Projektovanje softvera

Iterator

Iterator (1)

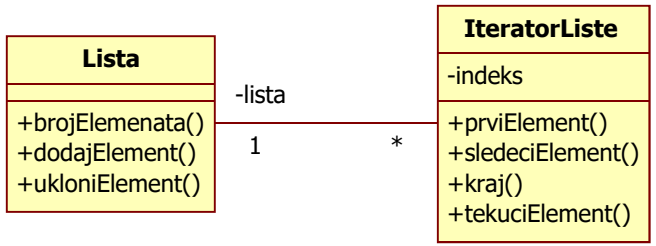
- Ime i klasifikacija:
  - Iterator (engl. *Iterator*) – objektni uzorak ponašanja
- Namena:
  - obezbeđuje pristup elementima zbirke (agregata) redom, bez eksponiranja interne strukture te zbirke
- Drugo ime:
  - Kurzor (engl. *Cursor*), za jednu specifičnu vrstu iteratora

Iterator (2)

- Motivacija:
  - lista treba da omogući pristup njenim elementima
  - interna struktura liste ne treba da bude vidljiva klijentima
  - moguće je da postoji više načina za obilazak liste
    - klasa liste ne treba da se optereti operacijama za razne načine obilazaka
    - ni klijent ne treba da se optereti poznavanjem načina obilaska liste
  - potrebno je da više klijenata simultano obilazi listu
  - ključna ideja uzorka *Iterator:*
    - prenošenje odgovornosti za obilazak liste na poseban objekat – iterator
  - klasa iteratora definiše ugovor za pristup elementima liste
  - objekat iteratora poznaje način obilaska liste i to:
    - od kog elementa početi obilazak
    - koji su elementi liste već posećeni, odnosno koji je naredni na redu
    - kada je obilazak završen, odnosno kada više nema neobiđenih elemenata
  - objekat iteratora čuva informaciju o tekućem elementu obilaska

Iterator (3)

- Motivacija (nastavak):
  - relacija između klasa `Lista` i `IteratorListe`:



  - `prviElement()` - inicijalizuje pokazivač tekućeg elementa postavljanjem na prvi element
  - `sledeciElement()` - proglašava naredni element tekućim
  - `kraj()` - testira da li je obilazak završen
  - `tekuciElement()` - vraća tekući element liste

Iterator (4)

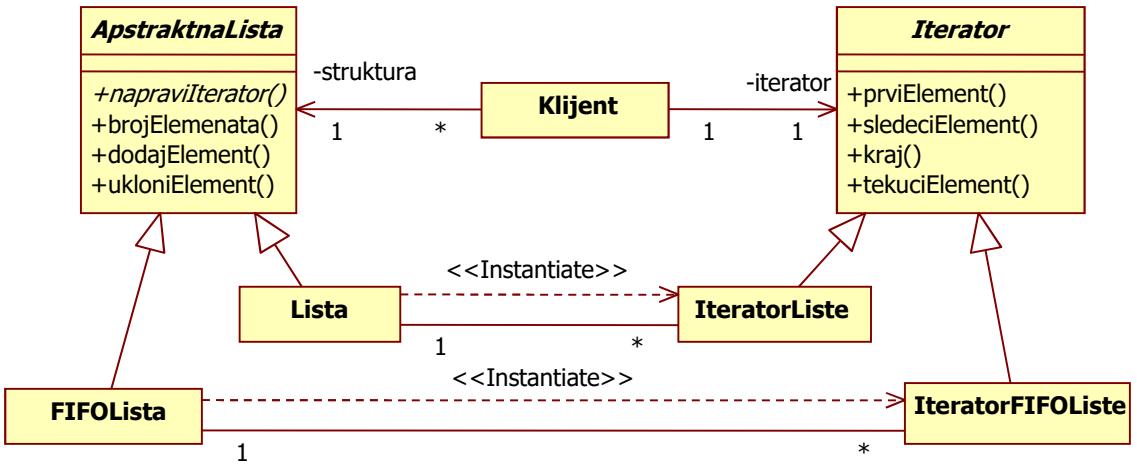
- Motivacija (nastavak):
  - razdvajanje mehanizma obilaska od objekta liste omogućava definisanje posebnih iteratora za različite načine obilaska
    - na primer: `IteratorFiltriraneListe` može da pruža pristup samo onim elementima koji zadovoljavaju uslove filtra
    - ugovor klase `Lista` nije opterećen operacijama za razne načine obilaska
  - nedostatak:
    - klijent mora da bude svestan da se obilazi baš konkretna vrsta liste i da se za nju koristi baš konkretna vrsta iteratora
    - za konkretnu vrstu liste klijent mora da napravi objekat iteratora odgovarjuće klase za tu listu
    - ukoliko bi se menjala vrsta liste koja se obilazi ili samo način obilaska liste – morao bi da se menja kod klijenta za stvaranje iteratora
  - ideja za prevazilaženje nedostatka:
    - da lista bude odgovorna za stvaranje iteratora za njen obilazak

