**PATERNI PONAŠANJA**

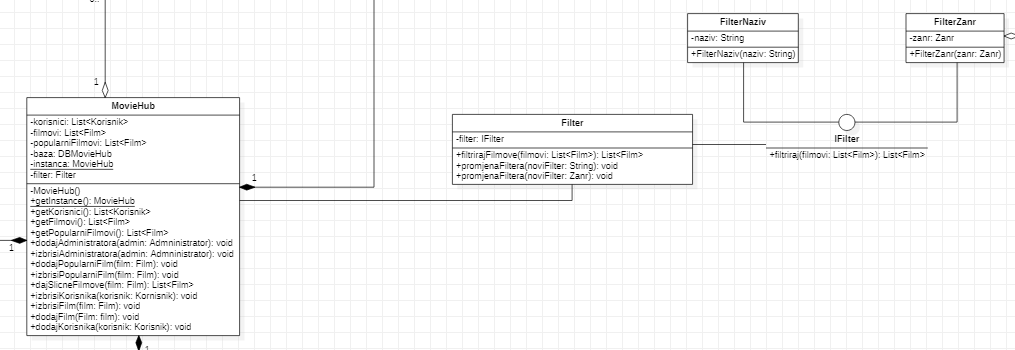
**Strategy Pattern**

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 paterna. 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 atribute 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**

**Observer Pattern**

**Iterator Pattern**