Primjena dizajn paterna

Strukturalni paterni:

Na dijagramu su iskorištena 3 strukturalna paterna, i to Adapter patern, Bridge patern i Proxy patern.

Adapter patern je iskorišten za prilagođavanje interfejsa baze podataka klasama koje je koriste. Time je također riješen problem što nemaju sve klase ista prava korištenja baze podataka. Dakle klase BazaAdministrator, BazaKorisnik i BazaRecepcioner su adapter klase koje prilagođavaju interfejs klase IBaza odgovarajućm interfejsima: IBazaAdministrator, IBazaKorisnik i IBazaRecepcioner.

Ovdje je također primjenjen istovremeno i Bridge patern, koji odvaja klase u posebne hijerarhije, i odvaja module korištenjem interfejsa. Ovaj patern je također primijenjen kod provjere računa pri plaćanju, korištenjem mosta za razdvajanje od sistema za autorizaciju računa/kartica.

Proxy patern je vezan za konkretnu implementaciju baze. Naime, svaka klasa će dobiti proxy objekat baze, koji će u sebi sadržavati konkretnu implementaciju baze, čime se omogućava autorizacija pristupa bazi. Pošto će svi korisnici baze dobiti isti proxy objekat ovim se čuva memorija sistema, jer je klasa baze jako kompleksa i zauzima puno memorije, čime se djelimično primjenjuje i Flywight patern.

Moguća primjena ostalih strukturalnih paterna:

Kada bi sistem imao neku vrstu hijerarhije, npr. pri informisanju o usluzi korisnik može nazvati informacije, koje ga dalje prosljeđuju recepcioneru, koji ga dalje prosljeđuje administratoru, tada bi se mogao iskoristiti Composite patern.

Ako bi se omogućila promjena usluge za vrijeme korištenja iste, to bi omogućilo korištenje Decorator paterna.

Facade patern bi se mogao iskoristiti ako bi se unutar sistema implementirao poseban način plaćanja, koji je vezan za sistem, ali je jako kompleksan.

Kreacijski paterni

Na dijagramu su iskorištena 3 kreacijska paterna: Builder patern, Prototype patern i Singleton patern.

Pošto klasa koja predstavlja rezervacije ima puno atributa, pa bi kreiranje instance bilo zamorno i kompleksno, ovdje je iskorišten Builder patern. On omogućava da se rezervacija kreira postepeno, i omogućava to da klasa Rezervacija bude imutable klasa.

Pošto je jako puno klasa naslijeđeno iz klase Osoba, onda bi se neizbježnom primjenom polimorfizma, kreiranje identičnog objekta jako zakomplikovalo. Idealno rješenje za ovo je primjena Prototype paterna, koji rješava ovaj problem tako što će se prilikom dupliciranja objekta koristiti clone metoda.

Pošto je administrator jedinstven na nivou sistema, klasa koja ga predstavlja će biti Singleton klasa, kako bi se onemogućilo kreiranje više instanci. Također, konkretna klasa vezana za bazu je singleton klasa, kako bi se izbjegli problemi istovremenog višestrukog pristupa bazi.

Što se tiče ostalih kreacijskih paterna imamo još Factory method i Abstract factory paterne.

Factory method patern bi se mogao iskoristiti pri kreiranju novih instanci novih vrsta soba, ako bi se hotel odlučio za naknadnu promjenu soba, tako da ima više različitih vrsta na osnovu nekog kriterija, tako da bi se omogućila kasnija promjena kreiranja soba u odnosu na neki određeni kriterij, npr. na osnovu zabilježenih preferenci korisnika, bez promjene već napisanog koda.

Ukoliko bismo imali više ponuda u aplikaciji, čime bi korisnik mogao kombinirati koje usluge želi, tada bi se koristio Abstract factory patern.

Paterni ponašanja:

Pošto će naš sistem biti implementiran po MVC paternu, samim tim će Medijator patern biti korišten uvijek kada pogled komunicira sa ostatkom sistema.

Observer patern: nakon što korisnik napravi rezervaciju, ta rezervacija će se pratiti i korisnik će dan prije početka rezervacije dobiti obavijest da mu rezervacija počinje sutra.

State patern: instance klase Soba će se na osnovu promjene stanja različito ponašati, nakon što soba postane slobodna, njeno stanje će se automatski postaviti na „slobodno“, tj. kada je soba zauzeta, drugi korisnik neće moći rezervisati tu istu sobu.

Ostali paterni ponašanja:

Command patern se može koristiti za odabir svih usluga hotela. Nakon dodavanja dodatnih usluga hotela, poput restorana, gost bi imao veliki izbor usluga, pa bi se korištenjem ovog paterna svi izbori gosta izvršili odjednom, čime bi se pojednostavio rad sistema i njegova efikasnost.

Chain of Responsibility patern bi se koristio ako bi sistem imao ranije navedeni način informacije korisnika o usluzi.

Iterator patern bi se koristio ako bi sistem imao posebno implementiranu strukturu podataka, koja nije sadržana u bibliotekama, za prolazak kroz elemente iste.

Memento patern se može iskoristiti, ako se omogući individualno prilagođavanje aplikacije, čime bi korisnik dobio uvijek onakav izgled kakkov je prethodno izabrao.

Ako se omogući kreiranje rezervacije na više mogućih načina, tada bi se mogao koristiri strategy patern.

Ako bi se, u sistem dodale još mnogobrojne ponude, tada bi odabir usluge bio jako komplikovan, pa bi Template Method u mnogome pomogao time što bi svaka klasa učestvovala u logici sistema, onako kako ona „zna“.

Kada bi bilo potrebno npr. serijalizirati objekte i spasiti ih u bazu, tada bi se koristio visitor patern, kako bi se ovo omogućilo bez naknadne promjene postojećih klasa.