

Projektovanje softvera

Iterator



Iterator (1)

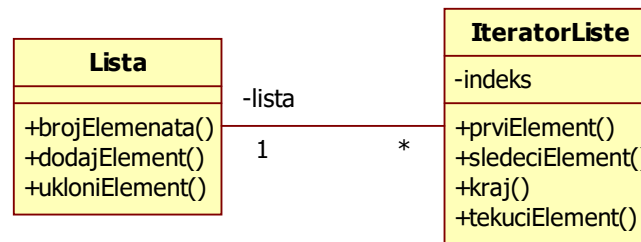
- Ime i klasifikacija:
 - Iterator (engl. *Iterator*) – objektni uzorak ponašanja
- Namena:
 - obezbeđuje pristup elementima zbirke (agregata) redom, bez eksponiranja interne strukture te zbirke
- Drugo ime:
 - Kurzor (engl. *Cursor*), za specifičnu vrstu iteratora

Iterator (2)

- Motivacija:
 - lista (agregat) treba da omogući pristup njenim elementima
 - interna struktura liste ne treba da bude eksponirana klijentima
 - može postojati više načina za obilazak liste
 - klasu liste ne treba opteretiti operacijama za razne načine obilazaka
 - ni klijenta ne treba opteretiti specifičnostima 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 interfejs za pristup elementima liste
 - objekat iteratora zna 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 (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)

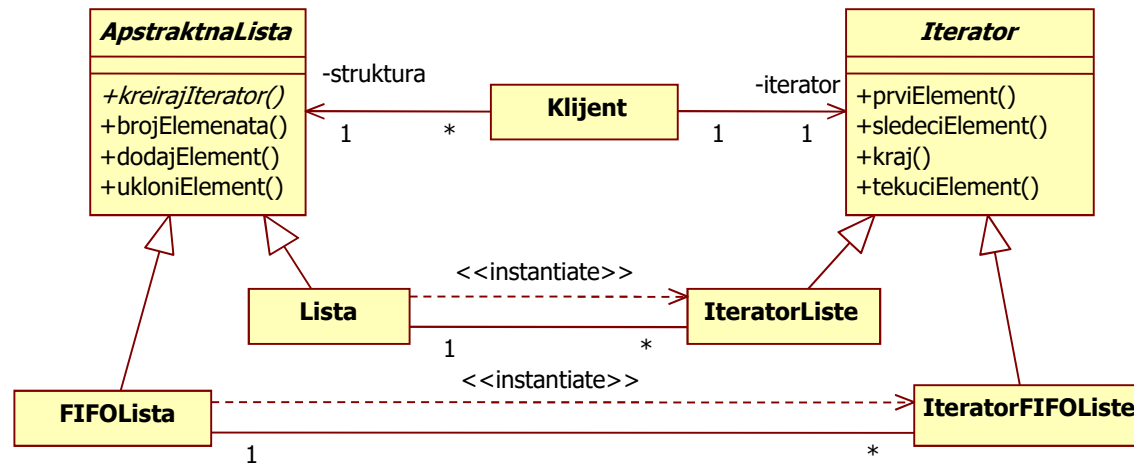
- Motivacija (nastavak):
 - razdvajanje mehanizma obilaska od objekta liste omogućava definisanje posebnih iteratora za različite politike obilaska
 - na primer: `IteratorFiltriraneListe` može pružiti pristup samo onim elementima koji zadovoljavaju uslove filtra
 - interfejs klase `Lista` nije opterećen raznim politikama obilaska
 - nedostatak:
 - klijent mora biti svesan 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
 - morao bi se menjati kod klijenta za stvaranje iteratora

Iterator (5)

- Motivacija (nastavak):
 - generalizacija koncepta iteratora: polimorfni iterator
 - primer:
 - klijent treba da radi sa klasom `Lista` ali i sa klasom `FIFOLista`
 - definiše se interfejs `ApstraktnaLista`
 - za manipulisanje proizvoljnom listom
 - definiše se interfejs `Iterator`
 - za objekte iteratora svih vrsta lista
 - potklase koje implementiraju interfejs `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

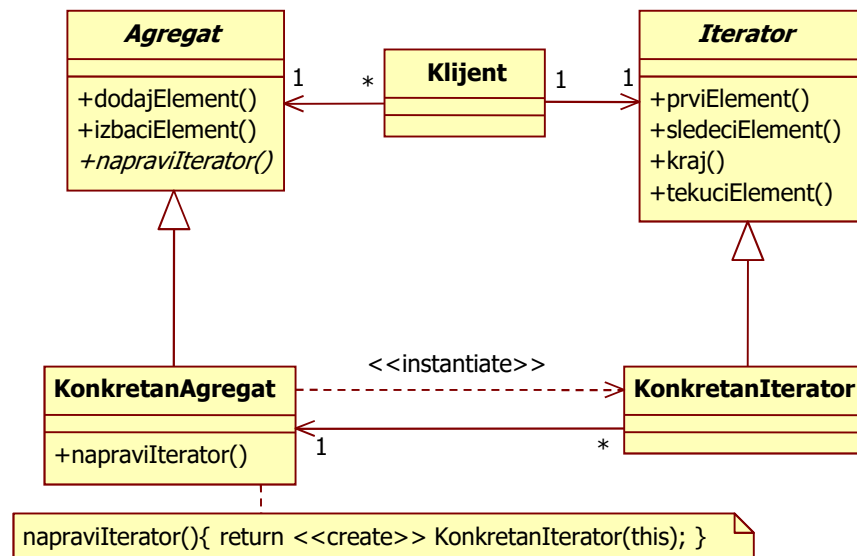
Iterator (6)

