

PATERNI PONAŠANJA

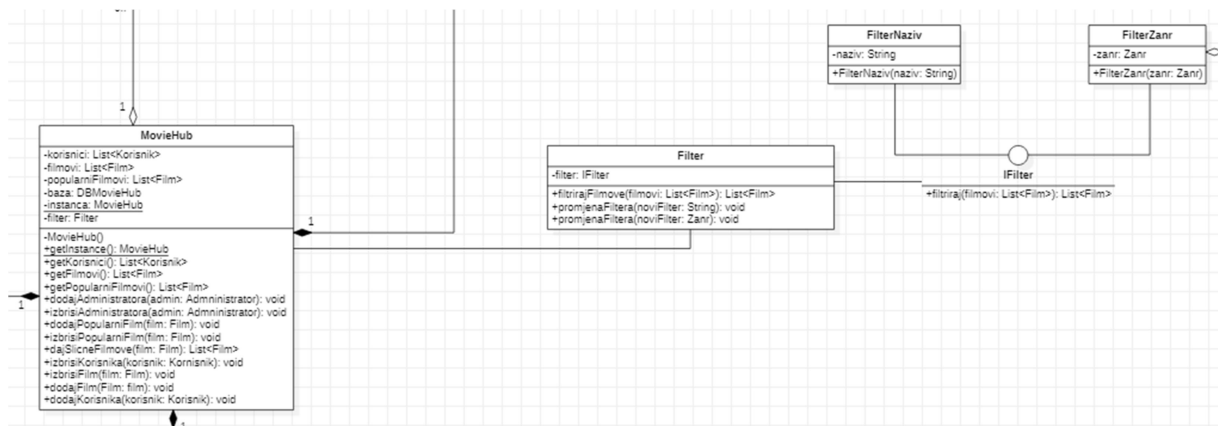
Strategy Pattern

Strategy patern izdvaja algoritam iz matične klase i uključuje ga u posebne klase. Pogodan je kada postoje različiti primjenjivi algoritmi (strategije) za neki problem. Strategy patern omogućava klijentu izbor jednog od algoritma iz familije algoritama za korištenje.

U našem sistemu Strategy pattern možemo iskoristiti kada radimo filtriranje filmova prilikom pretrage. Moguće je filtrirati filmove ili po žanru ili po nazivu (pretraga na osnovu unesene riječi).

Primjenili smo ga tako što smo dodali novu klasu Filter koja ima atribut tipa IFilter, a IFilter je zapravo interfejs sa metodom filtriraj. Ovaj interfejs realizuju dvije klase, a to su FilterNaziv i FilterZanr, koje u zavisnosti koji je filter trenutno odabran, rade odgovarajuće filtriranje filmova. Promjena filtriranja se zapravo vrši u klasi Filter, koja je zadužena da to reguliše.

Korištenjem Strategy patterna olakšali smo korisniku da bira koje filtriranje želi, tako što smo odvojili različite algoritme u zasebne klase. Na slici ispod možemo vidjeti kako sada izgleda naš dijagram klasa:



State Pattern

State Pattern je dinamička verzija Strategy patterna. Objekat mijenja način ponašanja na osnovu trenutnog stanja. Postiže se promjenom podklase unutar hijerarhije klasa.

U našem sistemu nismo iskoristili ovaj patern, međutim kada bi napravili neke promjene mogao bi se iskoristiti. To promjene bi se ogledale u tome da uvedemo neko stanje filmova, tj. popularnost koja bi se određivala na osnovu broja pregleda. Dodali bi neku klasu Popularnost koja bi bila kao Context klasa, koja bi držala u sebi attribute potrebne za određivanje popularnosti, te također atribut nekog interfejsa IPopularity koji bi nasljeđivale klase npr. SlaboPopular, SrednjePopular, JakoPopular, itd. Interfejs IPopularity bi imao neku metodu dajPopularnost(), koju bi implementirale navedene klase.

