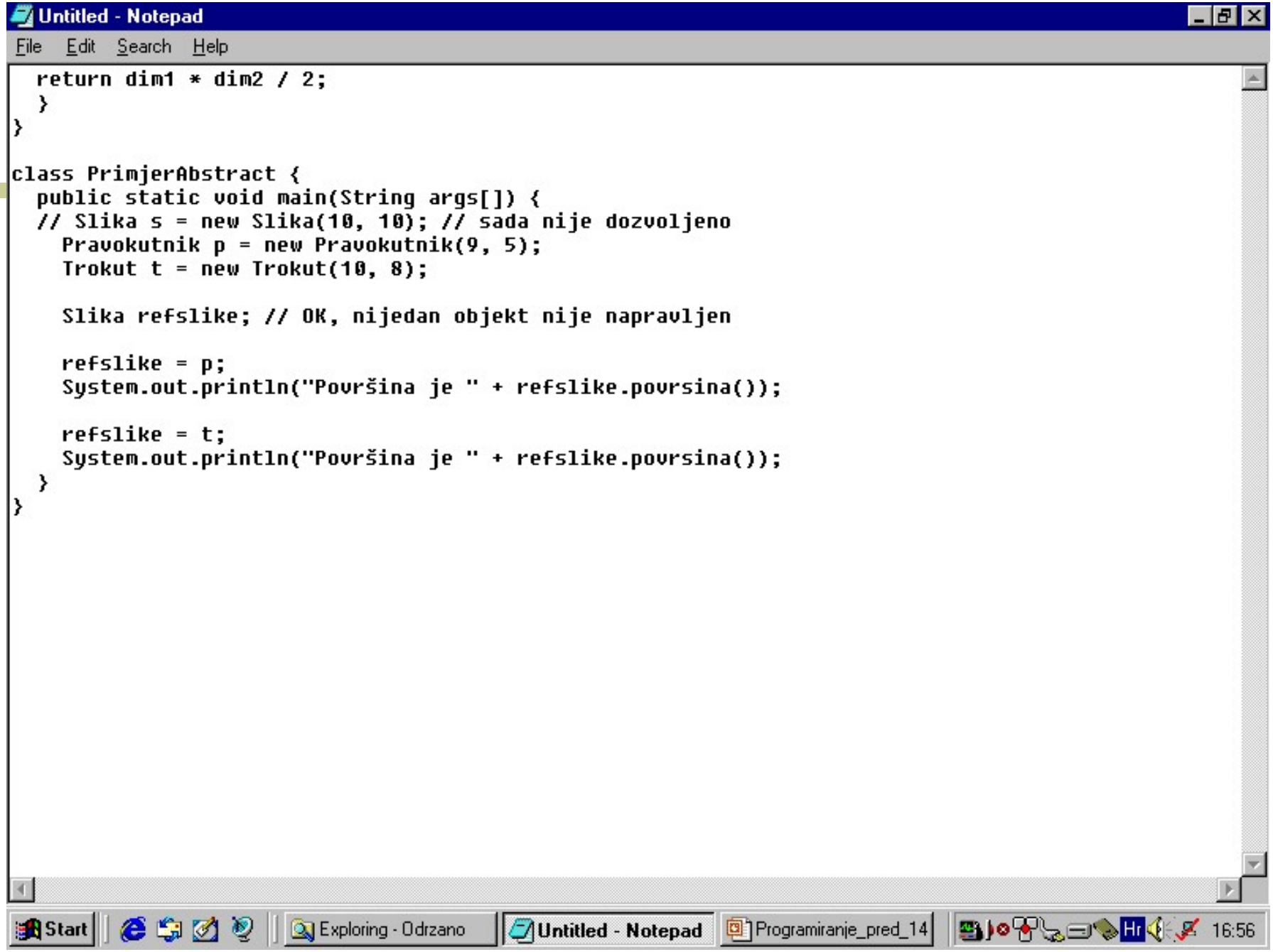
Može se kreirati objektna varijabla apstraktne klase, ali ona mora referirati na neapstraktnu podklasu.

