STRUKTURALNI PATERNI

1 – Adapter patern

Adapter patern služi da omogući širu upotrebu postojećih klasa. Koristimo ga kada

želimo drugačiji interfejs već postojeće klase a ne želimo mijenjati tu klasu.

Korištenjem Adapter paterna kreirala bi se nova Adapter klasa pomoću koje bi dobili željenu funkcionalnost bez da mijenjamo Originalnu klasu I da ne ne narušimo integritet aplikacije.

Mi u našoj aplikaciji nećemo implementirati ovaj patern, ali kada bismo to željeli uraditi mogli bi npr. omogućiti registrovanim korisnicima da predlažu filmove koje bi željeli gledati, a to bismo uradili tako što bi dodali Adapter klasu I napravili interfejs koji bi omogućio odnosno sadržavao metode da bi realizovali ovu funkcionalnost.

2-Fasadni patern

Facade patern se koristi kada sistem ima više identificiranih podsistema pri čemu su apstrakcije i implementacije podsistema usko povezane. Osnovna namjena Facade paterna je da osigura više pogleda visokog nivoa na podsisteme.

Fasadni patern ćemo implementirati tako što ćemo napraviti napraviti klasu FlexFlix koja bi predstavljala fasadnu klasu koja će sadržavati druge klase kao attribute te pozivom jedne metode ove klase će biti pozvano više različitih metoda iz različitih klasa na koje korisnik nema uvid.

3-Dekorator patern

Osnovna namjena Decorator paterna je da omogući dinamičko dodavanje novih elemenata i ponašanja postojećim objektima. Objekat pri tome ne zna da je

urađena dekoracija što je veoma korisno za ponovnu upotrebu komponenti softverskog sistema.

Mi u našoj aplikaciji nećemo implementirati Dekorator patern jer nemamo potrebe za tim. Ukoliko bismo željeli da implementiramo ovaj patern mogli bismo to uraditi npr. tako što bismo omogućili korisnicima da osvajaju nagradne poene te na osnovu određenog broja osvojenih poena dobiva besplatnu ulaznicu za koju bi dobio obavijest na početnoj stranici svog naloga.

4-Bridge patern

Osnovna namjena Bridge paterna je da omogući odvajanje apstrakcije i implementacije neke klase tako da ta klasa može posjedovati više različitih apstrakcija i više različitih implementacija za pojedine apstrakcije. Bridge patern pogodan je kada se implementira nova verzija softvera a postojeća mora ostati u funkciji.

U našoj aplikaciji nemamo mogućnost za implementaciju ovog paterna.

5-Proxy patern

Namjena Proxy paterna je da omogući pristup i kontrolu pristupa stvarnim objektima. Proxy je obično mali javni surogat objekat koji predstavlja kompleksni objekat čija aktivizacija se postiže na osnovu postavljenih pravila. Proxy patern rješava probleme kada se objekt ne može instancirati direktno (npr. zbog restrikcije pristupa).

U našoj aplikaciji ovaj patern ćemo iskoristiti tako što ćemo zabraniti zlonamjerne upotrebe. Recimo ukoliko neki korisnik koji nije registrovan kao administrator pokuša da koristi privilegije koje ima administrator, ta akcija će biti zabranjena i nalog tog korisnika će biti pohranjen u bazu podataka, te će administrator biti obavješten o tome.

6-Composite pattern

Osnovna namjena Composite paterna je da omogući formiranje strukture stabla pomoću klasa, u kojoj se individualni objekti (listovi stabla) i kompozicije individualnih objekata (korijeni stabla) jednako tretiraju. Ovaj patern nećemo

implementirati u našoj aplikaciji, kada bi se odlučili da implementiramo ovaj patern uradili bi to na način tako što bi dodali jos jednu vrstu korisnika(VIP) te omogućili tim korisnicima kupovinu godišnje karte.

7-Flyweight pattern

Ovaj patern koristimo ukoliko želimo da onemogućimo stvaranje velikog broja objekata koji u suštini predstavljaju jedan objekat. Ovaj patern nećemo implementirati u našoj aplikaciji ali kada bismo to željeli uraditi mogli bi na sljedeći način: Ukoliko se jedan korisnik koji je Gost prijavljuje više puta, tada će biti instanciran samo jedan objekat ovog tipa, a ne svaki put kada se prijavi.