Hijerarhijska dekompozicija

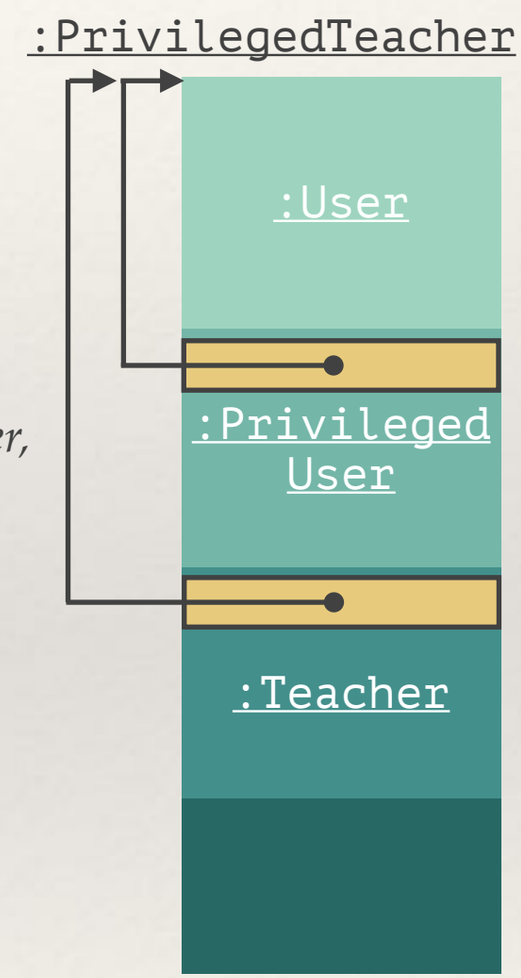
- Da bi se rešio ovaj problem, osnovna klasa *User* treba da se deklariše kao tzv. *virtuelna osnovna klasa*:

```
class User {…};

class PrivilegedUser : virtual public User {…};
class NonprivilegedUser : virtual public User {…};

class Teacher : virtual public User {…};
class Student : virtual public User {…};

class PrivilegedTeacher : public PrivilegedUser, public Teacher {…};
```



- Značenje virtuelne osnovne klase je sledeće: ako se objekat neke klase (*PrivilegedUser, Teacher, …*) nađe kao podobjekat unutar objekta neke izvedene klase (*PrivilegedTeacher*), u njega ugrađen podobjekat virtuelne osnovne klase (*User*) će biti zajednički (deljen) za sve druge podobjekte unutar tog objekta koji takođe imaju tu osnovnu klasu kao virtuelnu
- Ovo se implementira na sledeći način: unutar objekta izvedene klase postoji pokazivač na podobjekat virtuelne osnovne klase, jer on ne mora uvek biti na unapred poznatom mestu unutar tog objekta (na njegovom početku), pošto se može nalaziti unutar nekog drugog objekta sa kojim se deli
- Zbog toga prevodilac ne može generisati kod za pristup članovima virtuelne osnovne klase statičkim vezivanjem (adresiranjem preko adresa, tj. pomeraja u odnosu na adresu početka objekta poznatim u vreme prevođenja), već dinamičkim vezivanjem, preko ovog pokazivača

Hijerarhijska dekompozicija

- Kod višestrukog izvođenja, stvari se dosta komplikuju:
  - Konstruktori osnovnih klasa se svakako pozivaju unutar konstruktora izvedenih klasa, ali kojim redom? - Po redosledu navođenja tih osnovnih klasa u definiciji izvedene klase; prema tome, zbog ugnežđenih poziva, redosled poziva konstruktora je uvek po dubini grafa, sleva nadesno (posmatrano po redosledu navođenja osnovnih klasa)
  - Međutim, konstruktori virtuelnih osnovnih klasa se pozivaju pre konstruktora ostalih (nevirtuelnih) klasa, opet po određenom redosledu, ali je to pravilo sada već teško za pamćenje i razumevanje programa, pa ako logika programa zavisi od tog redosleda, takav program postaje težak za razumevanje
  - Konverzije pokazivača imaju komplikovaniju implementaciju, rezultati tih konverzija mogu davati različite vrednosti pokazivača, ali je dinamička konverzija (pomoću *dynamic_cast*) uvek ispravna i bezbedna i može se raditi nagore, nadole i bočno po grafu izvođenja
- Ovakve situacije u kojima je potrebno raditi višestruko nasleđivanje klasa su u praksi ipak retke. Čak i ako se na njih naiđe, mogu se rešiti nekim drugim načinom, npr. atributima ili vezama sa objektima različitih klasa
- Zbog toga mnogi drugi jezici ne podržavaju višestruko nasleđivanje/izvođenje
- Međutim, višestruko izvođenje jeste korisno kada se upotrebljava za implementaciju različitih interfejsa i *mixin* klase
- Kako bi podržali ovu važnu upotrebu, a ipak pojednostavili implementaciju i logiku jezika, mnogi drugi jezici podržavaju ograničen koncept *interfejsa*, koji može imati samo polimorfne operacije (ali ne i strukturu, atribute), sa metodama ili bez njih (apstraktne operacije)
- U takvim jezicima (npr. Java) klasa može *naslediti* (*proširiti*, *extend*) samo jednu osnovnu klasu (nasleđujući njenu strukturu i ugrađujući samo jedan podobjekat osnovne klase), ali može *implementirati* (*implement*) više interfejsa; pošto interfejsi tu imaju samo polimorfne operacije, implementacija se svodi na jednostavno dinamičko vezivanje preko VTP