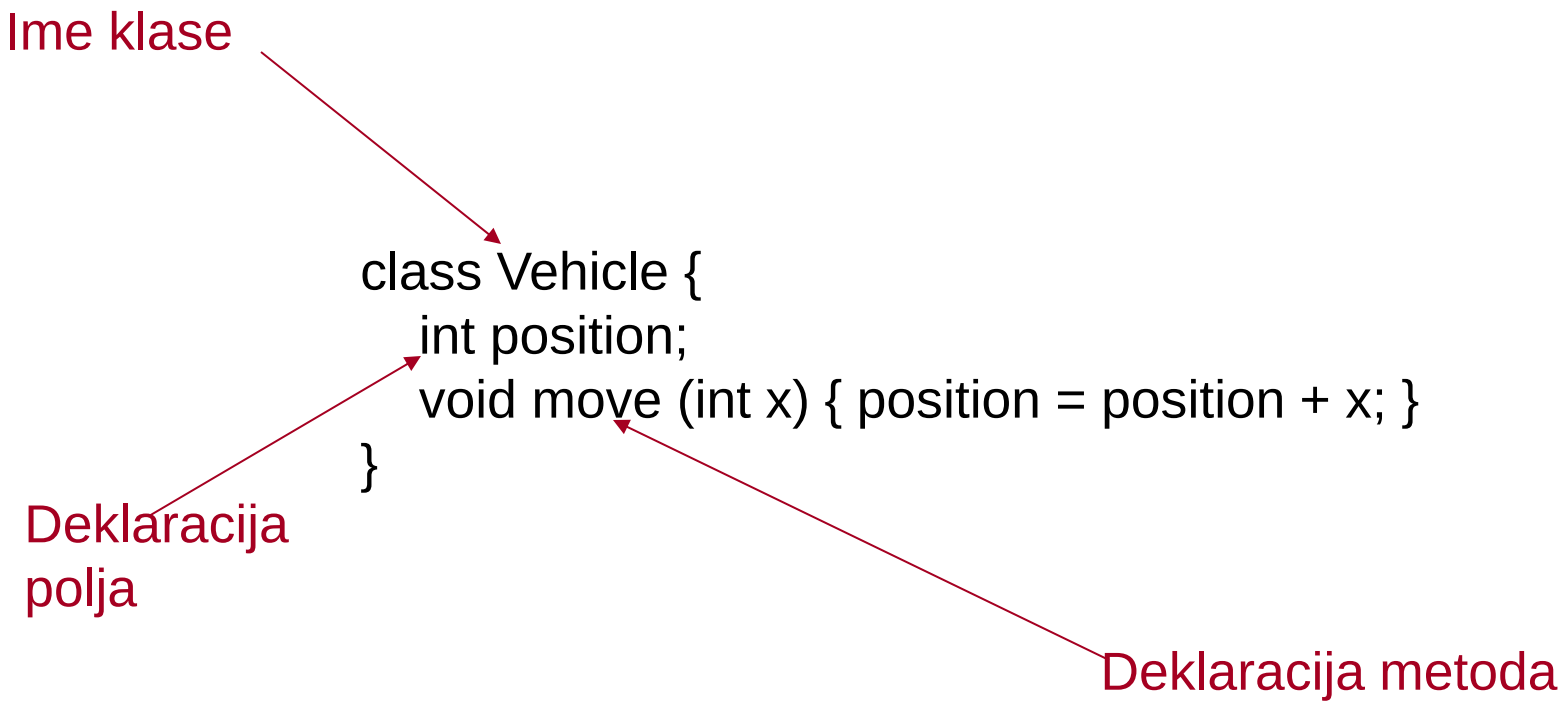
Prevođenje OO jezika

- Literatura: Appel, Modern Compiler Implementation in Java, Second Ed, Chapter 14

Objektno orijentisano programiranje

- Objekti
- Apstrakcija
- Enkapsulacija
- Polimorfizam
- Nasleđivanje

Primer upotrebe OO konstrukata na Javi



OO primer nastavak

```
class Vehicle {
    int position;
    void move (int x) { position = position + x; }
}
class Car extends Vehicle{
    int passengers;
    void await(Vehicle v) {
        if (v.position < position)
            v.move(position - v.position);
        else
            this.move(10);
    }
}
```

Nadklasa od Koje je izvedena Klasa Car

Nova deklaracija polja

Nova deklaracija metoda

Poziv nasleđenog metoda

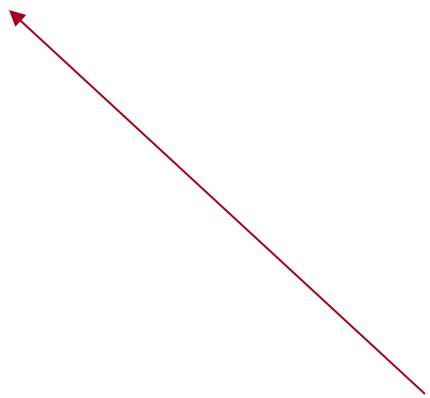
polje “position” objekta v

polje “position” tekućeg objekta

OO primer nastavak

```
class Vehicle {
    int position;
    void move (int x) { position = position + x; }
}

class Truck extends Vehicle{
    void move(int x) {
        if (x <= 55) { position = position + x; }
    }
}
```



Redefinisanje metoda (override)

