Hijerarhijska dekompozicija

- Kod višestrukog izvođenja, stvari se dosta komplikuju:
  - Konstruktori osnovnih klasa se svakako pozivaju unutar konstruktora izvedenih klasa, ali kojim redom? - Po redosledu navođenja tih osnovnih klasa u definiciji izvedene klase; prema tome, zbog ugnežđenih poziva, redosled poziva konstruktora je uvek po dubini grafa, sleva nadesno (posmatrano po redosledu navođenja osnovnih klasa)
  - Međutim, konstruktori virtuelnih osnovnih klasa se pozivaju pre konstruktora ostalih (nevirtuelnih) klasa, opet po određenom redosledu, ali je to pravilo sada već teško za pamćenje i razumevanje programa, pa ako logika programa zavisi od tog redosleda, takav program postaje težak za razumevanje
  - Konverzije pokazivača imaju komplikovaniju implementaciju, rezultati tih konverzija mogu davati različite vrednosti pokazivača, ali je dinamička konverzija (pomoću *dynamic_cast*) uvek ispravna i bezbedna i može se raditi nagore, nadole i bočno po grafu izvođenja
- Ovakve situacije u kojima je potrebno raditi višestruko nasleđivanje klasa su u praksi ipak retke. Čak i ako se na njih naiđe, mogu se rešiti nekim drugim načinom, npr. atributima ili vezama sa objektima različitih klasa
- Zbog toga mnogi drugi jezici ne podržavaju višestruko nasleđivanje/izvođenje
- Međutim, višestruko izvođenje jeste korisno kada se upotrebljava za implementaciju različitih interfejsa i *mixin* klase
- Kako bi podržali ovu važnu upotrebu, a ipak pojednostavili implementaciju i logiku jezika, mnogi drugi jezici podržavaju ograničen koncept *interfejsa*, koji može imati samo polimorfne operacije (ali ne i strukturu, atribute), sa metodama ili bez njih (apstraktne operacije)
- U takvim jezicima (npr. Java) klasa može *naslediti* (*proširiti*, *extend*) samo jednu osnovnu klasu (nasleđujući njenu strukturu i ugrađujući samo jedan podobjekat osnovne klase), ali može *implementirati* (*implement*) više interfejsa; pošto interfejsi tu imaju samo polimorfne operacije, implementacija se svodi na jednostavno dinamičko vezivanje preko VTP

Obrada izuzetaka

- Važno svojstvo robusnih složenih softverskih sistema jeste *tolerancija otkaza* (*fault tolerance*): sposobnost da softver nastavi svoje izvršavanje u prisustvu *otkaza* (*faults*)
- Otkazi su odstupanja softvera ili njegovog okruženja od specifikovanog, predviđenog ili očekivanog ponašanja
- Softver koji nema mehanizme tolerancije otkaza će u slučaju otkaza ili u potpunosti prekinuti svoj rad ("pad", *failure*) ili nastaviti izvršavanje sa nepredvidivim ili neželjenim ponašanjem
- Softver koji ima ugrađene mehanizme tolerancije otkaza će u slučaju otkaza nastaviti svoje izvršavanje uz zadovljavajuće i predviđeno ponašanje, eventualno uz degradirane neke funkcionalnosti i/ili performanse, ili preduzeti kontrolisane procedure za sopstveno gašenje i gašenje sistema kojim upravlja
- Jedan od osnovnih mehanizama u hardveru i softveru za toleranciju otkaza jeste mehanizam *obrade izuzetaka* (*exception handling*)
- *Izuzetak* (*exception*) je nastanak nenormalnih, neregularnih ili neočekivanih uslova koji zahtevaju poseban tretman, različit od onog predviđenog normalnim, regularnim tokom programa