Iterator (5)

- Motivacija (nastavak):
  - generalizacija koncepta iteratora: *polimorfni iterator*
  - primer:
    - klijent treba da radi sa klasom `Lista` ali i sa klasom `FIFOLista`
  - definiše se apstraktna klasa *`ApstraktnaLista`*
    - za manipulisanje proizvoljnom listom
  - definiše se apstraktna klasa *`Iterator`*
    - za objekte iteratora svih vrsta lista
  - potklase koje implementiraju ugovor klase *`Iterator`*
    biće iteratori za odgovarajuće liste
  - klasa konkretne liste je odgovorna za stvaranje odgovarajućeg iteratora
  - mehanizam je postao nezavisan od konkretnih klasa lista
    - klijent ne mora da vodi računa o vrsti liste
      da bi za nju napravio odgovarajući objekat iteratora

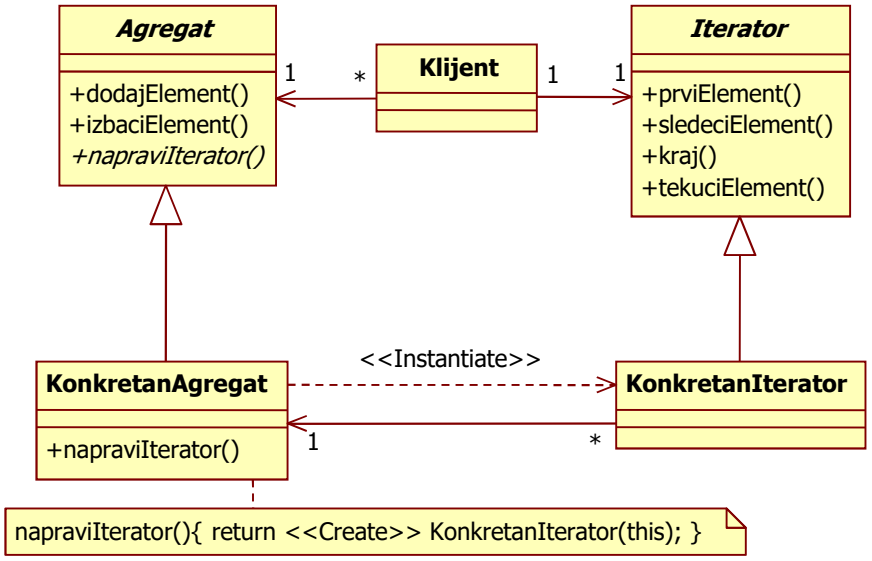
Iterator (6)

- Motivacija (nastavak):
  - da bi se klijent učinio nezavisnim od konkretne liste/iteratora:
    - definiše se apstraktna operacija liste `napraviIterator()` kojom stvara svoj iterator
    - klijenti koriste apstraktnu operaciju liste i tako su nezavisni od konkretnih klasa liste
    - klijenti koriste i ugovor klase *`Iterator`* i tako su nezavisni od konkretnih iteratora
    - operacija `napraviIterator()` je primer *Fabričkog metoda*
    - ostale operacije mogu da budu konkretne za podrazumevanu implementaciju liste