Upotreba klasa

```
class Main{
    public static void main(String args[]) {
        Truck t = new Truck();
        Car c = new Car();
        Vehicle v = c;
        c.passengers = 2;
        c.move(60);
        v.move(70);
        c.await(t);
    }
}
```

Kreiranje novog objekta

Auto može da se koristi kao opštiji tip vozilo

Poziv nasleđenog metoda

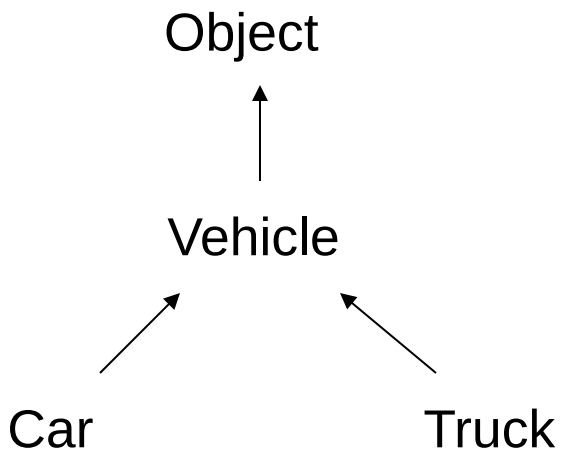
Kamion može da se koristi Kao opštiji tip vozilo

Implementacija objekata

- Ključni problemi koje treba razrešiti su:
  - Kako pristupiti poljima objekta?
    - Uključujući nasleđena i polja tekućeg objekta.
  - Kako pristupiti programskom kodu metoda?
    - Ako tekuća klasa nema definiciju metoda, kako stići do nasleđenog metoda?
    - Kako se obrađuje redefinicija metoda (method override)?

Hijerarhija klasa

- Grafički prikaz relacija nasleđivanja među klasama programa:



  - Ako postoji samo jednostruko nasleđivanje (JN) graf je oblika stabla
  - U jeziku sa višestrukim nasleđivanjem (VN), graf je aciklički
  - VN je nešto složenije za implementaciju od JN

Memorijska reprezentacija objekata kod jednostrukog nasleđivanja (JN)

- Objektima odgovaraju zapisi u memoriji
  - Svaka promenljiva ima polje u zapisu
  - Pristup polju zapisa
    - npr: v.position
    - Problem je što v može ukazivati na Vehicle ili može na Car ili na Truck
    - Isti mašinski kod mora funkcionisati kada znamo *statički tip* objekta v (recimo Vehicle) ali ne znamo njegov *dinamički tip* (Vehicle, Car ili Truck)
    - polje “position” mora biti na istom mestu u zapisima koji reprezentuju objekte klasa Vehicle, Car i Truck.

Rad sa klasama

```
class Vehicle { ...  }
class Car extends Vehicle {
  int passengers;

  void await(Vehicle v) {
    if (v.position  < position) then ... }}

class Truck extends Vehicle {...}

class Main { …
  public static void main(…) {
   Truck t = new Truck();
   Car c = new Car();
   Vehicle v = c;
   c.await(t);
   c.await(c);
   c.await(v);
}}
```

Pretpostavimo da je v pokazivač smešten u reg R

v.position se prevodi kao:

load [R+const]

Koju const treba uzeti?
Mora biti ista bez obzira da li v pokazuje na Car, Truck ili Vehicle

Raspored polja u zapisima Koji reprezentuju kola, kamione i vozila mora biti sličan.

Memorijsko predstavljanje objekata (jednostruko nasleđivanje)

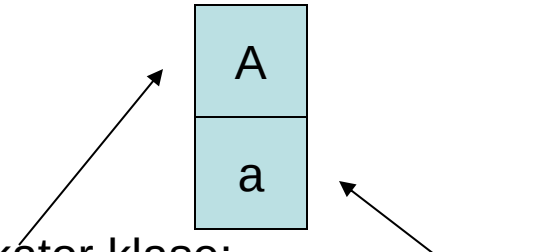
- Rešenje: proširenje na desno
  - Prvo rasporediti nasleđena polja u istom redosledu kao u roditeljskoj klasi (JN => samo jedan roditelj)
  - Zatim rasporediti novo deklarisana polja “na desno” tj. u nastavku memorije

Memorijsko predstavljanje objekata (jednostruko nasleđivanje) - primer

```
class A {                    int a = 0; }
class B extends A        { int b = 0;
                             int c = 0; }
class C extends A        { int d = 0; }
class D extends B        { int e = 0; }
```

Memorijsko predstavljanje objekata (jednostruko nasleđivanje) - primer

```
class A {                  int a = 0; }
class B extends A        { int b = 0;
                           int c = 0; }
class C extends A        { int d = 0; }
class D extends B        { int e = 0; }
```

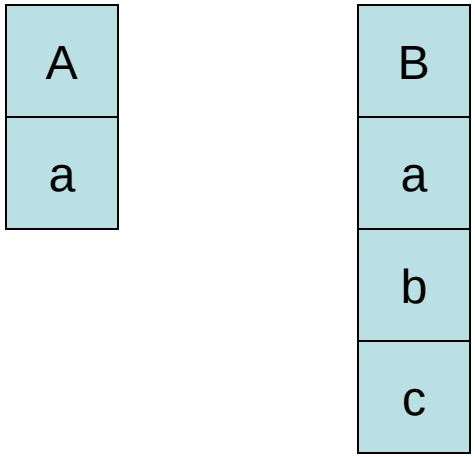


Identifikator klase;
Koristi se za implementaciju ekspl. konverzije u podklasu (tzv. downcasting);
Prvo polje svih klasnih zapisa mora biti identifikator klase

