Visoka tehnička škola Niš

Studijski program:

Savremene računarske tehnologije

Internet programiranje

(7)

Nasleđivanje klasa u Javi

Prof. dr Zoran Veličković, dipl. inž. el.

Novembar, 2018.

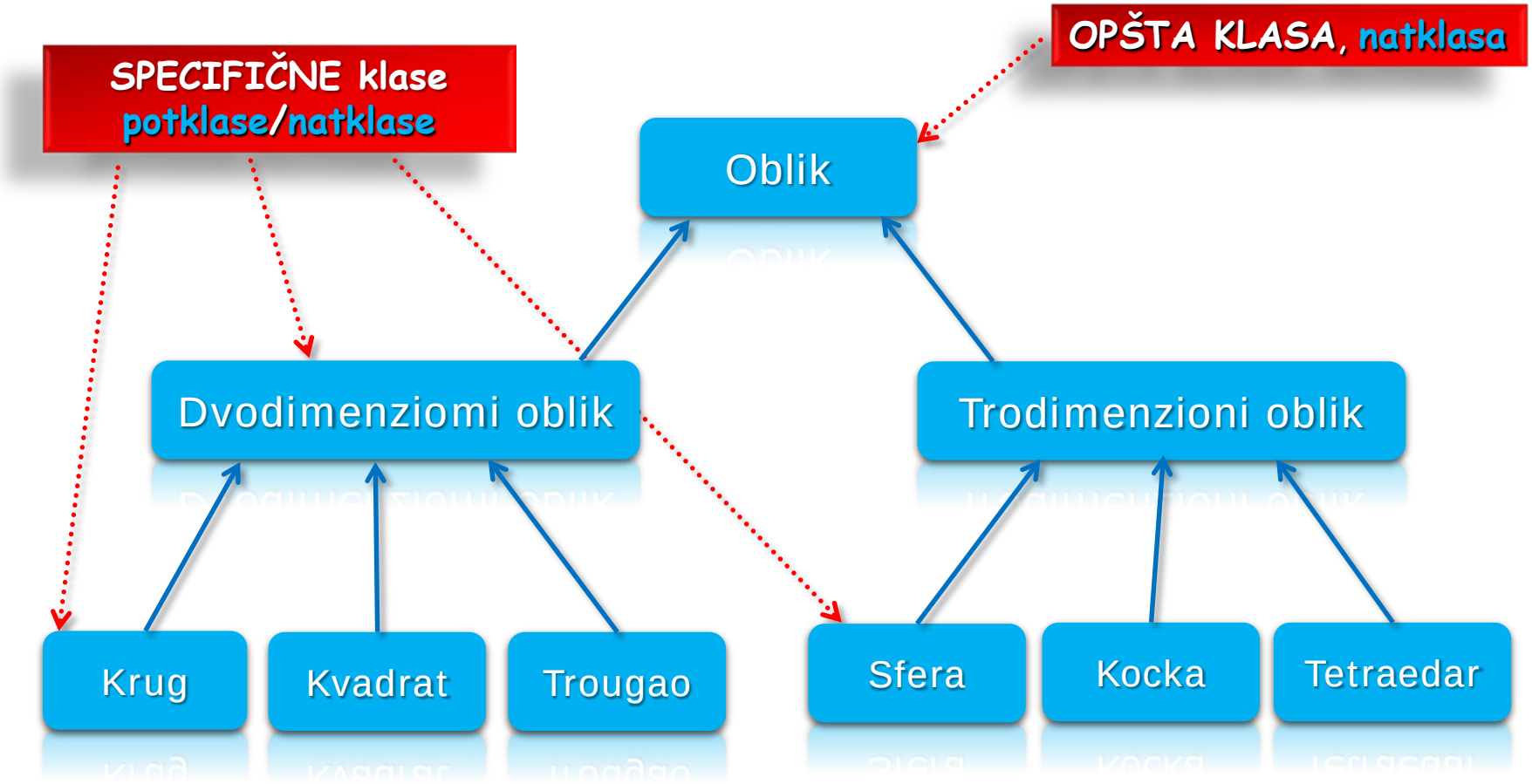
Nasleđivanje klasa

- **NASLEĐIVANJE KLASA** je kamen temeljac **OO** programiranja.
- **NASLEĐIVANJE KLASA** podrazumeva formiranje **OPŠTE KLASE** kojom se definišu **ZAJEDNIČKE KARAKTERISTIKE** skupa **srodnih pojava**.
- Ovako formiranu **OPŠTU KLASU** mogu da **NASLEDE SPECIFIČNE KLASE** koje kroz nasleđivanje **pridodaju SVOJE SPECIFIČNOSTI** opštoj klasi.
- **Nasleđena klasa** se naziva **NATKLASA** (engl. superclass).
- **Klasa koja nasleđuje** naziva se **POTKLASA** (engl. subclass).
- Takođe, može da se kaže da je **potklasa specifičan slučaj natklase**.
- Nasleđivanje se u Javi implementira rezervisanom reči **extends**:

```
class ime_potklase extends ime_natklase {
    //telo klase
}
```

Rezervisana reč **extends** za nasleđivanje klasa

Nasleđivanje klase Oblik



Redefinisanje i preklapanje metoda (1)

- Ako **metoda POTKLASE** ima **ISTO IME** kao i **metoda NATKLASE**, onda se ona **REDEFINIŠE** (engl. **override**).
- **REDEFINISANJE METODE** je moguće **SAMO** ako su **IMENA** i **TIPOVI** metoda **IDENTIČNI** (već obrađeno u .NET-u). .
- Kada se **UNUTAR POTKLASE** pozove **REDEFINISANA METODA** izvršiće se verzija definisana u **POTKLASI**!
