Univerzitet u Sarajevu
Elektrotehnički fakultet Sarajevo

Patterni ponašanja

OOAD 2020-2021

Naziv grupe: Hilbert's Grand Hotel

Članovi: Nedim Badžak
Harun Alagić
Emil Fejzagić

1. Strategy pattern (Odabran)

Strategy pattern izdvaja algoritam iz matične klase i uključuje ga u posebne klase. Pogodan je kada postoje različiti primjenjivi algoritmi (strategije) za neki problem. Strategy pattern omogućava klijentu izbor jednog od algoritma iz familije algoritama za korištenje. Podržava open-closed princip.

Ovaj pattern bi mogli iskoristiti pri sortiranju soba. Uveli bismo u klasu “Soba” metodu Sortiraj() i napravili interface “ISortiranje” koji bi bio “IStrategy” interface. Iz njega bismo naslijedili različite vrste sortiranja, kao i imali mogućnost da kasnije, po želji i potrebi, samo napravimo novu klasu koja će naslijediti “ISortiranje” interface bez promjene postojećih klasa.

2. State pattern

State pattern je dinamička verzija Strategy patterna. Objekat mijenja način ponašanja na osnovu trenutnog stanja. Postiže se promjenom podklase unutar hijerarhije klasa.

Ovaj pattern bismo mogli iskoristiti za prikaz “Smještaja” koji bi suštinski imao metode prikaza “Apartmana” i “Sobe”. Umjesto toga, napravit ćemo prikaziSmjestaj(tip: ITip) metodu koja bi ITip interface koristila da zaključi o kojem se tipu smještaja radi i u zavisnosti od toga prikazala pravilan pogled, ApartmanPogled ili SobaPogled.

3. Template method pattern

Omogućava izdvajanje određenih koraka algoritma u odvojene podklase. Struktura algoritma se ne mijenja - mali dijelovi operacija se izdvajaju i ti se dijelovi mogu implementirati različito.

Ovaj pattern bismo mogli iskoristiti kada bismo administratoru (Šefu) željeli omogućiti generisanje različitih kupona. Trebao bi nam interface koji bi imao metodu generisi. Također bi nam trebale različite klase za generisanje koje bi bile naše AnyClass i koje bi implementirale taj interface. Ovo bi pomoglo administratoru da ima jednostavniji i pregledniji način za generisanje kupona gostima.

4. Observer pattern

Uloga Observer patterna je da uspostavi relaciju između objekata tako kada jedan objekat promijeni stanje drugi zainteresirani objekti se obavještavaju. Ovaj pattern bismo mogli primijeniti ako bismo željeli da korisnici budu obaviješteni kada se njihova soba očisti.

U našem slučaju, ovaj pattern bi bio vezan za promjenu stanja sobe i obavještavanje korisnika o tim promjenama. Subject bi bilo Osoblje koje mijenja status sobe, IObserver obavijest putem e-maila ili telefonske notifikacije, Observer bi bio sami Korisnik koji odsjeda u toj sobi, Update bi bio prijem tog obavještenja putem e-maila ili notifikacije, Notify bi u suštini bila neka vrsta e-mail poruke koju korisnici dobiju kada se prijave za određenu sobu i u kojoj bi bio link za potvrdu da žele primati obavještenja na mail o budućim promjenama ili eventualni link za odbijanje primanja takvih obavještenja, te finalno State bi predstavljalo samo novo stanje.

5. Iterator pattern

Iterator pattern omogućava sekvencijalni pristup elementima kolekcije bez poznavanja kako je kolekcija strukturirana. Njega vrlo lako možemo iskoristiti u našem sistemu budući da smo na mnogo mjesta predvidjeli upotrebu liste kao atributa klase.

Pošto će se sobe koristiti često, možemo iskoristiti listu objekata Soba u klasi “Recepcioner” za iterator pattern. S obzirom da smo sve liste pretvorili u Stringove sa id-evima, ovaj pattern nećemo koristiti, ali ćemo demonstrirati mogući slučaj upotrebe. SobeIterator će imati kao privatni atribut listu odbranih soba. Kroz tu listu će se iterirati. Bit će nam potreban i interface IEnumerable koji će lista implementirati, te koji ima metodu getEnumeration().

6. Chain of responsibility pattern

Chain of responsibility pattern namijenjen je kako bi se jedan kompleksni proces obrade razdvojio na način da više objekata na različite načine procesiraju primljene podatke.

Mi bismo ovaj pattern mogli iskoristiti za potrebe potvrde rezervacije s obzirom da svaku rezervaciju podnesenu od strane “Korisnika” treba potvrditi “Recepcioner”. Potrebna nam je klasa “ZahtjevPotvrda”, interface IHandler i klase koje nasljeđuju “ZahtjevPotvrda”.

7. Mediator pattern (odabran)

Mediator pattern enkapsulira protokol za komunikaciju među objektima dozvoljavajući da objekti komuniciraju bez međusobnog poznavanje interne strukture objekta.

Ovaj pattern bismo mogli iskoristiti kod recenzija. Naš sistem zahtijeva da recenziju može ostaviti samo neko ko je prethodno odsjedao u toj sobi. Klasa “Decorator” bi imala atribut tipa interface-a IOcjena kojeg od ranije imamo iz Decorator patterna, koji bi sada sadržavao metodu sa funkcionalnostima provjere ranijeg odsjedanja tog Korisnika.