

Strukturalni paterni

1. Adapter pattern

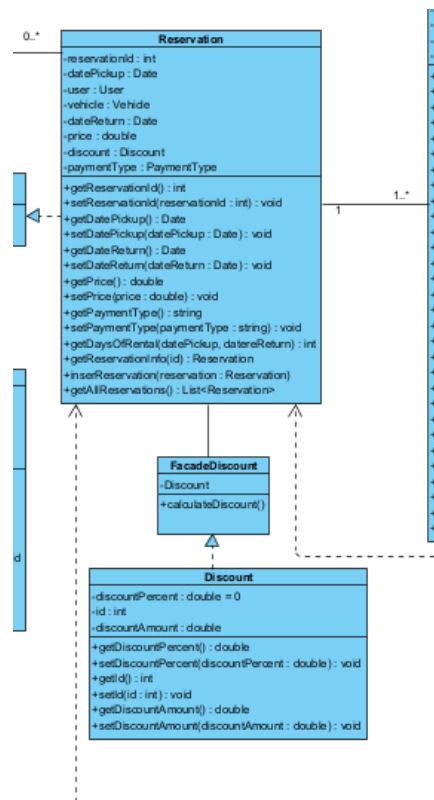
Adapter pattern omogućava objektima sa nekompatibilnim interfejsima da komuniciraju međusobno.

Ovaj pattern bi se mogao iskoristiti prilikom računanja popusta kad bi npr klasa Discount bila neki sistem koji računa popust, ali zahtjeva specifičan ulaz. To može biti obični string ili neki drugi format podataka.

2. Facade pattern

Facade pattern koristimo ako imamo više nekih podsistema koji vrše obradu, mi ovim patternom sakrivamo kompleksnost te obrade i klijentu dajemo jednostavan interfejs.

Ovaj pattern možemo iskoristiti prilikom plaćanja rezervisanja vozila. Ukoliko postoji više metoda plaćanja možemo napraviti fasadu i klasi Reservation dati interfejs s jednom metodom da se procesira plaćanje. Također moglo bi se iskoristiti ukoliko imamo različite kriterije za obračun popusta, onda možemo iskoristiti Facade pattern i tako prekriti kompleksnost tog proračuna. Takvo nešto se već nalazi u našem klasnom dijagramu samo što trenutno postoji jedna Discount klasa koja računa popust na osnovu broja dana rezervacije (npr. drugi način bi mogao biti po učestalosti rezervacija). Na ovaj način podržavamo OCP princip.



3. Decorator pattern

Decorator pattern koristimo kako bi dinamički dodali neke funkcionalnosti postojećim objektima.

To bi u našem sistemu bilo moguće iskoristiti prilikom registracije korisnika. Npr da pored slanja verifikacije na mail možemo poslati verifikaciju preko poruke ili čak na još neki način. Također ovaj pattern možemo iskoristiti prilikom prikaza vozila korisniku npr. za dodavanje raznih filtera po boji, imenu, godištu itd.

4. Bridge pattern

Bridge pattern moguće je iskoristiti na sljedeće načine:

Bridge pattern omogućava razdvajanje apstrakcije i implementacije klase tako da klasa može imati više apstrakcija i više implementacija za apstrakcije. Koristi se kada imamo 2 ili više ortogonalne hijerarhije klasa.

U našem dijagramu ovaj pattern bi se mogao iskoristiti jedino kod klase User jer su iz nje izvedene Admin i Registrovani korisnik i imamo tu hijerarhijsku strukturu.

5. Composite pattern

Osnovna namjena ovog patterna je da se omogući formiranje strukture drveta pomoću klasa, u kojoj su objekti i kompozicije objekata jednaki. Composite pattern je sličan sa dekoratorom.

Pošto u našem projektu nemamo izraženu strukturu drveta ovaj pattern nećemo koristiti.

6. Proxy pattern

Pomoću proxy patterna možemo izvršiti kontrolu pristupa pojedinim elementima sistema.

U našem sistemu postoji administrator i obični korisnik i samim time ne bi bilo dobro da korisnik može upravljati svim rezervacijama ili brisati i dodavati vozila u sistem. Iz tog razloga moguće je primijeniti proxy pattern. Drugim riječima proxy patternom se vrši kontrola pristupa pojedinim dijelovima sistema.

7. Flyweight pattern

Flyweight pattern se koristi za potencijalne ušetede memorije na podacima koji se ponavljaju ukoliko imamo veliki broj objekata.

Pošto za vozila dosta podataka se ponavlja, pa recimo podaci kao što su kategorija, tip goriva, broj sjedista, vrata ili tip mjenjača su često isti za dosta vozila. Tako da će se pri ubacivanju novog vozila napraviti tih par objekata s tim karakteristikama umjesto za svako vozilo ti podaci iznova.

Objektno Orijentisana Analiza i Dizajn