Prvo polje naziva a;
Zapisi svi objekata koji proširuju A moraju imati prvo polje a

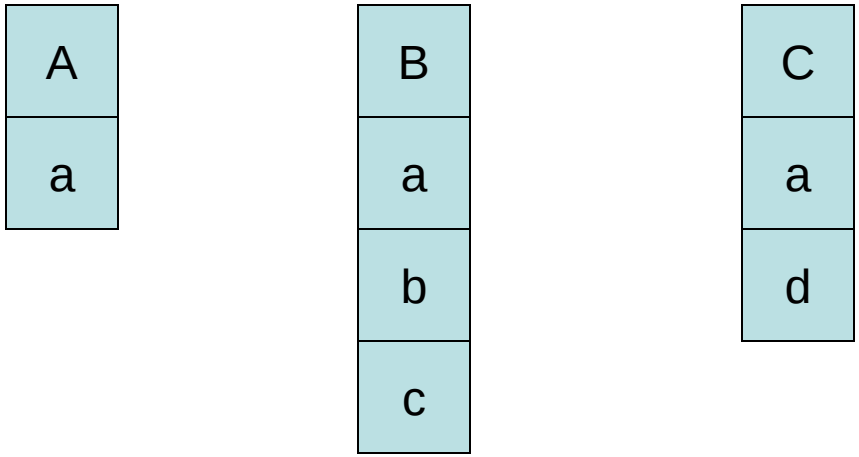
Memorijsko predstavljanje objekata (jednostruko nasleđivanje) - primer

```
class A {                    int a = 0; }
class B extends A          { int b = 0;
                             int c = 0; }
class C extends A          { int d = 0; }
class D extends B          { int e = 0; }
```



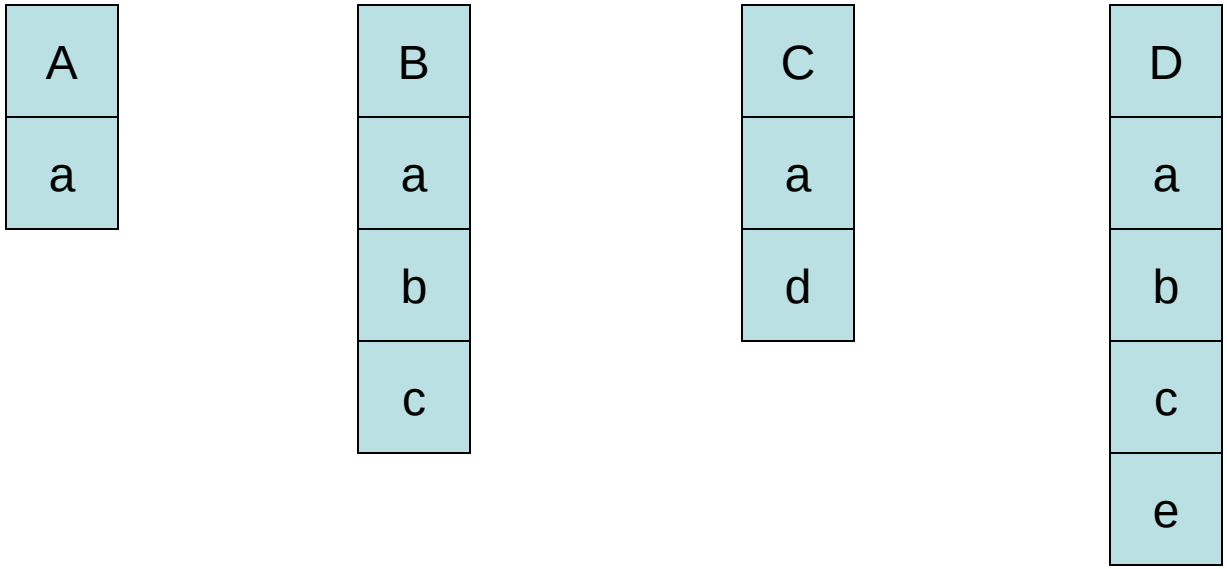
Memorijsko predstavljanje objekata (jednostruko nasleđivanje) - primer

```
class A {                    int a = 0; }
class B extends A          { int b = 0;
                             int c = 0; }
class C extends A          { int d = 0; }
class D extends B          { int e = 0; }
```



Memorijsko predstavljanje objekata (jednostruko nasleđivanje) - primer

```
class A {                    int a = 0; }
class B extends A          { int b = 0;
                             int c = 0; }
class C extends A          { int d = 0; }
class D extends B          { int e = 0; }
```



Statički & dinamički metodi

- Rezultat prevođenja metoda je mašinski kod na određenoj memorijskoj adresi
  - Na mestu pozivanja metoda, moramo odrediti na koju adresu treba izvršiti skok
- Java ima statičke & dinamičke methode
  - Za razrešavanje poziva statičkih metoda, dovoljno je odrediti statički tip objekta čiji se metod poziva
  - Za razrešavanje poziva dinamičkih metoda, treba znati dinamički tip pozvanog objekta

