Hijerarhijska dekompozicija

- Jedna od osnovnih tendencija u OO programiranju jeste upravo generalizacija: apstrahovanje zajedničkih svojstava i formiranje uopštenih interfejsa za klijente, kako bi oni bili što manje zavisni od specifičnosti pojedinačnih slučajeva
- Specifičnosti se onda sakrivaju iza polimorfnih operacija i ugrađuju u njihove različite implementacije (metode) u izvedenim klasama
- Na ovaj način se sprege između delova softvera čine labavijim, jednostavnijim za kontrolu, a klijenti čine nezavisnijim i time fleksibilnijim (“*the less you know, the better*”)
- Ovo se ponekad naziva *principom inverzije zavisnosti* (*dependency inversion principle*):
  - u proceduralnom programiranju, u algoritamskoj dekompoziciji, potprogram (apstrakcija) na višem nivou i krupnije granularnosti zavisi od potprograma (apstrakcije) na nižem nivou, jer je poziva kao deo svoje implementacije
  - u OOP, tendencija je da jedna apstrakcija, klijent (*Drawing*) zavisi od druge uopštene apstrakcije na višem nivou (*Figure*), a da specijalizacije (*Rectangle*, *Circle*, …), kao pojedinačni slučajevi, zavise od opštije apstrakcije (*Figure*) - obrnuta zavisnost
- Generalizacijom i korišćenjem polimorfizma se drastično smanjuje količina uslovnih grananja na osnovu tipa objekta kojim se rukuje

Hijerarhijska dekompozicija

- Klasa *Figure* neće imati svoje direktne instance, već samo indirektne instance (koje su direktne instance izvedenih klasa)
- Ovakva klasa se naziva *apstraktnom klasom* (*abstract class*) ili apstraktnom generalizacijom
- Neki jezici omogućavaju da se apstraktna klasa posebno označi
- Na jeziku C++ nema posebne oznake za apstraktnu klasu, ali se pravljenje objekata te klase (direktnih instanci) može sprečiti deklarisanjem *svih* postojećih konstruktora zaštićenim (*protected*). Zašto baš zaštičenim?
  - ne mogu se kreirati nezavisni objekti te klase, jer konstruktor nije javan (*public*), pa bi prevodilac generisao grešku pri svakom pokušaju kreiranja tog objekta van te klase
  - kada se pravi objekat izvedene klase, poziva se konstruktor te izvedene klase; ali svaki konstruktor izvedene klase obavezno poziva neki konstruktor osnovne klase; ako bi on bio privatan (*private*), ne bi bio dostupan ni u izvedenim klasama, pa se data klasa ne bi mogla nasleđivati