TemplateMethod Pattern

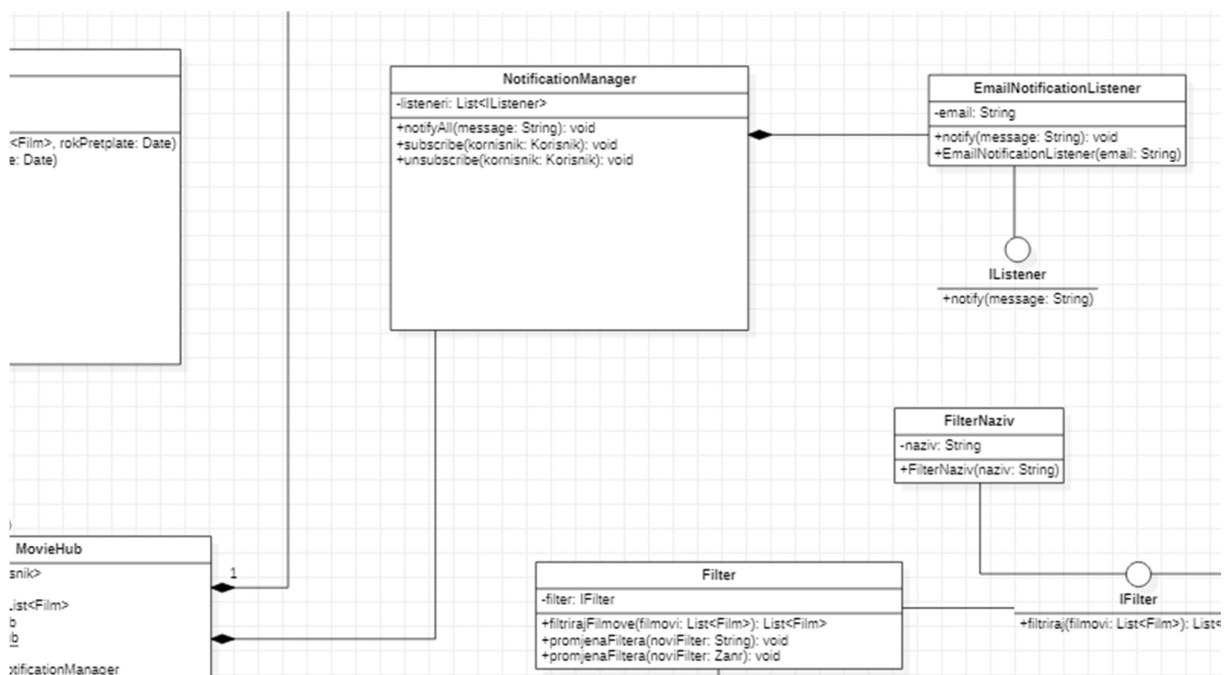
Omogućava izdvajanje određenih koraka algoritma u odvojene podklase. Struktura algoritma se ne mijenja - mali dijelovi operacija se izdvajaju i ti se dijelovi mogu implementirati različito.

Trenutno nigdje nije iskorišten ovaj patern, međutim kada bi u našem sistemu imali dvije podjele registrovanih korisnika na ObicniKorisnik i VIPKorisnik mogli bismo ga iskoristiti kako bi regulisali različite načine plaćanja pretplate, s obzirom da bi VIPKorisnik imao neke pogodnosti, kao npr. da plaća pretplatu 50% jeftinije od ObicniKorisnik.

Kada bi ga implementirali ovaj patern, uradili bi to na način da bi dodali neku klasu Clanarina, te bi ta klasa imala neku metodu TemplateMethod(). Definišemo također interfejs IClanarina koji ima metodu placanjeClanarine(). Ovaj interfejs bi implementirale gore navedene klase ObicniKorisnik i VIPKorisnik. Metoda TemplateMethod() bi pozivala odgovarajuću metodu placanjeClanarine() u ovisnosti koji je korisnik u pitanju.

Observer Pattern

Uloga *Observer* paternna je da uspostavi relaciju između objekata tako kada jedan objekat promijeni stanje drugi zainteresirani objekti se obavještavaju. U našem sistemu su ovaj patern implementirali na način da smo dodali klasu NotificationManager koja predstavlja *Subject* klasu te se unutar nje registruju (odnosno *subscribe-ju*) korisnici. Dalje, imamo interfejs IListener kojeg realiziraju konkretni listeneri. Za sada u našem sistemu potreban je samo EmailNotificationListener. Unutar klase MovieHub se nalazi instanca tipa NotificationManager, te u zavisnosti od promjene liste popularni filmovi korisnicima koji su pretplaćeni, šalje se notifikacioni mail. Pretplaćivanje korisnika je zamišljeno putem nekih od *controllera*.



Iterator Pattern

Iterator patern omogućava sekvencijalni pristup elementima kolekcije bez poznavanja kako je kolekcija strukturirana.

Ovaj patern bi mogli koristiti pri kretanju kroz liste u klasi MovieHub, međutim pošto postoji foreach petlja ovaj patern bi bio suvišan. Ovaj patern bi mogli primjeniti da smo u sistemu imali klase VipKorisnik i ObičniKorisnik izvedene iz RegistrovaniKorisnik. Te klase VipKorisnik i ObičniKorisnik bi implementirale interfejs IEnumerable. Ovaj interface bi imao metode getNext() i hasMore(). Također bi u samoj MovieHub klasi imali listu korisnika tipa IEnumerable.