Strukturalni paterni

1. Adapter patern

Osnovna namjena Adapter paterna je da omogući širu upotrebu već postojećih klasa. Kada je potreban drugačiji interfejs već postojeće klase, a ne želimo mijenjati postojeću klasu koristimo ovaj patern. Adapter patern kreira novu adapter klasu koja služi kao posrednik između originalne klase i željenog interfejsa.

U klasi PoslovniKorisnik imamo listu događaja koje je on objavio. Možemo napraviti adapter koji bi umjesto liste događaja vraćao uređen raspored događaja po odabranoj kategoriji kojoj događaj pripada npr. svaka kolona bi predstavljala jednu kategoriju i bila bi implementirana kao matrica. Ovo bi se moglo uraditi na način da adapter naslijedimo iz klase PoslovniKorisnik i da implementiramo interfejs IVrsta.

2. Facade patern

Ovaj patern koristi se kada sistem ima više identificiranih podsistema pri čemu su apstrakcije i implementacije podsistema usko povezane. Osnovna funkcija mu je da osigura više pogleda visokog nivoa na podsisteme. Može se više fasada postaviti oko postojećeg skupa podsistema i na taj način formirati više prilagođenih pogleda na sistem.

U našem sistemu nema pretjerane potrebe za dodavanjem ovog paterna, ali bi se ovaj patern mogao iskoristiti sa interfejsima jer smo ih napravili da budu pomoćni interfejsi našim klasama u sistemu, pa ih možemo smjestiti u jednu class Fasadu.

3. **Decorator 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.

U našem sistemu kada poslovni korisnik objavljuje događaj ima mogućnost dodavanja fotografije za događaj. Da bi se korisniku omogućila manipulacija slikom koju doda u sistem se može dodati nova klasa Slika i u tu klasu bi se mogle odvojiti aktivnosti vezane za sliku. Također u sistem se može dodati i interfejs Islika koja identificira klase objekata koje želimo dekorisati. Interfejs između ostalog treba sadržavati metodu edituj(), dok klasa Slika treba da implementira interfes ISlika. Svaki konkretan dekorator treba da sadrži kao atribut ISlika potom treba da implementira interfejs ISlika te u slučaju potrebe dodati nove metode.

4. Bridge patern

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. Pogodan je kada se implementira nova verzija softvera a postojeća mora ostati u funkciji. Moguće je implementirati i sistem za razmjenu poruka primjenom Bridge sistema.

Iako u našem sistemu nije neophodno da se unosi datum rođenja prilikom registracije, kada bi to bilo moguće korisniku bi se mogao obračunavati određeni popust ukoliko kupovinu karte obavlja na svoj rođendan. U našem sistemu u klasi Karta postoji metoda dajCijenu() i nju bismo mogli proširiti interfejsom Bridge koja bi u sebi imala istoimenu metodu, te bi na osnovu datuma rođenja pozivala dvije različite implementacije od kojih bi jedna vraćala normalno obračunatu cijenu, a druga bi vraćala sniženu cijenu.

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.

Ovaj patern je pogodan ukoliko želimo da poboljšamo sigurnost našeg sistema. U našem sistemu ovaj patern se može koristiti da bi ograničili prava pristupa određenim funkcionalnostima. Nakon logina bi se ustanovilo da li su podaci ispravni, te koje je vrste korisnik koji je izvršio prijavu, jer svi korisnici nemaju iste mogućnosti na sistemu. Ovo ćemo realizovati dodavanjem interfejsa IProxyPrijava i klasu ProxyPrijava koja ima metodu preko koje se vrši provjera podataka i identifikacija korisnika na osnovu unesenih podataka.

6. Composite patern

Osnovna namjena Composite paterna je da omogući formiranje strukture stabla pomoću klasa u kojoj se individualni objekti i kompozicije individualnih objekata jednako tretiraju.

Ukoliko želimo dopustiti korisniku pregled svih događaja u sistemu podijeljenih na osnovu vrste kojoj pripadaju(kultura, sport, muzika i kino) trebali bi napraviti klasu VrsteDogađaja, koja bi imala atribut događaji:List<Događaj> i metodu sviDogađaji:List<Događaj> (npr. Kultura : Predstava Kralj Lear , Kino : Top Gun, Muzika : Koncert Nine Badrić.....), koju će korisnik moći pozvati i koja će vratiti sve događaje i sortirati ih po vrsti događaja. Kreirali bi i interfejs IDogađaji u koji bi implementirale metodu sortirajDogađaje za sortiranje po vrsti događaja. Klasa VrsteDogađaja bi naslijedila interfejs kako bi se napravila hijerarhija objekata. Klasa VrsteDogađaja bi sadržavala atribut type koji bi čuvao informaciju o vrsti događaja i koji bi koristila pri sortiranju.

7. Flyweight patern

U situacijama kada je potrebno da se omogući razlikovanje dijela klase koji je uvijek isti za sve određene objekte te klase od dijela klase koji nije uvijek isti za sve određene objekte te klase koristi se Flyweight patern koji omogućava da više različitih objekata dijele glavno stanje, a imaju različito sporedno stanje.

Ovaj patern bi se mogao iskoristiti sa slikama u Događajima. Ukoliko korisnik ne doda fotografiju za događaj prilikom njegovog objavljivanja, fotografija će imati neku deafultnu vrijednost u zavisnosti od odabrane kategorije kojoj događaj pripada.