Strukturalni paterni

Adapter patern

Adapter patern omogućava širu upotrebu već postojećih klasa. Ukoliko je potreban drugačiji interfejs već postojeće klase, a ne želimo mijenjati postojeću klasu koristi se Adapter patern koji kreira novu adapter klasu.

Recimo da korisnik može da upload-uje video uz recept u zastarjelom formatu, kao što je AVI, a želimo da omogućimo i opciju upload-a u formatu koji podržava HD reprodukciju, kao što je MKV. Javio bi se problem nekompatibilnih interfejsa klasa ta dva formata (ili više takvih), pa bi se to moglo riješiti upotrebom Adapter paterna. Uz pomoć Adapter klase koja implementira interfejse obje klase (AVI i MKV), omogućili bi poziv univerzalne metode tog adaptera, koja bi vršila reprodukcije niže rezolucije kada je u pitanju AVI video, a HD reprodukciju kada je u pitanju MKV format.

Facade patern

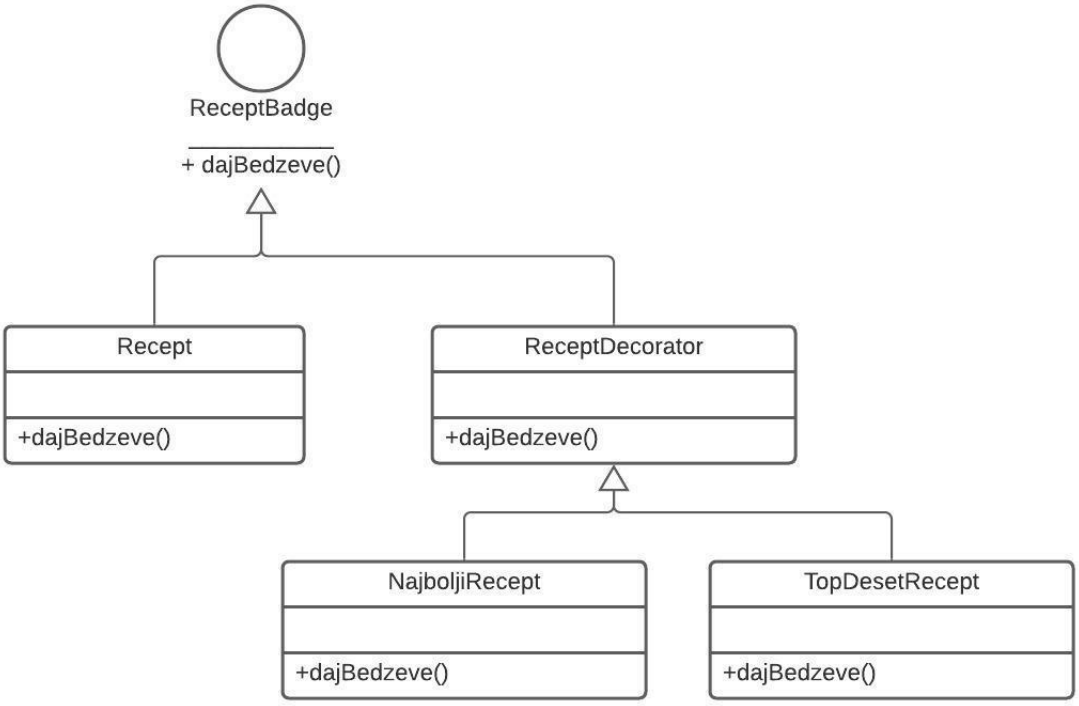
Facade patern osigurava više pogleda visokog nivoa na podsisteme (implementacija podsistema skrivena od korisnika).

Kada bi omogućili da korisnik može dati ocjenu i komentar istovremeno, a ne odvojeno, mogli bi napraviti fasadnu klasu *Recenzija*. Unutar te klase bile bi instance klase *Ocjena* i klase *Komentar* i ukinuli bi metode *dodajOcjenu*, *dodajKomentar*, *ukloniKomentar*, *ukloniOcjenu* i sl., a umjesto njih bi imali samo *dodajRecenziju* i *ukloniRecenziju*. Dakle, izgledalo bi kao da postoji samo klasa *Recenzija*, iako su u njoj sadržane klase *Ocjena* i *Komentar*. Međutim, u našoj aplikaciji data je mogućnost davanja ocjene bez komentara i obrnuto, tako da nema potrebe za tim.