Statički metodi

```
class A {
   static void foo (int x) {…}
   static void bar (int x) {…}
}
class B extends A {
   static void foo (int x) {…}
}

Class Main {…
public static void main(…){
  A a= new A();
  A b = new B();
  B c = new B();
  a.foo(3);    (* calls foo in class A *)
  b.foo(3);    (* calls foo in class A *)
  c.bar(3);    (* calls bar in class A *)
  c.foo(3);    (* calls foo in class B *)
}}
```

- Da bi preveo poziv statičkog metoda,
  - tokom semantičke analize, prevodilac određuje:
    - statički tip (klasu) objekta čiji se metod poziva
    - listu metoda u svakoj klasi
  - zatim utvrđuje
    - Najbliži metod nagore u hijerarhiji klasa sa datim imenom
  - zatim ubacuje
    - Neposredan poziv utvrđenog metoda

Dinamički metodi

```
class A {
        void foo (int x) {…}
        void bar (int x) {…}
 }
 class B extends A {
        void foo (int x) {…}
 }

Class Main {…
public static void main(…){
  A a= new A();
  A b = new B();
  A c =
    if long-and-tricky-computation
    then a
    else b
  c.foo(3)
}
```

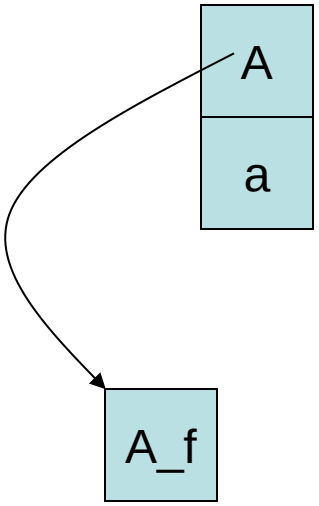
- Metod koji se poziva zavisi od dinamičkog tipa objekta
  - Ne mora biti poznat tokom semantičke analize
- U vreme izvršavanja (run-time), utvrđuje se gde skočiti
  - objekat ima pokazivač na tabelu svojih metoda (v-tabela) u dodatku na svoja polja
- U vreme prevođenja, generiše se kod koji
  - pretražuje v-tabelu objekta
  - nalazi lokaciju metoda u tabeli
  - Vrši skok na početak tela metoda

Memorijsko predstavljanje objekata (jednostruko nasleđivanje) – primer 2

```
class A extends Object { var a := 0; method f () }
class B extends A      { method g () }
class C extends B      { method g () }
class D extends C      { var b := 0 ; method f () }
```

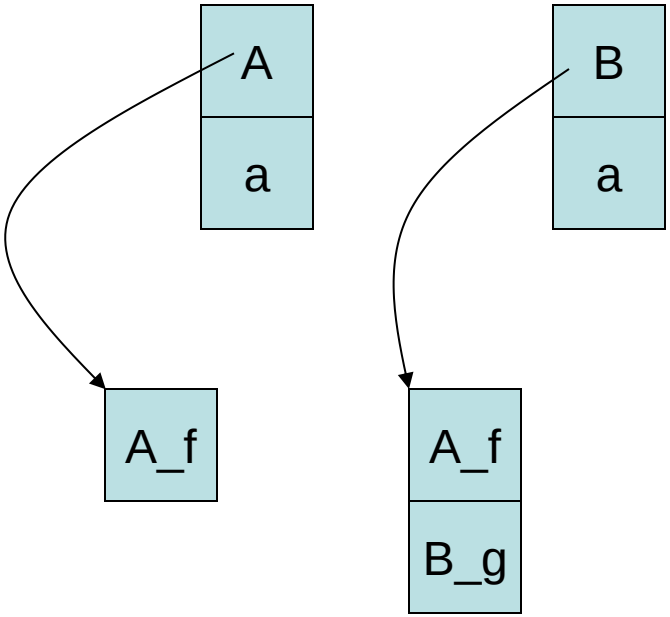
Memorijsko predstavljanje objekata (jednostruko nasleđivanje) – primer 2

```
class A  {                    int a = 0; int f() {…} }
class B extends A           { int g (){…} }
class C extends B           { int g (){…} }
class D extends C           { int b = 0 ; int f (){…} }
```