- Dakle, verzija metode u **NATKLASI** biće **SAKRIVENA**.
- Međutim, ako imena i tipovi promeljivih u metodi **NISU IDENTIČNI**, onda su metode **PREKLOPLJENE** (engl. **overlapped**).
- **REDEFINISANJE METODA** omogućava da **OPŠTA KLASA** definiše metode koje će biti **ZAJEDNIČKE ZA SVE KLASE**, ostavljajući potklasama slobodu da definišu **SPECIFIČNE VARIJANTE NEKIH** ili **SVIH** metoda.
- Od čega zavisi **koja će** od prekloplјenih/redefinsanih metoda biti **pozvana na izvršenje**?

Dinamičko razrešavanje metoda

- Ovaj problem se u **OO** programiranju naziva **DINAMIČKO RAZREŠAVANJE METODA**.
- **DINAMIČKO RAZREŠAVANJE METODA** je **Javin OO** mehanizam pomoću koga se poziv upućen redefinisanoj metodi razrešava **U TRENUTKU IZVRŠENJA** (a ne kao što je to uobičajeno u proceduralnim jezicima - prilikom prevođenja - kompajliranja).
- Ova **RAZLIKA** između **procedurnih** i **OO jezika** ima za posledicu da se u **trenutku prevođenja NE ZNA** koja će se metoda zapravo pozvati - o tome se odlučuje tek **U TRENUTKU IZVŠAVANJA** (ko to odlučuje?).
- Ova važna karakteristika **OO jezika OMOGUĆAVA** da referencna **PROMENLJIVA NATKLASE** može da **ukaže na OBJEKAT POTKLASE**!
- Posledica ove karakteristike je da **TIP REFERENCNOG OBJEKTA** određuje **KOJA** će **metoda biti pozvana**, **A NE TIP REFERENCNE PROMENLJIVE**.
- **U procesu nasleđivanja**, pored **REDEFINISANJA** i **PREKLAPANJA** metoda mogu se **DODATI** i **NOVE metode i promenljive**.