- Motivacija (nastavak):
 - da bi se klijent učinio nezavisnim od konkretne liste/iteratora:
 - definiše se operacija liste `kreirajIterator()` kojom ova treba da kreira svoj iterator
 - klijenti koriste apstraktnu operaciju i tako su nezavisni od konkretnih klasa liste
 - operacija `kreirajIterator()` je primer *Fabričkog metoda*



Iterator (7)

- Primenljivost: uзорak treba koristiti
 - da se podrže višestruki simultani obilasci agregata
 - da se obezbedi uniformni interfejs za obilazak različitih agregata
 - da se ni klijent ni agregat ne optereće politikom obilaska
- Struktura:

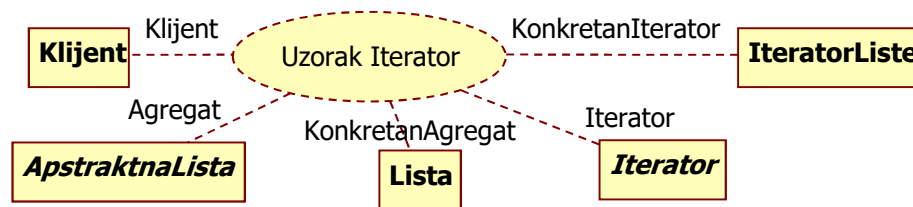


Iterator (8)

- Učesnici:
 - Agregat (klasa `ApstraktnaLista`)
 - definiše interfejs za kreiranje objekta iteratora
 - Iterator (klasa `Iterator`)
 - definiše interfejs za obilazak elemenata agregata i pristup elementu
 - KonkretanAgregat (klase `Lista`, `FIFOLista`)
 - implementira interfejs za kreiranje iteratora tako što vraća odgovarajući konkretan iterator
 - KonkretanIterator (klase `IteratorListe`, `IteratorFIFOListe`)
 - implementira interfejs `Iterator`
 - čuva informaciju o tekućoj 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 se može simultano sprovoditi nad agregatom
 - podržava varijacije u obilasku agregata:
 - lako se kreira nova klasa konkretnog iteratora za novi način obilaska

Iterator (10)

- Implementacija:
 - ko upravlja iterativnim procesom?
 - Spoljašnji (*external*) iterator – klijent kontroliše iteraciju
 - na klijentu je odgovornost za progres obilaska
 - eksplicitno zahteva od iteratora sledeći element
 - iterator o kojem je ovde bilo reči je spoljašnji iterator
 - Unutrašnji (*internal*) iterator – iterator kontroliše iteraciju
 - 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 definisati algoritam obilaska
 - agregat ima operacije za obilazak i dohvaćanje: `prviElement()`,...
 - iterator se tada koristi samo da čuva stanje iteracije
 - samo ukazuje na tekuću poziciju u agregatu
 - ovakav iterator se naziva Kurzor (*Cursor*)
 - klijent poziva operaciju `sledeciElement()` sa kurzorom kao argumentom
 - operacija `sledeciElement()` promeni stanje 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?
 - opasno je 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
 - sprečavanje interferencije
 - agregat treba da registruje iteratore koje je kreirao
 - pri umetanju i uklanjanju elementa
 - agregat prilagođava stanje registrovanih iteratora

Iterator (13)

- Implementacija (nastavak):
 - Dodatne operacije iteratora za uređene/indeksirane agregate
 - `prethodniElement()` pozicionira iterator na prethodni element
 - `skociNa()` pozicionira iterator na objekat koji odgovara specifičnim kriterijumima
 - Polimorfni iteratori u C++
 - polimorfni iteraratori imaju svoju cenu – objekat iteratora se alokira dinamički
 - nedostatak je što je klijent odgovoran za dealokaciju objekta iteratora
 - Iteratori mogu imati privilegovani pristup
 - iteratori se mogu posmatrati kao ekstenzije agregata sa kojima su čvrsto spregnuti
 - na C++ se čvrsta sprega može iskazati 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
 - date operacije koriste samo izvedene klase iz klase `Iterator`

Iterator (14)

- Implementacija (nastavak):
 - Iteratori za objekte *Sastava*
 - 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 tipa stabla (kao što je *Sastav*)
 - 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*
 - 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
 - *Iterator* može da interno koristi *Podsetnik* za čuvanje stanja iteracije