STRUKTURALNI PATERNI

Adapter patern [implementacija]

U slučaju da nam je potreban drugačiji interface već postojeće klase možemo iskoristiti adapter patern za kreiranje adapter klase kao posrednika između originalne klase i željenog interface-a. Ovim se sprječava izmjena postojeće klase.

U našoj aplikaciji adapter patern se može implementirati za prilagođavanje podataka o zauzetim sjedištima iz baze s ciljem tabelarnog prikaza na korisničkom interface-u. Ovaj adapter bi formatirao podatke na način da budu prilagođeni za tabelarni prikaz kroz html.

Facade patern [implementacija]

Ovaj patern omogućava stapanje više složenijih podsistema u jedan kako bi interakcija korisnika sa sistemom bila jednostavnija i intuitivnija. Istovremeno patern osigurava da korisniku sistema nisu dostupne informacije o samoj implementaciji sistema čime se ispunjava jedan od osnovnih principa objektno orijentisanog programiranja.

Kroz našu aplikaciju ovo smo implementirali na način da su funkcionalnosti rezervacije i kupovine karata kao i odabira željenih sjedišta grupisane u jedinstven korisnički interface, iako su interno implementirani kao odvojeni sistemi.

Decorater patern

Namjena ovog paterna je da omogući dinamičko dodavanje novih elemenata i funkcionalnosti u postojećim objektima pri čemu objekat ne zna da je urađena dekoracija.

Ovaj patern smo iskoristili tako što korisnik ima mogućnost korištenja popusta prilikom plaćanja karte nakon ostvarivanja određenog broja odgledanih predstava. Ovim se proširuje funkcionalnost i usluga sistema za rezervaciju i kupovinu karata.

U slučaju dodavanja posebne vrste karata (npr. VIP) moguće je iskoristiti ovaj patern na sličan način kao i u slučaju popusta.

Bridge patern

Ovaj patern omogućava klasi posjedovanje više različitih apstrakcija i implementaciju za svaku apstrakciju kombinujući dvije ili više ortogonalnih hijerarhija klasa. Koristeći ovaj patern zadovoljava se i open-closed princip.

Ukoliko bismo se odlučili na proširenje načina online plaćanja (npr. dodavanja opcije paypal pored opcije kartičnog plaćanja) ove dvije opcije bismo mogli objediniti u okviru jedne klase

online plaćanja (apstrakcija). Zatim bi bilo potrebno kreirati interface koji bi implementirao metode za plaćanje.

Proxy patern

Proxy patern omogućava klijentu da pristupa objektu putem posrednika a ne direktno. Ovaj posrednik može obavljati dodatne zadatke i vršiti kontrolu pristupa stvarnim objektima. Ovim se osigurava open-closed princip.

Budući da se u našoj bazi podataka čuvaju informacije o korisnicima sistema moguće je iskoristiti proxy patern za autentifikaciju pri pokušaju pristupa stvarnom objektu korisničkog računa. Ovim se osigurava da samo zaposlenici imaju pristup svim korisničkim podacima.

Composite patern

Ovaj patern omogućava formiranje strukture drveta pomoću klasa tako da se jednako tretiraju individualni objekti kao i njihova kompozicija.

U našoj aplikaciji ovaj patern bi se implementirao uvođenjem interface-a kojeg implementiraju predstava i repertoar koji sadrži kolekciju predstava. Ovaj interface bi implementirao ponašanje zajedničko za pojedinačnu komponentu i njihovu kompoziciju kao što su prikaz, izmjena predstave i sl.

Flyweight patern

Ovaj patern omogućava brži pristup objektima na zahtjev. Kako objekti jedne klase mogu vrlo često posjedovati iste vrijednosti atributa ovo svojstvo možemo iskoristiti pri bržem kreiranju novih objekata, pristupu postojećim a posljedično i smanjenom memorijskom zauzeću.

Pri implementaciji naše aplikacije patern se može zadovoljiti prilikom prikazivanja pojedinačnih predstava korisniku na način da bi se uvijek koristito isti layout u koji se ubacuju konkretni podaci za predstavu. Također, ovaj patern bi se mogao primijeniti i prilikom kreiranja novog korisnika na način da bi defaultna uloga korisnika bila korisnik sistema i inicijalni broj kupljenih karata bi bio 0. Potom bi se postavljali podaci za individualnog korisnika.