Nasleđivanje i dodavanje metoda (1)

```
// Kreiranje klase A - biće kasnije korišćena kao natklasa
class A {
  int i, j;
  void prikaži_ij() {
    System.out.println("i i j: " + i + " " + j);
   }
}
// Kreiranje potklase B nasleđivanjem natklase A
class B extends A {
  int k;
  void prikaži_k() {
   System.out.println("k: " + k);
}
  void zbir() {
    System.out.println("i+j+k: " + (i+j+k));
   } }
```

i, j su promenljive klase A

rezervisana reč: extends:
klasa B nasleđuje klasu A

k je promenljiva potklase B

U potklasi B nalaze se SVI ČLANOVI natklase A

2 dodatne metode klase B:
prikaži_k() i zbir()

Nasleđivanje i dodavanje metoda (2)

```
class PrimerNasleđivanja {
  public static void main(String args[]) {
    A superOBJ = new A();     B subOBJ = new B();
// natklasa može da se koristi samostalno.
    superOBJ.i = 10;     superOBJ.j = 20;
    System.out.println("Sadržaj natklase superOb: ");
    superOBJ.prikaži_ij();
    System.out.println();
/* potklasa ima pristup svim javnim članovima natklase*/
    subOBJ.i = 7;     subOBJ.j = 8;     subOBJ.k = 9;
    System.out.println("Sadržaj objekta subOb: ");
    subOBJ.prikaži_ij();  subOBJ.prikaži_k();
    System.out.println();
    System.out.println("Suma i, j i k u subOb je :");
    subOBJ.zbir();
  } }
```

Kreiranje superOBJ - Objekt tipa A

subOBJ - Objekt tipa B

i , j su vidljivi u objektima tipa A i B

Potklasa ima pristup SVIM javnim član. svoje natklase

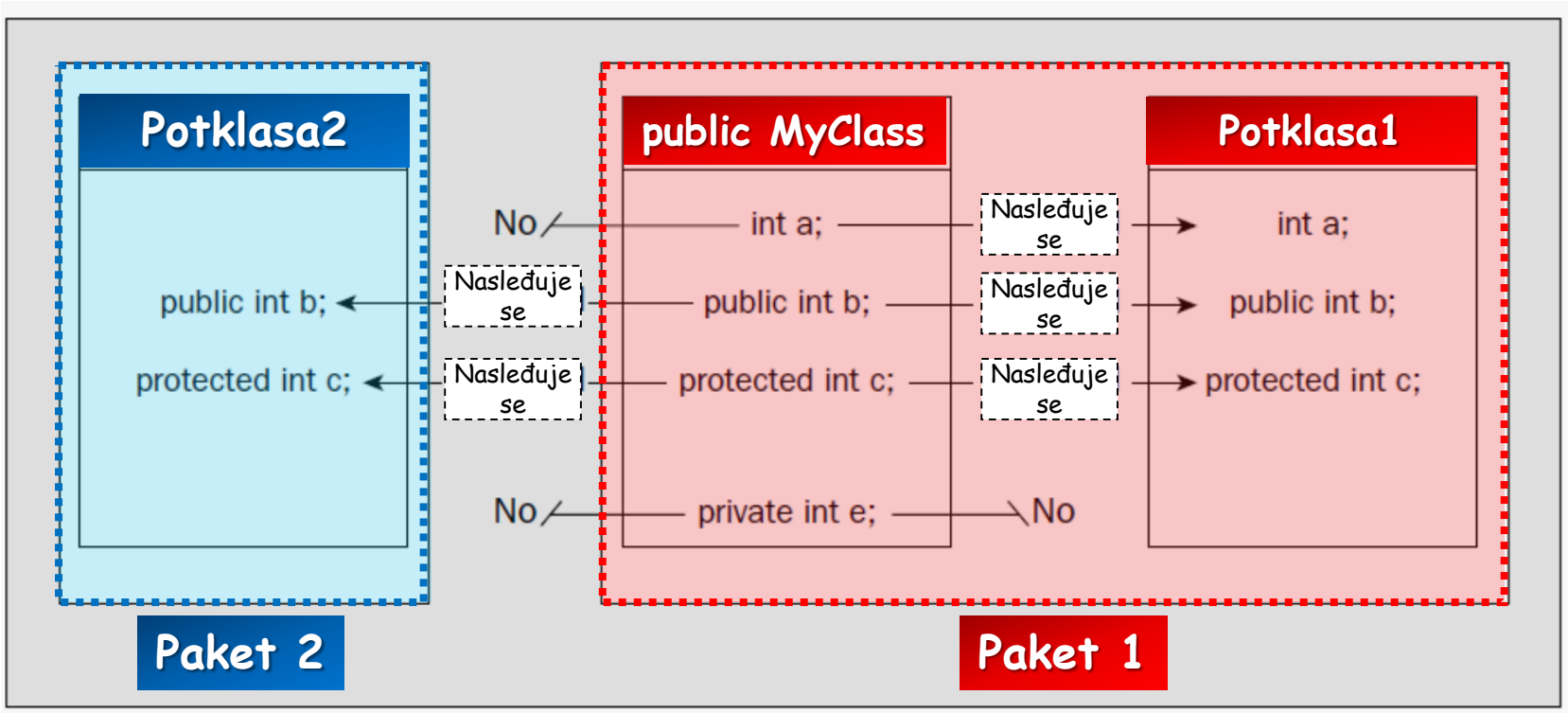
Kontrola pristupa i nasleđivanje (1)

