Paterni ponašanja

Predmet: Objektno orijentisana analiza i dizajn

Grupa: Tartarus
Hadžić Ajdin
Halilović Kemal
Mehmedović Faris

Paterni ponašanja

1. Strategy pattern
   Osnovna namjena Strategy paterna je da omogući da se različite implementacije istih algoritama mogu izdvojiti u različite klase, radi jednostavne implementacije one implementacije koja je potrebna u određenom trenutno. Strategy patern kreira novi interfejs koji sadrži glavnu metodu, koju će implementirati nove klase algoritama. U našem sistemu nemožemo navesti primjer ovog paterna, međutim možemo implementirati patern nad klasama Korisnik i Gost, kao funkcionalnost pretrage drugih korisnika. Morali bi definisati nove dvije klase, *pretragaZatvorenika* i *pretragaCuvara*, kao i interfajs *IPretraga*. Klase korisnik i gost bi implementirale metodu *pretrag(s:String)* iz interfejsa. Ovo bi nam omogučilo dodavanje novih sistema za pretragu korisnika tako što bi bilo dovoljno dodati novu klasu koja bi nasljedila interfejs *IPretraga*.

2. State pattern
   State pattern se korisiti radi promjene stanjna objekata, od kojeg zavisi njegovo ponašanje. Korisnik nije u mogućnosti da promjenji stanje po želji, već bi se promjena desila automatski, ako se steknu uslovi za promjenu. Primjena ovog paterna unutar sistema nije moguća, jer odstupa od generalnog uzorka po kojem bi trebala funkcionisati aplikacija.

3. Template Method pattern
   Osnovna namjena Template Methoda paterna je da omogući promjene ponašanja u jednom ili više djelova, i primjenju je se najčešće kada se trebaju izvršiti isti koraci za neki algoritam, ali je moguće izvršiti pojedinačne korake na razlišite načine. Nakon razmatranja, primjetili smo da bismo ovaj patern mogli primjeniti na klase *Upravnik*, da bi se smanjila kompleksnost metode *optimizirajKapacitete*. Zbog kompleksnosti načina na koji korisnik može korisiti tu metodu, korištenjem paterna omogućeno bi bilo pojednostavljanje implementacije rada Sistema.

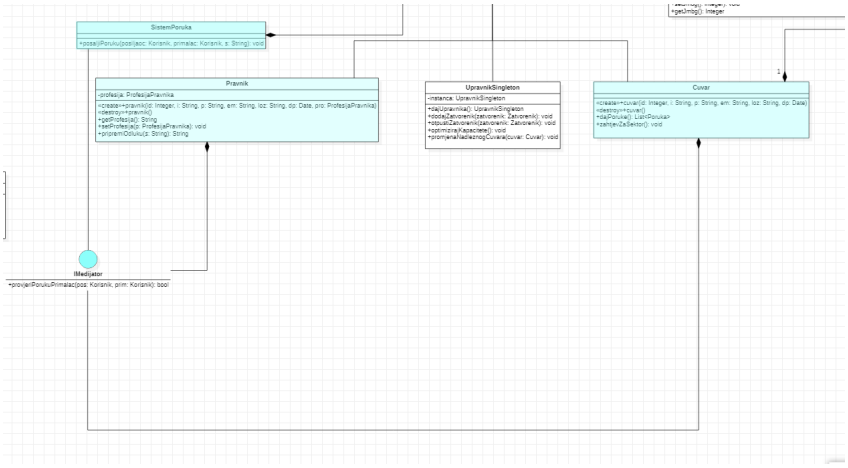
4. Observer pattern
   Observer paterna služi da bi se na jednostavanan način kreirao sistem pretplaćivanja, u kojem pretplanici dobivaju obavještenja vezana na ono što su pretplaćeni. Primjena ovog paterna unutar sistema nije moguća, jer odstupa od generalnog uzorka po kojem bi trebala funkcionisati aplikacija.

5. Iterator pattern
   Osnova namjerna Iterator paterna je da omogući prolazak kroz liste elemenata bez potrebe poznavanja načina na koji je struktura implementirana. Ovo dozvoljava da klijent na jednostavan način dolazi do traženog elementa u listi. U našem sistemu nemožemo navesti primjer

ovog paterna, međutim možemo implementirati patern nad klasama Korisnik i Gost, gdje bi paterna olakšavao pretragu zatvorenika u Sistemu po specifikacijama. Potrebo bi bilo dodati klase *IteratorPoImeIPrezime*, *IteratorPreksajima* i *IteratorPoSektorima*, koje bi kroz interfejs *IPretragaZatvorenika*, mijenjale način iteracije kroz listu zatvorenika, u zavisnosti od odabereni specifikacija.

6. Chain of Responsibility pattern
   Chain of Responsibility pattern služi radi razbijanja procesa obrade na način više objekata koji na različite načine procesiraju primljene podatke. Ovo mogućava smanjivanje kompleksnosti svakog pojedinačnog procesa, kao i jednostavniji pregled Sistema. Ovaj patern bi se mogao primjeniti na klasu *Korisnik*. Sistem za slanje poruka bi popunio potrebne podatke o primaocu i pošiljaocu uz korištenje Chain of Responsibility paterna.

7. Mediator pattern
   Mediator patern koristi se kako bi se omogučilo smanjenje broja veza između objekata. Veliki broj objekata bi bio povezan na međuobjektnom medijatorom, koji je zadužen za njihovu interakciju i komunikaciju. Primjer ovog paterna u našem sistemu je nad korisnik klasom i *SistemomPoruka*. Pošto određeni tipovi korisnika nisu u mogućnosti da šalju poruke tipovima vrstama korisnika, potrebna je kontrola. Interfejs *IMedijator* sadrži potrebne metode posredstva između korisnika, koja je metoda *provjeriPorukuPirmalac*, koja provjerava tipove korisnika koji šalju i primaju poruke. Klasa *Korisnik* implementira ovaj interfejs. Dodani su i atributi *medijator* u klase *Upravnik* i *Cuvar*, pošto su oni klase koje nisu u mogućnosti da komuniciraju izmeđui sebe.



Slika 1. Slika1. Primjer Mediator paterna primjenjen na class diagramu Tartarus aplikacije