**PATTERNI PONAŠANJA ZA ULAZNICE.COM**

1. **STRATEGY PATTERN**

Uloga Strategy patern-a je da izdvaja algoritam iz matične klase i uključuje ga u posebne klase. Koristi se kada postoje različite strategije koje možemo primijeniti za rješavanje nekog problema. On omogućava korisniku izbor jednog algoritma iz familije algoritama za korištenje. Ovaj patern u našem sistemu možemo iskoristiti kod filtriranja manifestacija. Naime, korisnik kada otvori formu za pretragu dostupnih manifestacija, može vršiti filtriranje po vrsti, cijeni ulaznice i udaljenosti. U našem slučaju, po vrsti filtiramo manifestacije na kulturne, sportske, sajmove itd.

1. **STATE PATTERN**

State patern omogućava objektu da promijeni svoje ponašanje ili stanje. Nakon promjene stanja, objekat se počinje ponašati kao da je promijenio klasu. Objekat mijenja način ponašanja na osnovu trenutnog stanja. State patern predstavlja dinamičku verziju Strategy patern-a. Postiže se promjenom podklase unutar hijerarhije klasa. Podržava open-closed princip. State patern bismo mogli iskoristiti u našem sistemu prilikom korisnikovog biranja Manifestacije. Klasa Manifestacija će imati različito ponašanje u zavisnosti od tipa manifestacije. Npr, ukoliko je tip manifestacije sajam ili izložba onda nam uopće ne treba funkcionalnost biranja mjesta na tribini. Dok kod kino-predstave ili neke sportske manifestacije imamo mogućnost biranja tribine i mjesta kojeg korisnik želi rezervisati.

1. **TEMPLATE METHOD PATTERN**

Svrha Template method patern-a je da omogući izdvajanje određenih koraka algoritma u odvojene podklase. Samim tim, struktura algoritma se ne mijenja, mali dijelovi operacija se izdvajaju i ti se dijelovi mogu implementirati različito. Ovaj pattern se sastoji od klase Algorithm koja uključuje metodu koja izdvaja dijelove svojih operacija u druge klase. Tu se nalazi interfejs IPrimitive koji definira operacije koja pomenuta metoda izdvaja u druge klase te AnyClass koja implementira interfejs IPrimitive. Template Method patern bismo mogli iskoristiti u našem sistemu kao nadgradnju klase Placanje. Tj. dodajemo u nju metodu TemplateMethod(). Ovu klasu povezujemo je sa interfejsom Placanje koji ima metode platiOsnovicu() i platiSPopustom(). Ovaj interfejs implementiraju klase Korisnik i VipKorisnik (Nova klasa). TemplateMethod() u svom dijelu poziva ove funkcije i u zavisnosti od tipa korisnika plati će se drugačija cijena. TemplateMethod() bi se pozivala u klasi Korisnik u kojoj bismo dodali metodu Placanje(). Dakle, razlika između ova dva korisnika je što bi VIP korisnik skupljao određene poene i ostvarivao određene pogodnosti i popuste.

1. **OBSERVER PATTERN**

Uloga Observer patterna je da uspostavi relaciju između objekata. Kada jedan objekat promijeni stanje, drugi zainteresirani objekti se obavještavaju. U našem sistemu Observer pattern je iskorišten u slučaju kada imamo situaciju da je vise korisnika zainteresovano za isto mjesto na tribini. Korisnici koji nisu u mogućnosti rezervisati i kupiti to mjesto dobit će obavještenje kada ono ponovo postane dostupno za kupovinu ili rezervaciju.

1. **ITERATOR PATTERN**

Iterator pattern omogućava sekvencijalni pristup elementima kolekcije bez poznavanja kako je ta kolekcija struktuirana. Ovaj pattern bismo mogli iskoristiti ako umjesto listi kao atribut, imamo mapu korisnika i njihovih kupljenih ulaznica, te ako bismo željeli naći korisnike koji nemaju nadolazećih manifestacija za koje su kupili ulaznice u tom trenutku. Definišemo interfejs IEnumerable sa metodom getEnumerator() kojeg implementira klasa Kolekcija koja pored metode getEnumerator() moze imati i druge metode koje pružaju vrijednosti kolekcije u nekom drugom redoslijedu ili obliku. U klasu Korisnik dodajemo atribut tipa Map i povezujemo klasu sa interfejsom IEnumerable.

1. **CHAIN OF RESPONSIBILITY PATTERN**

Chain of responsibility patern predstavlja listu objekata. Ukoliko objekat ne može da odgovori na zahtjev, prosljeđuje ga narednom u nizu. U našem sistemu, ovaj patern bi mogli iskoristiti u situaciji kada korisnik želi izvršiti promjenu mjesta na tribini, odnosno zauzeti neko atraktivnije mjesto koje bi se dodatno naplaćivalo. Napravili bi jednu klasu PromjenaNarudzbe sa metodom kreiraj() koja bi prvo pozvala klasu ZahjtevZaPromjenu koja bi kreirala jedan zahtjev za promjenu mjesta, a onda bi se nakon odobravanja tog zahtjeva od strane Administratora sistema, kreirala nova, finalna, instanca klase Karta(Ulaznica).

1. **MEDIATOR PATTERN**

Ovaj vrlo korisni pattern se koristi kada u sistemu postoji mnogo povezanih klasa koje zavise jedna od druge, Mediator ustvari definiše objekat koji enkapsuliše način na koji upravo ovi povezani objekti komuniciraju. U našem sistemu, kako je već naglašeno ranije, nema kompleksnih veza ni mnogo povezanih klasa tako da ovaj pattern trenutno nije moguće iskoristiti. Ukoliko bi se nekada sistem zakomplikovao mogli bi ga iskoristiti, npr. kada bi dodali na profil korisnika neke dodatne informacije koje zavise jedna od druge (kada bi se npr mogao dodati opis, spol, ...) tada bi mogli dodati neku posrednu klasu koja bi za odgovarajuće poruke prosljeđivala odgovarajuće manifestacije kao preporuku. Npr za žene i djevojke sajmovi knjiga i modne revije. Za muškarce bi to bile sportske manifestacije i sajmovi automobila...