- **POTKLASA** sadrži **SVE ČLANOVE** svoje **NATKLASE**.
- Međutim, **POTKLASA NE MOŽE DA PRISTUPI** onim članovima svoje natklase koji su označeni kao **PRIVATNI**.
- **ČLAN KLASE** koji je deklarisan kao **privatan**, **OSTAĆE PRIVATAN** za svoju klasu i **NEĆE BITI DOSTUPAN** izvan te klase – čak ni iz **POTKLASE**!
- U Javi **potklasa** može imati **SAMO JEDNU** natklasu!
- U procesu **višestrukog nasleđivanja POTKLASA MOŽE POSTATI NATKLASA** drugoj potklasi.
- Klasa **NE MOŽE** biti **SAMA SEBI NATKLASA**!
- Uticaj **PAKETA** i **IDENTIFIKATORA PRISTUPA** na dostupnost članovima klasa dat je u sledećoj tabeli.
- U nastavku biće prikazan primer koji se **NE MOŽE KOMPAJLIRATI** zbog greške koja se odnosi na **parvo pristupa** privatnoj promenljivoj u potklasi.

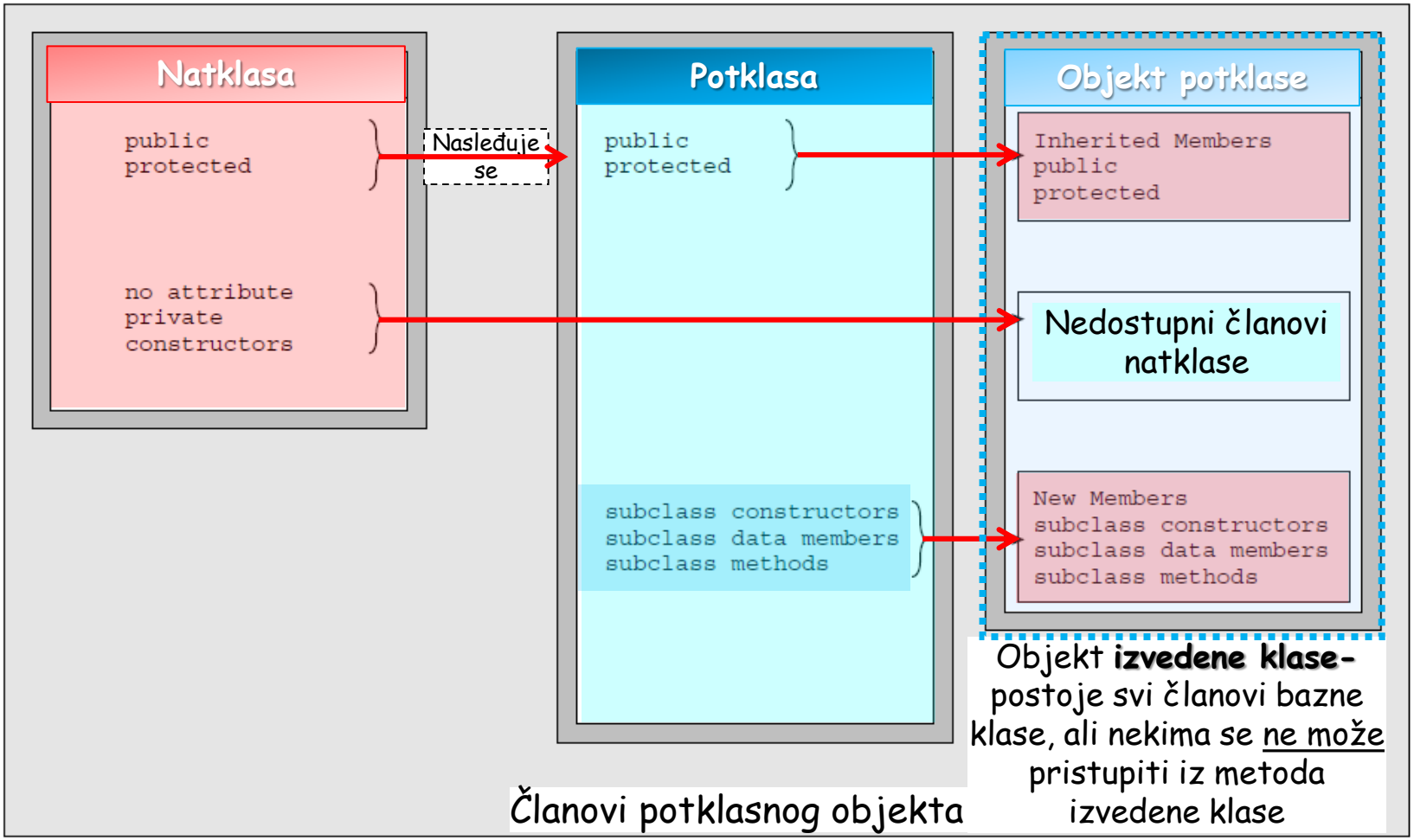
Paketi i pristup članovima klase

SPECIFIKATOR PRISTUPA →	private	bez spec.	protected	public
Ista KLASA	Da	Da	Da	Da
Potklasa istog PAKETA	Ne	Da	Da	Da
KLASA istog pak. koje nisu potklase	Ne	Da	Da	Da
POTKLASE iz drugog PAKETA	Ne	Ne	Da	Da
KLASE iz drugih PAKETA koje nisu POTKLASE	Ne	Ne	Ne	Da

Kontrola pristupa i nasleđivanje (1)



Kontrola pristupa i nasleđivanje (2)



Kontrola pristupa i nasleđivanje (3)

```
class A {
  int i;
  private int j;
