**PATTERNI PONAŠANJA**

1. **Observer (Promatrač)** je obrazac ponašanja koji omogućava uspostavljanje veze između objekata u kojem promjene u jednom objektu obavještavaju i automatski ažuriraju druge objekte koji su zainteresovani za te promjene. U kontekstu rezervacije koncerata, Observer patern se može primijeniti kako bi se muzičkim umjetnicima omogućilo da budu obaviješteni o novim događajima ili koncertima na koje mogu nastupiti. Uobičajeni scenario je da postoji subjekt ,koji je odgovoran za organizaciju događaja i koncerata. Muzički umjetnici, kao promatrači (observers), žele biti obaviješteni o novim događajima kako bi mogli donijeti odluku o eventualnom nastupu na tim događajima. Kada subjekt objavi novi događaj, on obavještava sve promatrače koji su pretplaćeni na te informacije. Muzički umjetnici koji su promatrači primaju obavještenje i mogu izvršiti odgovarajuće akcije, kao što su pregledavanje detalja događaja, provjera dostupnosti, donošenje odluka o nastupu i slično. Prednosti upotrebe Observer obrasca u ovom kontekstu su: Rasterećenje subjekta od direktnog pozivanja metoda muzičkih umjetnika. Subjekt samo obavještava da je došlo do promjene, a promatrači sami odlučuju kako će reagovati na tu promjenu. Fleksibilnost i lako proširivost. Mogu se dodavati i uklanjati novi promatrači bez utjecaja na subjekt ili druge promatrače. Smanjenje ovisnosti između subjekta i promatrača, što rezultira povećanom modularnošću i održavanjem. U konkretnom primjeru rezervacije koncerata, Observer patern omogućava uspostavljanje komunikacije između organizatora događaja (subjekta) i muzičkih umjetnika (promatrača), omogućavajući im da budu obaviješteni o novim događajima i donose odluke o nastupu na temelju tih informacija.
2. **Strategy (Strategija)** može biti koristan za definisanje različitih strategija koje koriste različiti akteri u procesu rezervacije: osoba koja sluša koncerte, osoba koja iznajmljuje dvorane i osoba koja rezerviše dvorane kako bi mogla nastupiti na koncertu. Pattern Ponasanja Strategy omogućava da se različiti algoritmi ili strategije grupišu u odvojene klase i omoguće njihova zamjenjivost. Svaka strategija predstavlja alternativni način izvršavanja određene funkcionalnosti. U našem slučaju, strategije mogu biti usmjerene na različite aspekte rezervacije koncerata, kao što su cijene karata, način plaćanja, pravila otkazivanja, itd. Svaka strategija je implementirana kao posebna klasa koja implementira zajednički interface koji definiše metode i operacije relevantne za tu strategiju. Na primjer, ako imamo interface "RezervacijskaStrategija" koje definiše metode poput "izracunajCijenu()" za strategiju cijena karata, "odaberiNacinPlacanja()" za strategiju odabira načina plaćanja.
3. **Iterator (Iterator)** patern se može koristiti kako bi se omogućio prolazak kroz listu dostupnih događaja ili koncerata i izvršavanje operacija nad njima. Iterator patern odvaja algoritam prolaska kroz kolekciju od samih elemenata te kolekcije, omogućavajući da se kolekcija prolazi bez otkrivanja njene unutrašnje strukture. U našem projektu Iterator patern može se primijeniti na različite načine:

* Prolazak kroz listu dostupnih događaja: Iterator omogućava da korisnici (osobe koje slušaju koncerte) prolaze kroz listu svih dostupnih događaja ili koncerata. Korisnik može koristiti iterator kako bi prikazao sve dostupne događaje, pregledao njihove detalje, filtrirao po datumu ili drugim kriterijima, ili vršio pretragu po određenim ključnim riječima.
* Iteriranje kroz rezultate pretrage: Korisnici mogu izvršiti pretragu događaja koristeći određene parametre kao što lokacija ili datum. Iterator omogućava iteriranje kroz rezultate pretrage, prikazivanje tih rezultata korisniku i obavljanje operacija nad njima, poput rezervacije karata ili pregleda detalja o događaju.
* Prolazak kroz listu rezervisanih događaja: Osobe koje su rezervisale karte za određeni događaj mogu koristiti iterator kako bi prolazile kroz listu svojih rezervacija, provjeravale detalje rezervacije, izvršavale otkazivanje ili modifikaciju rezervacija ili vršile plaćanje.

