Enkapsulacija

- Statička funkcija članica može da se pozove bez navođenja objekta za koji se poziva (mada taj objekat može i da se navede):

```
int num = Clock::getCount ();
```

- Statičke funkcije članice implementiraju se isto kao i globalne funkcije nečlanice, jer nemaju pokazivač *this*; zbog toga se umesto statičke funkcije članice može upotrebiti i globalna funkcija, ali je korišćenje statičkih funkcija članica bolje u mnogim slučajevima, jer je statička funkcija članica:
  - deo klase kao logičke celine, logički je "upakovana" u nju, pa je jasnija njena upotreba i namena - program je čitljiviji i lakši za razumevanje
  - u oblasti važenja klase, a nije globalna, pa se ne sukobljava po imenu (*name clashing*) sa ostalim globalnim imenima (može da se zove isto)
  - članica klase, kao i svaki drugi član, pa se može enkapsulirati: ona može da bude zaštićena ili privatna
  - članica klase, pa ima pravo pristupa do privatnih i zaštićenih članova te klase (globalna bi morala da bude prijatelj toj klasi)
- Zbog toga, neki noviji jezici (npr. Java) i ne omogućavaju globalne operacije, a umesto njih podržavaju statičke operacije: svaka operacija mora biti članica neke klase (statička ili nestatička)

Hijerarhijska dekompozicija

- *Hijerarhijska dekompozicija* (*hierarchical decomposition*) je još jedan element objektne dekompozicije i podrazumeva kreiranje hijerarhija klasa povezanih relacijama *nasleđivanja* (*inheritance*)
- Relacija nasleđivanja označava da izvedena klasa *nasleđuje* sve osobine osnovne klase i to:
  - *semantiku* (značenje): svaka tvrdnja ili ograničenje koje važi za instance osnovne klase, važi i za instance izvedene klase (obrnuto ne mora)
  - *intefejs*: sa objektima izvedene klase može se raditi sve što i sa objektima osnovne klase (obrnuto ne mora)
  - *svojstva* i *ponašanje*: objekti izvedene klase poseduju i strukturu i ponašanje definisano u implementaciji osnovne klase, s tim što ih mogu redefinisati i proširiti
- Ovakve relacije mogu se ravnopravno otkrivati ili osmišljavati u oba smera:
  - *specijalizacija* (*specialization*): izvedena klasa predstavlja konkretizaciju, posebnu ili pojedinačnu potkategoriju (podskup) instanci osnovne klase koje imaju neke specifičnosti, redefinišu, variraju i/ili specijalizuju ponašanje
  - *generalizacija* (*generalization*): osnovna klasa je apstrakcija, generalizacija, uopštenje više izvedenih klasa, unija njihovih instanci i generalizuje (uopštava) njihove interfejse i prikuplja zajednička svojstva
- Prema tome, ista relacija između dve klase može se posmatrati u oba smera, pa se na jeziku UML ona i naziva relacija *generalizacije/specijalizacije*