  void setij(int x, int y) {
    i = x;
    j = y;
  }
}

class B extends A {
  int total;
  void sum() {
    total = i + j;
  }
}
```

Natklasa A

Podrazumeva se javni pristup

Privatno za klasu A

Potklasa B

Promenljiva j NIJE DOSTUPNA ovde jer je privatna promeljiva clase A !!

// GREŠKA, program se ne može kompajlirati!

Kontrola pristupa i nasleđivanje (4)

```
class Access {
  public static void main(String args[])
   {
    B subOBJ = new B();
    subOb.setij(10, 12);
    subOBJ.sum();
    System.out.println("Total is " + subOb.total);
  }
}
```

Ovaj program NIJE MOGUĆE kompajlirati !!

Natklasa referencira obj. potklase (1)

- Referencnoj promenljivoj **NATKLASE** može se dodeliti referenca na **BILO KOJU POTKLASU** izvedenu iz te natklase.
- **Za određivanje** kojim se članovima **može pristupiti** bitan je **TIP REFERENCNE PROMENLJIVE**, a **NE TIP OBJEKTA** na koji ona ukazuje.
- Kada se referenca na **OBJEKT POTKLASE** dodeli **referencnoj promenljivoj NATKLASE**, imaće pristup **SAMO ONIM DELOVIMA OBJEKTA** koji su definisani **NATKLASOM**.
- Ovo je logično, jer **NATKLASA NE ZNA** šta je sve dodato u **potklasi** kroz proces nasleđivanja!
- Pogledamo **primer** na sledećem slajdu.
- Pretpostavlja se da je klasa **Box NATKLASA** klase **BoxWeight**.

