Hijerarhijska dekompozicija

- Ako pak osnovna klasa nije javna, već *zaštićena* (*protected*) ili *privatna* (*private*), objekat izvedene klase i dalje će u sebi imati ugrađen podobjekat te osnovne klase, ali on neće biti dostupan van te klase, osim u izvedenim klasama ako je zaštićen:
  - implicitna konverzija pokazivača (ili reference) na objekat izvedene klase u pokazivač (ili referencu) na osnovnu klasu dozvoljena je samo u toj klasi (i u izvedenoj klasi, za zaštićeno izvođenje)
  - javni i zaštićeni članovi osnovne klase dostupni su samo u toj izvedenoj klasi, ali ne i van nje (osim u izvedenim klasama za zaštićeno izvođenje):

```
class Task : protected ListElem {…};

Task* aTask = …;

List* taskList = …;

taskList->addAtTail(aTask);
```

Ovo više nije moguće van klase *Task* ili njene izvedene klase

- Ali unutar te izvedene klase (*Task*), može se vršiti ova implicitna konverzija, čime klasa zapravo nudi svoj odgovarajući interfejs potrebnom kontekstu:

```
void Task::addToTaskList(List* taskList) {
  if (taskList) taskList->addAtTail(this);
}
```

Implicitna konverzija pokazivača *this* tipa *Task** u pokazivač tipa *ListElem** na zaštićenu (ili privatnu) osnovnu klasu dozvoljena je u toj izvedenoj klasi

- Prema tome, kod zaštićenog i privatnog izvođenja, ne važi pravilo supstitucije u celom programu i sa objektima izvedene klase ne može se uvek i svuda uraditi sve što i sa objektima osnovne klase, pa ovakvo izvođenje klasa na jeziku C++ *ne realizuje nasleđivanje* kao objektni koncept, već:
  - predstavlja relaciju ugrađivanja objekta jedne klase u objekte druge klase
  - može se koristiti za kontrolisanu i skrivenu ugradnju *mixin* klasa, uz otkrivanje tih interfejsa samo u određenim ograničenim delovima programa radi strožije enkapsulacije

Hijerarhijska dekompozicija

- Objekat izvedene klase u sebi sadrži podobjekat svake osnovne klase; i tako rekurzivno, ako ta osnovna klasa ima svoje osnovne klase, taj podobojekat imaće u sebi po jedan podobjekat svake od tih daljih osnovnih klasa itd. Kako je operativna memorija linearna, da bi se objekat klase izvedene iz više osnovnih klasa smestio u nju, ovi podobjekti se moraju poređati, i to po redosledu navođenja osnovnih klasa u definiciji izvedene klase:

```
class D : public B1, public B2 {…};
```

- Kada se pravi objekat izvedene klase (*D*), poziva se konstruktor te klase. Ali svaki konstruktor izvedene klase uvek poziva (pre izvršavanja svog tela) konstruktor osnovne klase; taj poziv obezbeđuje prevodilac u generisanom kodu za konstruktor izvedene klase. Analogno, konstruktor osnovne klase u sebi ima poziv konstruktora svoje osnovne klase itd.

- U slučaju višestrukog izvođenja, konstruktori osnovnih klasa pozivaju se po redosledu navođenja tih klasa u definiciji izvedene klase, bez obzira na to kako su ti pozivi navedeni:

```
D::D (…) : B1(…), B2(…) {
 …
}
```