Hijerarhijska dekompozicija

- Na taj način, ako je osnovna klasa u zaglavlju definicije izvedene klase označena kao *javna* (*public*) osnovna klasa, ovaj podobjekat osnovne klase dostupan je (implicitnom) konverzijom pokazivača (ili reference) na tu izvedenu klasu u pokazivač (ili referencu) na tu osnovnu klasu na svim mestima gde su dostupne i te klase:

```
class ListElem {…};

class Task : public ListElem {…};

Task* aTask = …;

List* taskList = …;

taskList->addAtTail(aTask);
```

Implicitna konverzija pokazivača *aTask* tipa *Task** u pokazivač tipa *ListElem** na javnu osnovnu klasu

- Osim toga, svi javni članovi javne osnovne klase jesu i javni članovi izvedene klase, pa se može raditi npr:

```
aTask->insert(…)
```

- Zaštićeni članovi osnovne klase dostupni su u izvedenoj klasi, ali ne i van nje i ostaju zaštićeni i dalje; privatni članovi osnovne klase nisu dostupni u izvedenoj klasi
- Dakle, važi pravilo supstitucije i sve što se može uraditi sa objektom osnovne klase može se uraditi i sa objektom izvedene klase - *javno izvođenje klasa* na jeziku C++ jeste jezički koncept koji realizuje *nasleđivanje* kao objektni koncept

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);
```

- 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);
}
```

- 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