Natklasa referencira obj. potklase (2)

```
class RefDemo {
  public static void main(String args[]) {
    BoxWeight weightbox = new BoxWeight(3, 5, 7, 8.37);
    Box plainbox = new Box();
    double vol;
    vol = weightbox.volume();
    System.out.println("Volume of weightbox is " + vol);
    System.out.println("Weight of weightbox is " + weightbox.weight);
    System.out.println();
    plainbox = weightbox;

    vol = plainbox.volume(); // OK, volume() je definisano u klasi Box
    System.out.println("Volume of plainbox is " + vol);
//  System.out.println("Weight of plainbox is " + plainbox.weight);
  }
}
```

weightbox je objekt klase BoxWeight (poseduje metodu volume) koja je izvedena iz klase Box

Dodeljivanje vrednosti referencne promenljive objekta BoxWeight referencnoj promenljivoj objekta Box (ovo je DOZVOLJENO)

GREŠKA, plain nema član weight!

Rezervisana reč: super (1)

- Na prethodnom primeru je pokazano kako **PROMENLJIVA NATKLASE može da referencira OBJEKT POTKLASE**.
- Međutim, kad **POTKLASA** treba da se obrati **svojoj neposrednoj NATKLASI** to se može uraditi rezervisanom reči **super**.
- Rezervisana reč **super** ima **dva** pojavna oblika:
  1. **Poziva konstruktor natklase ili**
  2. **Pristupa članu natklase** koji je bio **sakriven** članom poklase.
- **POTKLASA** može da pozove **konstruktor** definisan **natklasom** koristeći naredbu **super** na sledeći način:

  super (lista parametara)

- Naredba **super** mora da bude **PRVA NAREDBA** koja se izvršava **unutar KONSTRUKTORA POTKLASE**.

Rezervisana reč: super (2)

```
class Box {
  double width;
  double height;
  double depth;
    ...
    //      različiti konstruktori sa parametrima i/ili bez njih
    ...
}

class BoxWeight extends Box {
  double weight;
  BoxWeight(double w, double h, double d, double m) {
    super(w,h,d)
        weight = m;
}
```

Klasa Box se proširuje da uključi težinu: BoxWeight

1. konstruktor za BoxWeight koji inicijalizuje promenljive

Poziv (inicijalizacija) konstruktora NATKLASE Box

Rezervisana reč: super (2)

```
BoxWeight(BoxWeight ob) {
    super(ob)
        weight = ob.weight;
}

BoxWeight() {
    super()
        weight = -1;
  }
}
```

2. konstruktor za BoxWeight

Prosleđivanje objekta konstruktoru

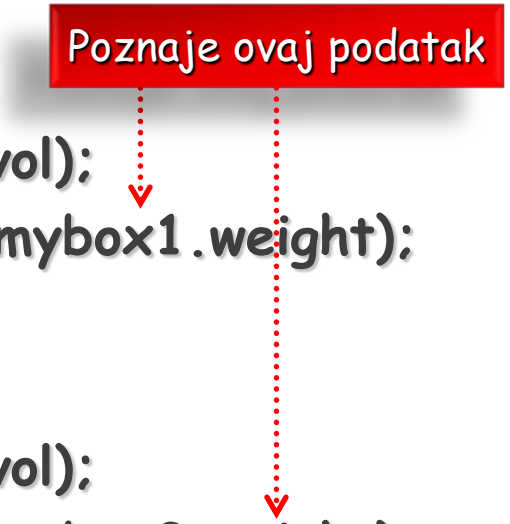
Pozivi konstruktora NATKLASE Box

3. Podrazumevani konstruktor za BoxWeight

Napomena: Nisu prikazani konstruktori koje poseduje klasa Box, a na koje se poziva klasa BoxWeight

Primer: klase Kutija sa težinom

```
class DemoBoxWeight {
  public static void main(String args[]) {
    BoxWeight mybox1 = new BoxWeight(10, 20, 15, 34.3);
    BoxWeight mybox2 = new BoxWeight(2, 3, 4, 0.076);
    double vol;
    vol = mybox1.volume();
    System.out.println("Volume of mybox1 is " + vol);
    System.out.println("Weight of mybox1 is " + mybox1.weight);
    System.out.println();
    vol = mybox2.volume();
    System.out.println("Volume of mybox2 is " + vol);
    System.out.println("Weight of mybox2 is " + mybox2.weight);
  }
}
```



