Hijerarhijska dekompozicija

- Klase imaju interpretaciju i u skupovnoj matematičkoj logici:
  - klasa je skup, objekat je element; objekat *x* je instanca klase *X* <=> *x* je element skupa *X*
  - instanca izvedene klase je uvek (indirektno) i instanca osnovne klase: ako je objekat *x* instanca izvedene klase *D* (*x* je element *D*), iz toga sledi da je *x* i instanca osnovne klase *B* (*x* je element *B*)
  - prema tome: osnovna klasa je nadskup, izvedena klasa je podskup
- Iz svega toga sledi jedan od fundamentalnih principa objektnog programiranja, *princip supstitucije* (*Liskov substitution principle*, Barbara Liskov, 1994): instance izvedene klase *D* mogu se pojaviti i upotrebiti gde god i kad god se očekuju instance osnovne klase *B* - instance izvedene klase mogu biti supstituti (zamene) za instance osnovne klase, bez ugrožavanja bilo kog željenog ponašanja programa
- Ovaj princip je posledica semantike nasleđivanja: kako objekti izvedene klase nasleđuju sve osobine objekata osnovne klase, i za njih važe sve tvrdnje koje važe za objekte osnovne klase, sa njima se može raditi sve što i sa objektima osnovne klase; zapravo, oni su zato instance te osnovne klase

Hijerarhijska dekompozicija

- Svi savremeni OO jezici, pa i C++, podržavaju ovaj princip sledećim pravilom *implicitne konverzije*: pokazivač/referenca na izvedenu klasu može se konvertovati (implicitno, bez eksplicitnog zahteva) u pokazivač/referencu na osnovnu klasu - tzv. "kalupljenje nagore" (*upcast*):

`DerivedClass*` → `BaseClass*`

`DerivedClass&` → `BaseClass&`

- Upravo ova konverzija omogućava da se objektima konkretnih, izvedenih klasa pristupa kao instancama osnovnih, generalizovanih klasa
- Informacija o konkretnom tipu objekta (klasi čija je on direktna instanca) treba da bude što manje bitna i poznata ostatku softvera; ta informacija se može zanemariti odmah nakon kreiranja objekta:

```
Figure* fig = new Circle(…);
```

- Suprotna konverzija, nadole (*downcast*), nije uvek bezbedna, jer objekat osnovne klase ne mora biti i instanca neke izvedene klase; pošto prevodilac ne može da proveri tu činjenicu, ovakva konverzija ne može se raditi implicitno:

```
Figure* fig = new Circle(…);
Circle* crc = fig;
```

ali može eksplicitno:

```
Circle* crc = (Circle*)fig;
```

- Prevodilac generiše kod za pristup objektu te izvedene preko tog pokazivača, bez ikakvih dodatnih provera. Na ovaj način, programer preuzima odgovornost da se iza pokazivača na osnovnu klasu zaista krije objekat tražene izvedene klase. Ako ovo nije zadovoljeno, program će se ponašati potpuno nepredvidivo u vreme izvršavanja (nepredvidive posledice: greška u logici, poremećaj podataka ili izuzetak na nivou hardvera ili operativnog sistema zbog neovlašćenog pristupa delu memorije)