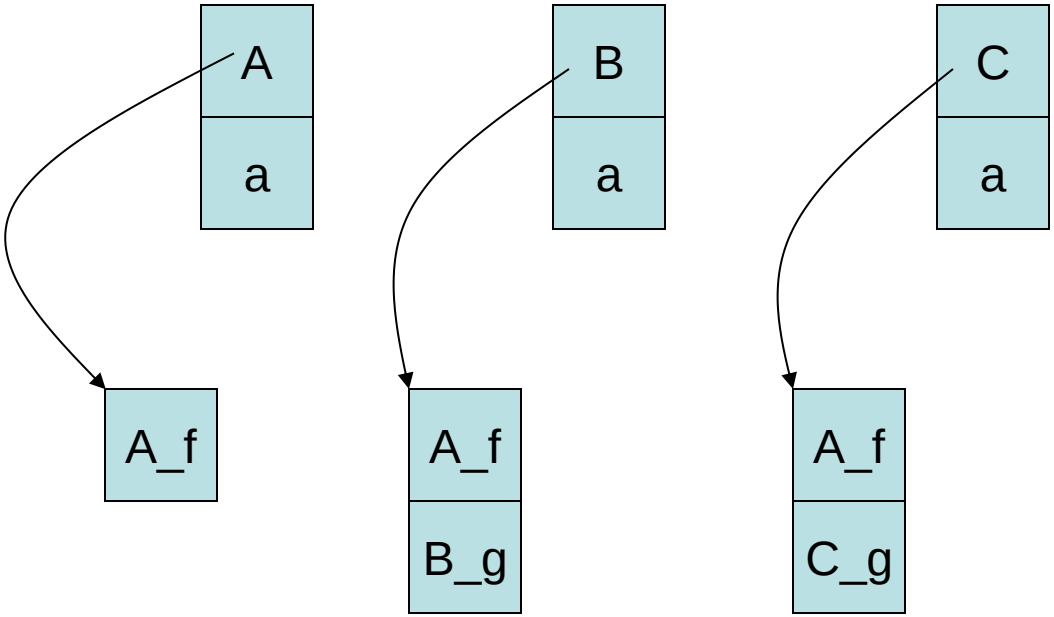
Memorijsko predstavljanje objekata (jednostruko nasleđivanje) – primer 2

```
class A  {                    int a = 0; int f() {…} }
class B extends A            { int g (){…} }
class C extends B            { int g (){…} }
class D extends C            { int b = 0 ; int f (){…} }
```



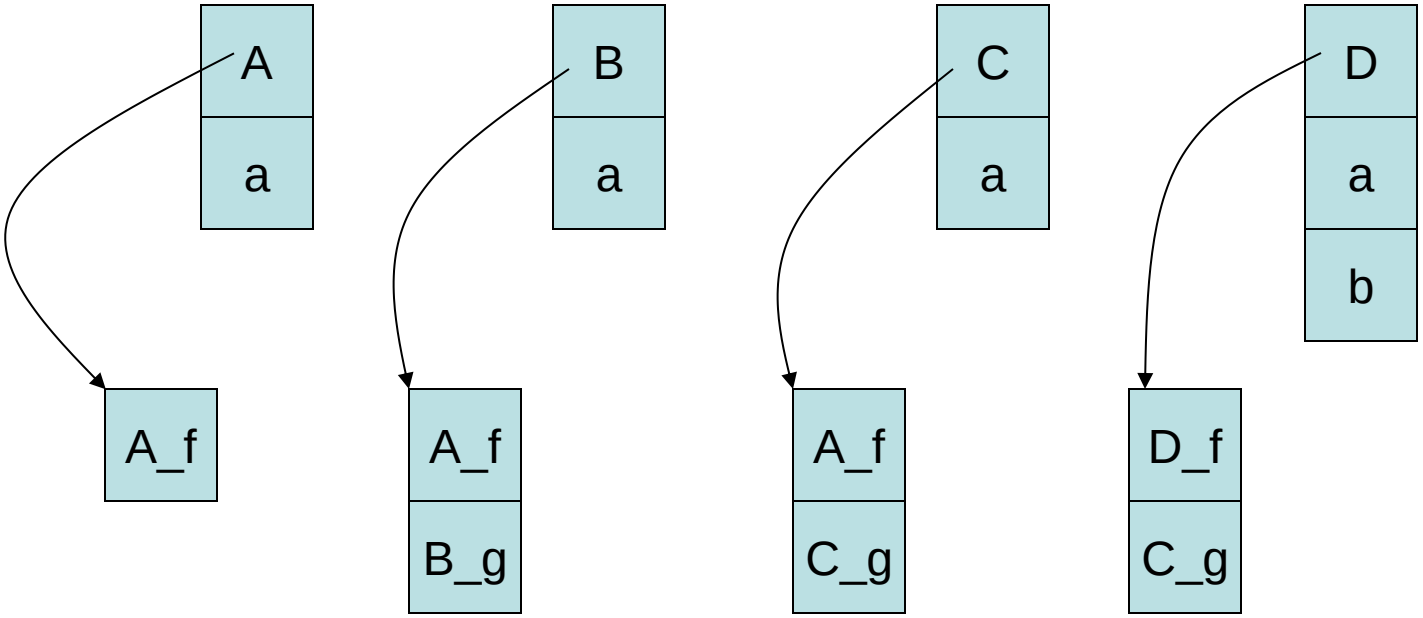
Memorijsko predstavljanje objekata (jednostruko nasleđivanje) – primer 2

```
class A  {                    int a = 0; int f() {…} }
class B extends A            { int g (){…} }
class C extends B            { int g (){…} }
class D extends C            { int b = 0 ; int f (){…} }
```