Implementacija Iterator paterna uključuje definisanje iteratora koji implementira zajednički interface s metodama poput "hasNext()" (provjera da li ima još elemenata), "next()" (prelaženje na slijeće događaje), i opcionalno "remove()" (uklanjanje trenutnog elementa).

Ova implementacija omogućava da se kolekcija pretraga događaja ili rezervacija koncerata može prolaziti kroz standardizirani interfejs iteratora, bez obzira na to kako je unutrašnje organizirana.

1. **Memento (Memento)** patern se može koristiti za snimanje trenutnog stanja objekta (npr. rezervacije ili korisničkih podataka) i vraćanje tog stanja u budućnosti. Jedan od načina da se implementira Memento patern je kroz definisanje sljedećeg interfejsa i metoda: Interface: IMemento saveState(): Metoda koja će snimiti trenutno stanje objekta i vratiti Memento objekat koji predstavlja to stanje. restoreState(memento: Memento): Metoda koja će preuzeti snimljeno stanje objekta iz Memento objekta i vratiti objekat u to stanje.
2. **Chain of responsibility** patern U kontekstu rezervacije koncerata, Chain of Responsibility (Lanac odgovornosti) patern se može koristiti za organiziranje različitih koraka ili handlera koji obrađuju zahtjeve ili akcije u procesu rezervacije. Svaki handler ima odgovornost da obradi zahtjev, a ukoliko ne može da ga obradi, proslijeđuje ga sledećem handleru u lancu. Primjer upotrebe ovog patterna u našem se zasniva na validacija unosa podataka. Možemo implementirati različite handlere koji provjeravaju validnost unesenih podataka prilikom rezervacije. Na primjer, jedan handler se može baviti provjerom ispravnosti formata broja telefona, drugi handler provjerom dostupnosti karata za odabrani događaj, dok treći handler provjerava ispravnost unesenih ličnih podataka korisnika.
3. **Mediator (Posrednik)** patern može se koristiti za olakšavanje komunikacije između različitih komponenti sistema koje su uključene u proces rezervacije. Posrednik ima odgovornost za upravljanje interakcijama između tih komponenti, umjesto da se one direktno međusobno povezuju.Ako imamo interface iMediator komponenta za prikaz dostupnih koncerata bi bile: registerComponent(component: IComponent): Registracija ove komponente kod posrednika kako bi posrednik znao o njenom postojanju. sendNotification(sender: IComponent, notification: string): Slanje obavijesti od komponente za prikaz dostupnih koncerata ka ostalim komponentama, na primjer, obavijest o promjeni dostupnosti koncerta ili novom dodanom koncertu.
4. **Template Method (Template metoda)** patern ponasanja omogućava definisanje zajedničke strukture algoritma u apstraktnoj klasi, sa specifičnim koracima koji se implementiraju u podklasama. Evo opisa Template Method paterna za vaš projekat rezervacije koncerata: Definisanje apstraktne klase: Prvo, definišete apstraktnu klasu koja sadrži zajedničku strukturu algoritma rezervacije koncerata. Ova klasa će implementirati interface IConcertBooking i sadržati Template Method BookConcertTicket(), koji definiše korake rezervacije. Odabir koncerta: U apstraktnoj klasi, definišete apstraktnu metodu ChooseConcert(), koja će biti implementirana u podklasama. Ova metoda se koristi za odabir koncerta koji se želi rezervisati. Odabir sjedišta: Također u apstraktnoj klasi, definišete apstraktnu metodu SelectSeats(), koja će biti implementirana u podklasama. Ova metoda se koristi za odabir sjedišta na koncertu. Procesiranje plaćanja: U apstraktnoj klasi, definišete apstraktnu metodu ProcessPayment(), koja će biti implementirana u podklasama. Ova metoda se koristi za procesiranje plaćanja za rezervaciju. Slanje potvrde: Također u apstraktnoj klasi, definišete virtualnu metodu SendConfirmation(). Ova metoda ima defaultnu implementaciju, ali se može preklopiti u podklasama ako je potrebno specifično ponašanje.