Iterator (7)

- Primenljivost: uzorak treba da se koristi
  - da se podrže višestruki simultani obilasci agregata
  - da se obezbedi uniformni ugovor za obilazak različitih agregata
  - da se ni klijent ni agregat ne opterete načinom obilaska
- Struktura:

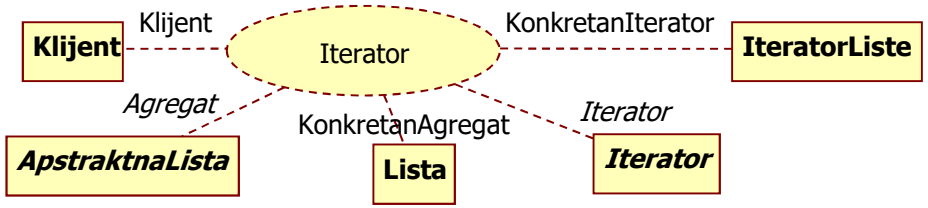


Iterator (8)

- Učesnici:
  - *Agregat* (klasa *ApstraktnaLista*)
    - definiše ugovor za stvaranje objekta iteratora
  - *Iterator* (klasa *Iterator*)
    - definiše ugovor za obilazak elemenata agregata i pristup tekućem elementu
  - KonkretanAgregat (klase Lista, FIFOLista)
    - implementira ugovor za stvaranje iteratora tako što vraća odgovarajući konkretan iterator
  - KonkretanIterator (klase IteratorListe, IteratorFIFOListe)
    - nadjačava podrazumevanu implementaciju ugovora klase *Iterator*
    - čuva informaciju o tekućem elementu (njegovoj poziciji) pri obilasku agregata
- Saradnja:
  - klijent zahteva od agregata da napravi odgovarajući iterator
  - klijent se obraća konkretnom iteratoru
    - za pozicioniranje na prvi i svaki sledeći element agregata i proveru kraja obilaska
    - za dohvatanje tekućeg elementa agregata

Iterator (9)

- UML notacija:



- Posledice:
  - *Iterator* ima sledeće dobre strane:
    - pojednostavljuje interfejs agregata: interfejs za obilazak je u klasi iteratora
    - više od jednog obilaska može simultano da se sprovodi nad agregatom
    - podržava varijacije u obilasku agregata:
      - lako se dodaje nova klasa konkretnog iteratora za novi način obilaska

Iterator (10)

- Implementacija:
  - ko upravlja iterativnim procesom?
    - Spoljašnji (*external*) iterator – klijent kontroliše iterativni proces
      - na klijentu je odgovornost za progres obilaska
      - eksplicitno zahteva od iteratora pomeranje na sledeći element
      - iterator o kojem je ovde bilo reči je spoljašnji iterator
    - Unutrašnji (*internal*) iterator – iterator kontroliše iterativni proces
      - klijent samo zahteva od iteratora da izvrši neku operaciju
      - sam iterator primenjuje operaciju na svaki element agregata
    - spoljašnji iterator je fleksibilniji od unutrašnjeg

Iterator (11)

- Implementacija (nastavak):
  - ko definiše način (algoritam, politiku) obilaska?
    - po pravilu – iterator, ali ne i obavezno
    - agregat može da definiše algoritam obilaska
      - agregat ima operacije za obilazak i dohvatanje: `prviElement()`,...
    - iterator se tada koristi samo da čuva stanje iteracije
      - samo ukazuje na poziciju tekućeg elementa u agregatu
      - ovakav iterator se naziva *Kurzor* (*Cursor*)
    - klijent poziva operaciju `sledeciElement()` sa kurzorom kao argumentom
      - operacija `sledeciElement()` promeni stanje objekta kurzor-objekta
    - ako se algoritam obilasaka definiše u klasi agregata gube se povoljnosti:
      - lakog variranja iterativnih algoritama nad istim agregatom
      - reupotrebe algoritma za obilazak sličnih agregata
    - zadržava se povoljnost višestrukih simultanih obilazaka agregata

Iterator (12)