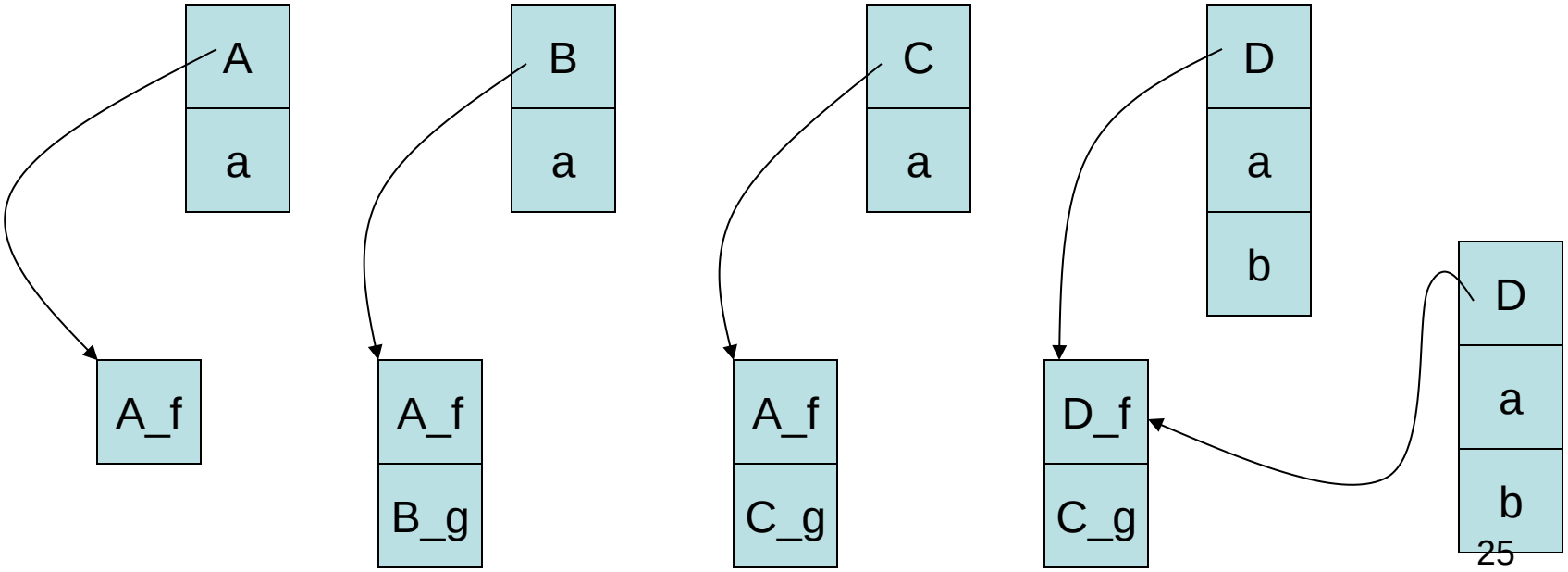
Memorijsko predstavljanje objekata (jednostruko nasleđivanje) – primer 2

```
class A  {                    int a = 0; int f() {…} }
class B extends A           { int g (){…} }
class C extends B           { int g (){…} }
class D extends C           { int b = 0 ; int f (){…} }
```



Memorijsko predstavljanje objekata (jednostruko nasleđivanje) – primer 2

```
class A  {                    int a = 0; int f() {…} }
class B extends A            { int g (){…} }
class C extends B            { int g (){…} }
class D extends C            { int b = 0 ; int f (){…} }
```

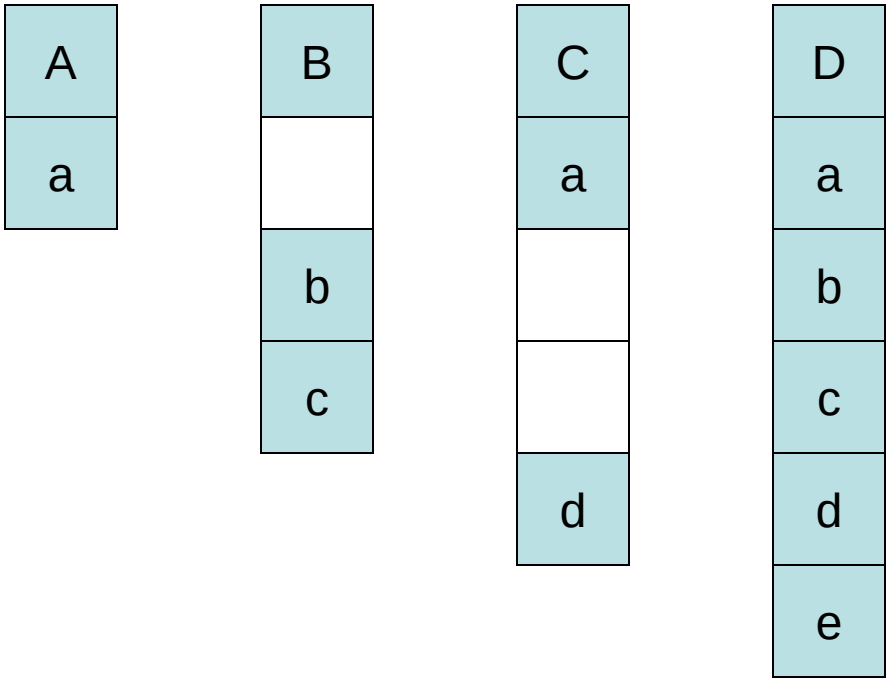


Višestruko nasleđivanje

- Višestruko nasleđivanje je složenije za implementaciju od jednostrukog jer za objekte potklasa “proširenje na desno” ne funkcioniše
  - ako C nasleđuje i od A i od B, ne možemo na isto mesto na početku zapisa objekta staviti i promenljive klase A i promenljive klase B!
  - Za predstavljanje objekta neophodna je globalna analiza

Memorijsko predstavljanje objekata (višestruko nasleđivanje) - primer

```
class A                          { int a = 0; }
class B                          { int b = 0;
                                   int c = 0; }
class C extends A                { int d = 0; }
class D extends A,B,C            { int e = 0; }
```

