# Raspodela odgovornosti

- Jedan od osnovnih principa softverskog inženjerstva, srodan sa navedenim, jeste princip *lokalizacije projektnih odluka* (*localization of design decisions*): manifestacija neke projektne odluke u programu treba da bude *lokalizovana* na jednom mestu, a ne rasuta na više mesta
- Ovo stoga što bi promena te odluke bila teška za sprovođenje ukoliko su njene manifestacije rasute po programu: izmena je teška i podložna greškama, a rizik domino efekta veliki; lokalizacija projektne odluke olakšava njenu promenu i povećava šansu da ta promena prođe bez problema
- Projektna odluka može da bude različite vrste, složenosti i nivoa apstrakcije: od najprostije odluke da je neka promenljiva nekog tipa ili neka konstanta ima neku vrednost, pa sve do odluke o načinu sprovođenja nekog scenarija u programu ili o arhitekturi datog softvera
- Najjednostavniji primer jeste korišćenje konstanti u programu:
  - korišćenje neposrednog literala (npr. broja 100) za neku konstantu (npr. dimenziju nekog niza) na više mesta u kodu (npr. prilikom deklarisanja niza i prilikom iteriranja kroz niz do njegove granice) predstavlja prestup ovog principa: promenu odluke da niz bude baš te veličine na neku drugu: a) zahteva prolazak kroz kod i sva mesta korišćenja tog literala i pažljivu zamenu drugom vrednošću, b) podložno je greškama, jer se na nekom mestu ista ta vrednost (npr. 100) može koristiti sa potpuno drugim značenjem, i ovakvom zamenom pogrešno promeniti
  - umesto toga, uvođenje simboličke konstante koja ima željenu vrednost predstavlja doslednu primenu ovog principa: odluka da je data dimenzija baš određena je lokalizovana na jednom mestu - u deklaraciji te konstante
- Naravno, i ovo ima ograničene domete i važi samo za odluke koje imaju perspektivu da se promene: celokupnu arhitekturu datog softvera nije lako promeniti, dok se neke činjenice verovatno neće promeniti za života softvera (npr. činjenica da sat ima 60 minuta ili sedmica 7 dana itd.)
- Klasa može da bude način i mesto za lokalizaciju projektnih odluka određenog nivoa granularnosti i apstrakcije: raspodela neke odgovornosti u neku klasu jeste jedna pojava lokalizacije projektne odluke

# Algoritamska dekompozicija

- Fundament proceduralnog programiranja jeste *algoritamska* (*proceduralna*) *dekompozicija* (*algorithmic, procedural decomposition*):
  - u centru pažnje jeste zadatak ili posao koji treba uraditi, za koji se definiše *postupak* (*procedura, algoritam*)
  - taj postupak se inicijalno deli na *korake*, najpre visokog nivoa apstrakcije i velike granularnosti ("da bi se to uradilo, potrebno je najpre uraditi *a*, pa onda *b*, potom, ako je ispunjen uslov *c*, treba uraditi *d*, inače *e* itd); ovi koraci definišu se kao *procedure* (potprogrami)
    - dalje se svaki krupniji korak deli istim postupkom na manje korake (rekurzivna primena istog postupka dekompozicije), sve dok se ne dođe do najsitnijeg nivoa granularnosti, onog koji se može izraziti elementarnim konstruktima programskog jezika koji se koristi
- Procedure su takođe proizvod apstrakcije: određeni (krupniji) korak se apstrahuje kao celina, pri čemu se njegova razrada i detalji odlažu za kasnije, jer za korišćenje nisu bitni (bitan je samo interfejs)
- Primer: Program koji održava *n* stek-mašina koje treba da izvršavaju komande (operacije *add*, *sub*, *push*, *pop* itd). Na ulazu se nalaze komande u tekstualnom obliku, svaka u po jednom redu. Svaka komanda je u formatu *i*:*command*, gde je *i* redni broj stek-mašine na koju se odnosi komanda u nastavku. Sa ulaza se učitava jedan po jedan znak operacijom *getChar*. Zapis komande sa ulaza treba prevesti pre nego što se izvrši.