- Implementacija (nastavak):
  - koliko je robustan iterator?
    - nije bezbedno modifikovati agregat dok se on obilazi
      - moguće je ukloniti tekući element
        (iterator postaje “viseći”)
      - moguće je umetnuti ili ukloniti naredni element
        (problem ako je iterator već zapamtio adresu narednog)
    - robusno implementiran uzorak iteratora obezbeđuje da umetanje/uklanjanje elemenata agregata ne interferira sa obilaskom
    - trivijalno ali restriktivno sprečavanje interferencije
      - za vreme obilaska agregat se zaključava za izmene strukture
    - sofističnije sprečavanje interferencije
      - agregat treba da registruje iteratore koje je stvorio
      - pri umetanju i uklanjanju elementa agregata
        - agregat prilagođava stanje registrovanih iteratora ili
        - agregat obaveštava iteratore o promeni stanja (*Posmatrač*)

Iterator (13)

- Implementacija (nastavak):
  - Dodatne operacije iteratora za uređene/indeksirane agregate
    - `prethodniElement()` postavlja iterator na prethodni element
    - `skociNa()` postavlja iterator na objekat koji odgovara zadatom kriterijumu
  - Polimorfni iteratori na jeziku C++
    - polimorfni iteraratori imaju svoju cenu – objekat iteratora se alocira dinamički
    - nedostatak je što je klijent odgovoran za dealokaciju objekta iteratora
  - Iteratori mogu da imaju privilegovani pristup
    - iteratori mogu da se posmatraju kao proširenja agregata sa kojima su čvrsto spregnuti
    - na jeziku C++ čvrsta sprega može da se iskaže tako što su iteratori prijatelji agregata
    - ako su iteratori prijatelji, agregat ne mora da implementira operacije za efikasni pristup
    - loša strana rešenja je što pri definisanju novih iteratora mora da se menja agregat (da bi se proglasio novi prijatelj)
    - klasa `Iterator` može da uključi neke zaštićene operacije za direktan pristup agregatu
      - samo izvedene klase iz klase `Iterator` mogu da koriste date operacije te nije narušena kapsulacija agregata
      - samo klasa `Iterator` ostaje prijatelj klase `Agregat` izvedene klase iteratora ne moraju da se proglašavaju prijateljima agregata

Iterator (14)

- Implementacija (nastavak):
  - Iteratori za objektna stabla (objektne strukture uzorka *Sastav*)
    - objekti *Sastava* (u hijerarhiji stabla) se često obilaze na razne načine:
      - prefiksnim/infiksnim/postfiksnim redosledom, po širini/dubini stabla
    - nije jednostavno spolja pamtiti poziciju – ona treba da uključi putanju od korena
    - zato su spoljašnji iteratori komplikovaniji za agregate tipa sastava
    - unutrašnji iteratori implicitno čuvaju putanju na steku
      - rekurzivno se izvršava zadata operacija
    - spoljašnji iterator
      - klijent može da od tekućeg čvora traži iterator za obilazak njegove dece
  - *NullIterator*
    - degenerisani iterator pogodan za obradu graničnih uslova
    - po definiciji, njegova operacija `kraj()` uvek vraća `true`
    - pogodan je za obilazak agregata sa strukturom stabla
      - svi čvorovi osim listova vraćaju iterator za obilazak dece, a list vraća *NullIterator*
      - ovim se dobija uniformnost u obilasku strukture stabla

Iterator (15)

- Povezani uzorci:
  - *Iterator* se često primenjuje na rekurzivne strukture kao što je *Sastav*
  - *Posmatrač* može da se primeni za realizaciju robusnog *Iteratora*
  - Polimorfni iteratori se zasnivaju na *Fabričkom metodu*
    - apstraktna operacija agregata (fabrike) za kreiranje iteratora
    - konkretizuje se u potklasi konkretnog agregata
    - objekat konkretnog agregata stvara objekat konkretne potklase iteratora
    - klijent rasterećen brige o konkretnom tipu iteratora za dati agregat
  - *Iterator* može interno da koristi *Podsetnik* za čuvanje stanja iteracije