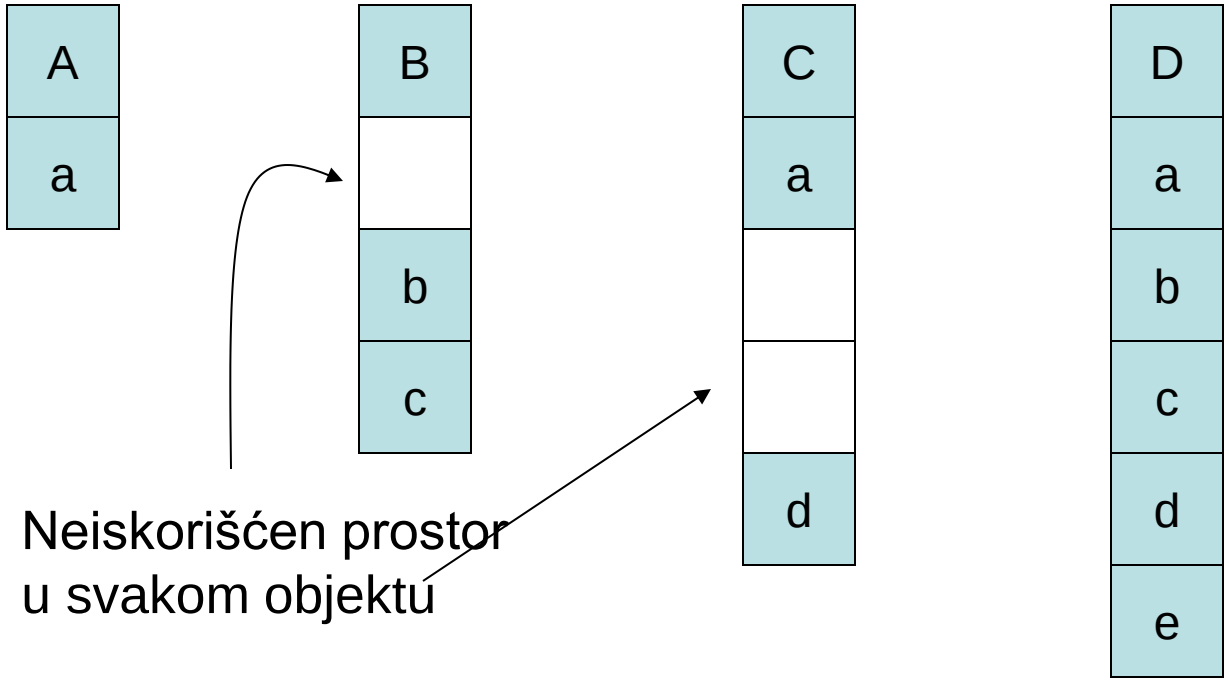


Mem. predstavljanje objekata može se odrediti algoritmom bojenja grafa

- čvor za svako posebno ime polja
- grana za polja koja koegzistiraju u istoj klasi (putem nasleđivanja ili drugačije)
- pomeraji u zapisu odgovaraju bojama grafa

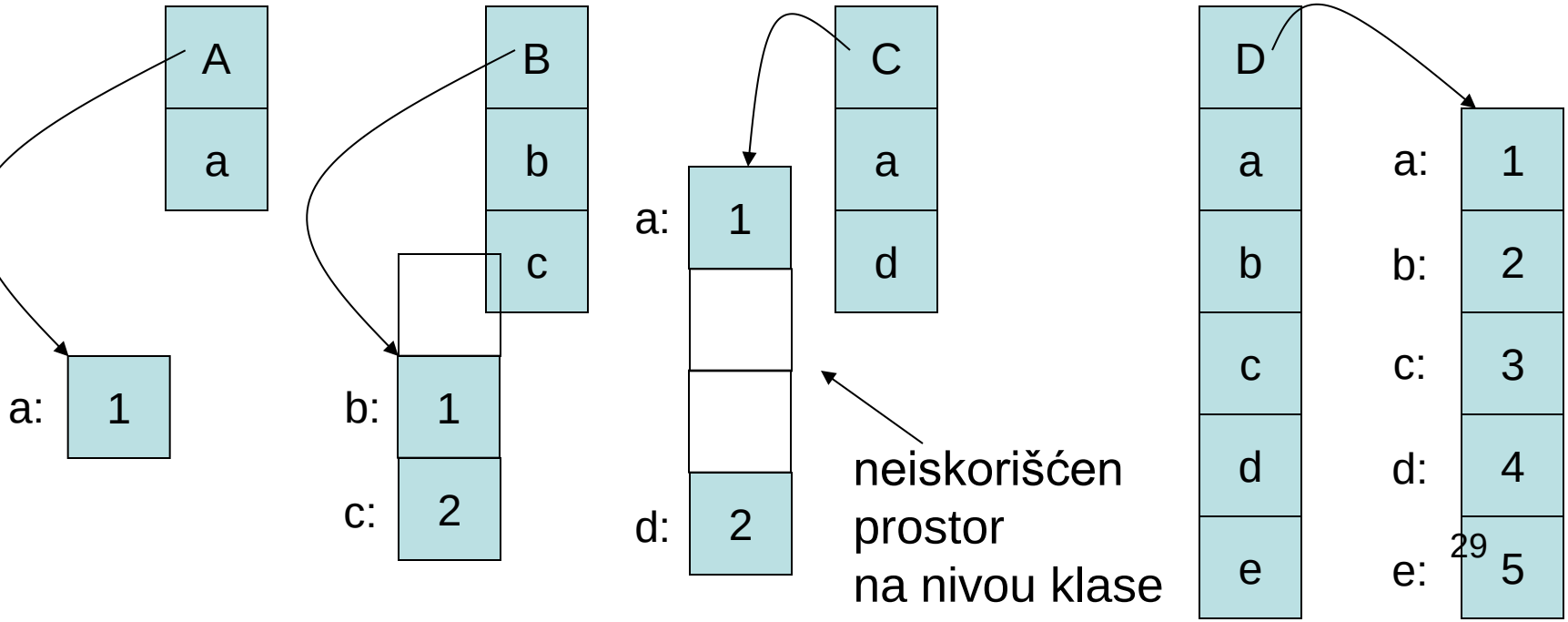
Memorijsko predstavljanje objekata (višestruko nasleđivanje) - primer

```
class A                          { int a = 0; }
class B                          { int b = 0;
                                   int c = 0; }
class C extends A                { int d = 0; }
class D extends A,B,C            { int e = 0; }
```