Npr. Osoba o = new Student(“xx yy”, “ekonomija”);


OsobaTest - Notepad
File Edit Format Help
import java.text.*;
public class OsobaTest
{
public static void main(String[] args)
{
Osoba[] ljudi = new Osoba[2];
// popuni podruèje ljudi sa Student i Djelatnik objektima
ljudi[0] = new Djelatnik("Mate Haker", 100000);
ljudi[1] = new Student("Maria Maric", "ekonomija");
// ispisi ime i opis svih objekata Osoba
for (int i = 0; i < ljudi.length; i++)
{ Osoba o = ljudi[i];
System.out.println(o.uzmiime()+", "+o.uzmiopis());
}
}
}
abstract class Osoba
{ public Osoba(String i)
{ ime = i;}
public abstract String uzmiopis();
public String uzmiime()
{ return ime; }
private String ime;
}
Start
Semestar 2
programi
Command Pr...
OsobaTest ...
Microsoft Po...
13:29


OsobaTest - Notepad
File Edit Format Help
class Djelatnik extends Osoba
{
public Djelatnik(String i, double p)
{ // proslijedi ime konstruktoru nadklase
super(i);
placa = p;
}
public double uzmiplacu()
{ return placa; }
public String uzmiopis()
{ NumberFormat formatter
= NumberFormat.getCurrencyInstance();
return "djelatnik s placom od "+ formatter.format(placa);
}
public void uvecajplacu(double postotak)
{ double uvecanje = placa * postotak/100;
placa += uvecanje;
}
private double placa;
}
class Student extends Osoba
{ /** @param p je ime student
@param m je student gazda
*/
public Student(String i, String g)
{ // proslijedi i konstruktoru nadklase
super(i);
gazda = g;
}
public String uzmiopis()
{ return " glavni je student " + gazda;
}
private String gazda;
}
Start
Semestar 2
programi
Comman...
OsobaT...
Microsoft...
Norton A...
13:30


PrimjerAbstract - Notepad
File Edit Search Help
// Uporaba apstraktnih metoda i klasa .
abstract class Slika {
double dim1;
double dim2;
Slika(double a, double b) {
dim1 = a;
dim2 = b;
}
// povrsina() je sada apstraktna metoda
abstract double povrsina();
}
class Pravokutnik extends Slika {
Pravokutnik(double a, double b) {
super(a, b);
}
// nadjačavanje metode povrsina() za pravokutnik
double povrsina() {
System.out.println("Površina unutar klase prvokutnik.");
return dim1 * dim2;
}
}
class Trokut extends Slika {
Trokut(double a, double b) {
super(a, b);
}
// nadjačavanje metode povrsina() za trokut
double povrsina() {
System.out.println("Površina unutar klase Trokut.");
Start
Exploring - Odrzano
PrimjerAbstract - ...
Programiranje_pred_14
16:55


Untitled - Notepad
File Edit Search Help
return dim1 * dim2 / 2;
}
}
class PrimjerAbstract {
public static void main(String args[]) {
// Slika s = new Slika(10, 10); // sada nije dozvoljeno
Pravokutnik p = new Pravokutnik(9, 5);
Trokut t = new Trokut(10, 8);
Slika refslike; // OK, nijedan objekt nije napravljen
refslike = p;
System.out.println("Površina je " + refslike.povrsina());
refslike = t;
System.out.println("Površina je " + refslike.povrsina());
}
}
Start
Exploring - Odrzano
Untitled - Notepad
Programiranje_pred_14
16:56

Vježba

- Formirati polje od 3 elementa
- Formirati klasu Student (ime, prezime, tip, indeks)
- Formirati podklasu Redoviti uz plaćanje (ime, prezime, tip, indeks, školarina)
- Tip: R - redoviti bez plaćanja

  P – redoviti uz plaćanje

Student8.java

Za Predavanje 08.05.2018.

Za Test 3 (teorijski): Poglavlje 6

Test na računalu: imperativna paradigma

PITANJA