Drugi način primene reči super

- Drugi oblik upotrebe rezervisane reči **super** je sličan upotrebi reči **this**.
- Rezervisana reč **super** uvek ukazuje na **NATKLASU POTKLASE** u kojoj je upotrebljena!
- Ovaj oblik se primenjuje na sledeći način: **super.član**

```
class A { int i; }
  class B extend A {
        int i;
        B(int a, int b) {
         super.i=a;
         i=b;
}
...
```

skriva i iz klase A

konstruktor klase B

i iz klase A

i iz klase B

Apstraktne metode i klase (1)

- Može se definisati **NATKLASA** koja **SAMO DEKLARIŠE** strukturu klase **BEZ DETALJNOG DEFINISANJA SVIH METODA**.
- Ovako definisane metode nazivaju se **APSTRAKTNE METODE**.
- Da bi se **sprečile greške** koje bi prijavio kompajler, ove metode se moraju označiti kao **APSTRAKTNE** ključnom rečju **abstract**.
- Potklase apstraktne klase **MORAJU REDEFINISATI SVE METODE**.
- Dekleracija **apstraktne metode** je sledeća:

abstract tip ime_metode (lista parametara);

- Apstraktne metode **NEMAJU TELO METODE**!
- Metode koje sadrže apstraktne klase **I SAME SU APSTRAKTNE**!

Apstraktne metode i klase (2)

```
abstract class A {
  abstract void callme();
// Konkretne metode su dozvoljene u apstraktnim klasama
  void callmetoo() {
    System.out.println("This is a concrete method.");
  }
}
class B extends A {
  void callme() {
    System.out.println("B's implementation of callme.");
  }
}
class AbstractDemo {
  public static void main(String args[]) {
    B b = new B();    b.callme();    b.callmetoo();
  }
 }
```

apstraktna klasa A (sadrži aps.metodu)

apstraktna metoda callme()

Pri nasleđivanju MORA SE REDEFINISATI apstraktna metoda!

Redefinisanje apstraktne metode callme()

Višestepena hijerarhija

- Može se napraviti onoliko **nivoa** (slojeva) **nasleđivanja** koliko nam je potrebno!
- Dakle, **POTKLASA** može biti **NATKLASA** neke druge klase!
- Ako se imaju **tri klase** A, B i C, od kojih C može da bude **potklasa** klase B, a ova potklasa klase A!
- Svaka potklasa nasleđuje **SVE OSOBINE** svojih natklasa!
- Ponekad je potrebno **SPREČITI NASLEĐIVANJE** klasa.
- To se obavlja **modifikatorom final**.

```
final class A  {

        // telo klase ...

}
```

Sprečavanje nasleđivanja klase *A*

Modifikator: final

```
class A {
  final void meth() {
    System.out.println("Ovo je final metoda!.");
  }
}

class B extends A {
  void meth() {                    // Greška, ne može se redefinisati.
    System.out.println("Nelegalno!");
  }
}
```

Final metoda meth() se NE MOŽE redefinisati, kompajler će prijaviti grešku

Korenska klasa Object

- **SVE** klase su **potklase** klase **Object**!
- Klasa **Object** definiše sledeće metode:

  Object clone() -- pravi novi object isti kao onaj koji se klonira

  boolen equals(Object objekat) -- utvrđuje jednakost objekata

  void finalize() -- poziv pre čišćenja nekorišćenog objekta.

  class getClass() -- nalazi klasu objekta u trenutku izvršavanja.

  int hashCode() -- vraća ključ za heš. pridružen objektu koji se poziva

  void notify() -- nastavak izvrš. programske niti koja čeka pozivaoca

  void notifyAll() -- nastavak izvrš. svih program. niti koje čekaju poziv

  String toString() -- vraća znakovni niz koji opisuje objekt

  void wait(long milisekunde) -- čeka izvršenje druge niti

  void wait(long milisekunde, int nanosekunde) -- isto