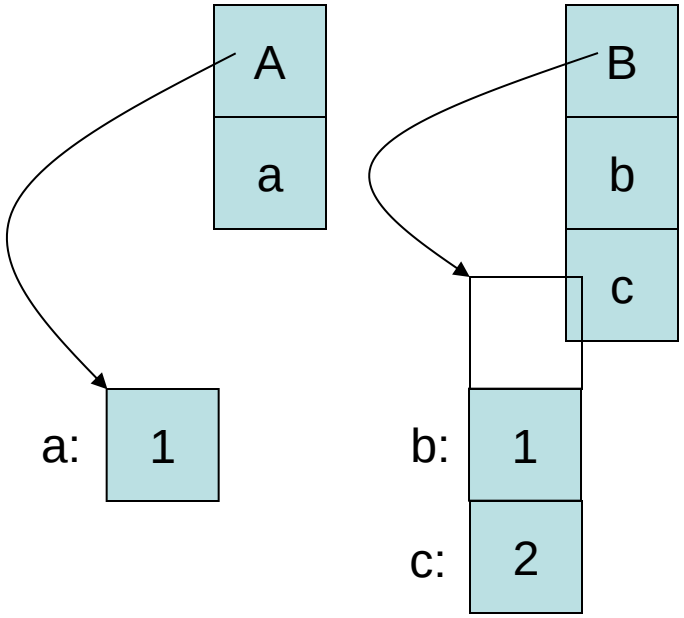
Memorijsko predstavljanje objekata (višestruko nasleđivanje) – primer 2

```
class A                          { int a = 0; }
class B                          { int b = 0;
                                   int c = 0; }
class C extends A                { int d = 0; }
class D extends A,B,C            { int e = 0; }
```



Memorijsko predstavljanje objekata (višestruko nasleđivanje) – primer 2

```
class A                          { int a = 0; }
class B                          { int b = 0;
                                   int c = 0; }
class C extends A                { int d = 0; }
class D extends A,B,C            { int e = 0; }
```



Za pristup polju prema ovom rasporedu, treba:

1. p = prvo polje objekta (deskriptor klase) c
2. M = c + fieldOffset
3. load M

Problem

- Globalna analiza programa izbegava se u modernom računarstvu
  - mnoge aplikacije koriste dinamički linkovane biblioteke, zakrpe za ispravku grešaka itd.
  - Kada nemamo ceo program u vreme prevođenja, ne možemo vršiti globalnu analizu!
  - Iz tog razloga, Java ima jednostruko nasleđivanje
  - Moguće rešenje: **hešing**
    - Umesto globalnog bojenja grafa, u svaki deskriptor klase ide heš tabela koja mapira imena polja u ofsete i imena metoda u adrese instanci metoda. Ovo se slaže sa odvojenim prevođenjem i dinamičkim povezivanjem.

Ostale OO specifičnosti

- Mehanizam prava pristupa
  - radi enkapsulacije lokalnog stanja objekta, Java ima kvalifikatore “private” “protected” i “public”
    - privatni metodi/polja ne mogu biti korišćeni izvan klase gde su definisani
  - Ove informacije treba čuvati u tabeli simbola i proveravati u semantičkoj analizi

Ostale OO specifičnosti

- Provera tipova
  - Provera da li objekat pripada klasi C, ili bilo kojoj njenoj podklasi.
    - X instanceof C (Java)
    - Typeid operator (C++ RTTI)
- Sigurna konverzija tipa (Safe casting)
  - Za datu promenljivu x klase C, gde x u stvari ukazuje na objekat klase D koja proširuje C, na sledeće načine se dobija izraz tipa D u vreme prevođenja:
    - (D)x (Java)
    - dynamic_cast < *D* > (*x*) (C++ RTTI)

Implementacija provere tipova

- Pošto svaki objekat pokazuje na svoj deskriptor klase, adresa deskriptora klase može služiti za identifikaciju tipa. Međutim, ako je x instanca klase D, a D proširuje C, onda je x takođe instanca klase C.
- Jednostavan način implementacije x instanceof C:

```
    T1 ← x.descriptor;
L1: if t1 = C goto true;
    t1 ← t1.super;
    if t1 = nil goto false;
goto L1;
```