KREACIJSKI PATERNI

Singleton patern

Singleton pattern služi kako bi se neka klasa mogla instancirati samo jednom. Ovim patternom se osigurava globalni pristup jedinstvenoj instaci, te olakšava kontolu pristupa kada je neophodno da postoji samo jedan objekat određenog tipa.

U našem slučaju, vidimo da je klasa MovieHub glavna klasa sistema, te se u njoj drže svi potrebni podaci za rad sistema. Ova klasa će biti singleton klasa, odnosno samo jedna instanca ove klase može postojati. U ovoj klasi, osim podaka, imamo privati atribut instanca koji je tipa MovieHub, također jedini konstruktor koji postoji je onaj bez parametara i on je privatni, te se poziva unutar metode getInstance(). Ova metoda će napraviti novu instancu ukoliko ona već ne postoji, u suprotnom će vratiti postojeću instancu.

Prototype patern

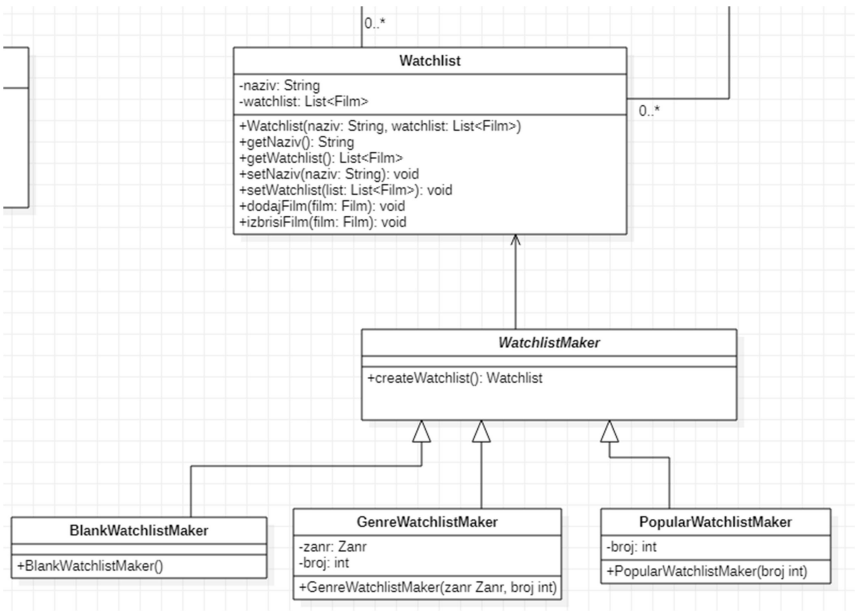
Prototype patern omogućava smanjenje kompleksnosti kreiranja novog objekta tako što se uvodi operacija kloniranja. Ovime se osigurava pojednostavljenje procesa kreiranja novih instanci, posebno kada objekti sadrže veliki broj atributa koji su za većinu instanci isti.

U našem slučaju da smo omogućili opciju kopiranja tuđe watchliste, ovaj patern bi imao smisla. U tom slučaju bi trebali napraviti interfejs IPrototip koji će imati metodu kloniraj(). Watchlista bi implementriala tu metodu, koja bi vraćala novu watchlistu.

Factory Method patern

Factory method patern služi za omogućavanje instanciranje različitih vrsta podklasa koristeći factory metodu koja odlučuje koja će se podklasa instancirati i koja programska logika izvršiti.

Mi smo ovaj patern implementirali na način da smo uveli različite načine za kreiranje watchliste, po korisnikovom izboru. To smo uradili tako što smo kreirali apstraktnu klasu WathclistMaker koja ima metodu createWatchlist() koja predstavlja factory metodu, I koja će vratiti kreiranu watchlistu. Iz ove klase su izvedene tri klase koje predstavljaju različite tipove pravljenja watchlisti: BlankWatchlistMaker, GenreWatchlistMaker i PopularWatchlistMaker. BlankWatchlistMaker će napraviti praznu watchlistu, GenreWatchlistMaker će napraviti watchlistu sa filmovima po zadanom žanru, dok će PopularWatchlistMaker napraviti watchlistu sa popularnim filmovima.



Abstract factory patern

Abstract Factory patern omogućava da se kreiraju familije povezanih objekata/produkata. Na osnovu apstraktne familije produkata kreiraju se konkretne fabrike (*factories*) produkata različitih tipova i različitih kombinacija. Dalje, moramo obratiti pažnju na to da je cilj ovog paterna uklanjanje potrebe za postojanjem ikakvih metoda za instanciranje podtipova željenih klasa – umjesto polimorfne implementacije factory metode, atributi podtipova dodjeljuju se factory klasama. Ovaj patern na prvi pogled nije primjenjiv na našem sistemu. Međutim, možemo zamisliti hipotetski slučaj u kojem bi u našem sistemu postojao objekat *Badge* koji bi se sastojao od nekih drugih objekata kao npr Color, Stars i sl, koji imaju razne podtipove. I recimo da bi taj *Badge* bio atribut klase Korisnik. te da imamo više vrsta korisnika kao vip, legend, master i sl, i da je potrebno da svaki korisnik ima drugačiji *badge*. Kako bi se riješio problem potrebe velikog broja *if-else* uslova prilikom kreiranja objekata *badge* pogodno bi bilo iskoristiti *Abstract factory* patern. Tada bi se klasa *badge* podijelila na različite *factory-e* kao naprimjer AdministatorBadgeFactory, MasterBadgeFactory koje bi u sebi već sadržale namjenjene atribute za tog korisnika. Također korisnici bi onda kao atribut imali odgovarajući *factory.*

Builder patern

Što se tiče Builder paterna, on nam služi za apstrakciju procesa konstrukcije objekta, kako bi se kao rezultat mogle dobiti različite specifikacije objekta koristeći isti proces konstrukcije. Ovaj patern nam omogućava da proizvedemo različite tipove I reprezentacije objekata koristeći isti konstrukcijski kod.

Mi nažalost u našem sistemu nismo iskoristili ovaj patern, jer trenutno nije bilo potrebe, međutim kada bi napravili neki sistem pravljenja pretplata, u kojem korisnik koji uzima pretplatu, sam bira od kojih komponenti će se njegova pretplata sastojati(npr. Korisnik pravi pretplatu za određeni broj dana, pretplatu samo na određene filmove ili filmove određenog žanra), ovaj patern bi se onda mogao iskoristiti. Da imamo ovaj sistem, onda bi bilo potrebno napraviti neku klasu PretplataMaker koja implementira interfejs IBuilder, te neku klasu Pretplata koja bi omogućavala konstrukciju objekta iz vise dijelova, a dijelove smo naveli iznad(trajanje, filmovi, žanrovi unutar pretplate).