Decorator patern

Osnovna namjena Decorator paterna je da omogući dinamičko dodavanje novih elemenata i ponašanja (funkcionalnosti) postojećim objektima. Objekat pri tome ne zna da je urađena dekoracija što je veoma korisno za ponovnu upotrebu komponenti softverskog sistema.

Ukoliko dodamo mogućnost da korisnici osvajaju bedževe na svojim receptima, koji se prikazuju uz iste, može nam poslužiti Decorator patern. Interfejs *ReceptBadge* definira zahtjev za metodom *dajBedzeve* koja će dati potrebne podatke o bedževima trenutnog recepta. Mogući bedževi su “Najbolji recept”, za recept koji je barem jednom bio na 1. mjestu među 10 najboljih recepata, te “Top deset” bedž za recepte koji su ušli u najboljih deset recepata. Postoji mogućnost dodavanja novih bedževa kreiranjem klase za bedž koja nasljeđuje Decorator klasu.



Bridge patern

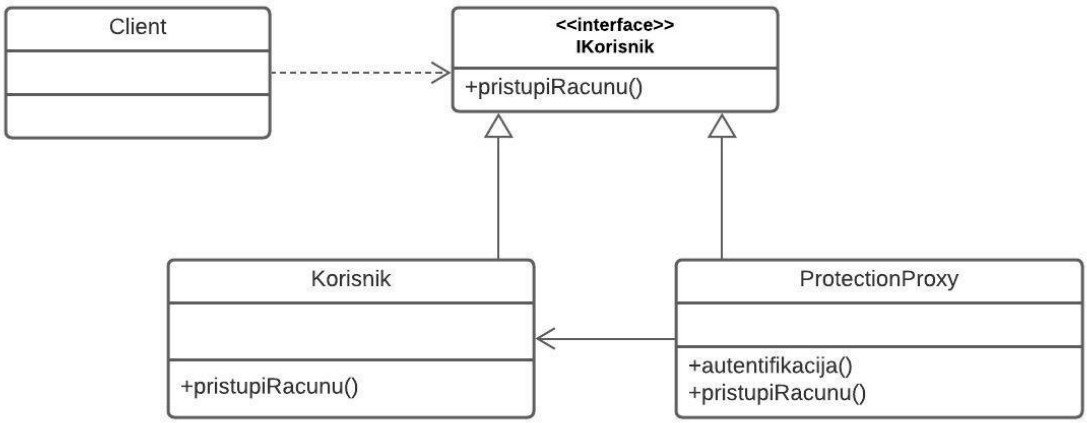
Bridge patern omogućava odvajanje apstrakcije i implementacije neke klase tako da ta klasa može posjedovati više različitih apstrakcija i više različitih implementacija za pojedine apstrakcije.

Kako u našoj aplikaciji treba pružiti različite GUI za registrovanog i neregistrovanog korisnika, te administratora, Bridge patern se može tu primjeniti.

Proxy patern

Namjena Proxy paterna je da omogući pristup i kontrolu pristupa stvarnim objektima.

Ovaj patern koristimo za kontrolu unosa lozinke prilikom prijave. U slučaju ispravno unesene lozinke, odobrava se pristup i korisnik će biti prijavljen na sistem, a u suprotnom korisnik dobija obavještenje o neispravnom unosu i mogućnost ponovnog unosa.



Composite patern

Da bi se mogao primjeniti ovaj patern potrebno je da se glavni dio aplikacije može predstaviti kao stablo. Međutim, u našoj aplikaciji to nije slučaj pa ovaj patern nećemo koristiti.
Recimo da smo imali apstraktnu klasu *Korisnik* i naslijeđene klase *RegistrovaniKorisnik* i *NeregistrovaniKorisnik*. Ako bismo željeli prikazati preporučene recepte za ove korisnike, mogli bismo prikazivati najnovije recepte za registrovane, a najbolje ocijenjene za neregistrovane korisnike. Za implementaciju ovog paterna trebali bi promijeniti klasu *Korisnik* u interfejs *IKorisnik* koja bi implementirala metodu *DajPreporuceneRecepte*.

Flyweight patern

Flyweight patern omogućava da više različitih objekata dijele isto glavno stanje, a imaju različito sporedno stanje.

Ovaj patern korisitimo kada sastojcima već upisanim u bazu mijenjamo atribute *kolicina* i *mjerna jedinica*.