Univerzitet u Sarajevu
Elektrotehnički fakultet u Sarajevu
Odsjek za računarstvo i informatiku

Strukturalni paterni
eDostava

Objektno orijentisana analiza i dizajn

Naziv grupe: eDostava
Članovi: Ina-Bedra Mujaković
Ema Kalmar
Rijad Zejnilović

1. Adapter patern

Svrha Adapter patern-a je da omogući širu upotrebu već postojećih klasa. U situacijama kada je potreban drugačiji interfejs već postojeće klase, a ne želimo mijenjati postojeću klasu koristi se Adapter patern. Adapter patern kreira novu adapter klasu koja služi kao posrednik izmedu originalne klase i željenog interfejsa. Adapter patern možemo koristiti u zastarjelim sistemima koji funkcionišu dobro, čije metode možemo i dalje koristiti, ali želimo nadograditi sistem nekim novim funckionalnostima.

Adapter patern bi se u našem slučaju mogao koristiti u situaciji kada administratoru sistema treba lista svih slobodnih dostavljača (vozača) artikala na adresu. Administrator bi pored pregleda liste mogao i vršiti sortiranje (filtiranje) liste po nekom parametru, naprimjer: vozač koji je najbliži nekom restoranu, ide po narudžbu ili recimo vozač koji je slobodan ide na neku lokaciju i slično. Ovo sve bi postigli kreiranjem AdapterKlase koja bi komunicirala s ostatkom sistema.

2. Facade patern

Svrha Facade patern-a je što pruža jedinstven interfejs na skup interfejsa u podsistemu. Facade definira interfejs na višem nivou koji olakšava upotrebu podsistema. Motivacija Facade paterna je upravo to što organiziranje sistema u podsisteme pomaže u smanjenju kompleksnosti. Uobičajeni cilj dizajna je da se minimizira komunikacija i ovisnosti izmedu podsistema.

S obzirom da je naš sistem izuzetno jednostavan, ne postoji baš puno situacija u kojima bi mogli iskoristiti ovaj patern. Jedan od načina na koji bi mogli iskoristiti **Facade patern** jeste da sve funkcionalnosti koje posjeduje naš sistem smjestimo u jednu klasu (kontejnersku) sa svim listama objekata i metodama.

3. Decorator patern

Svrha Decorator patern-a je 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.

U našem sistemu postoji klasa Korisnik koja služi za prikaz nekih osnovnih informacija o Korisniku, specifično neke lične informacije. U budućnosti se može pojaviti potreba za širenjem ove klase te dodavanjem novih atributa i metoda (mogućnosti) koje će korisnik moći koristiti. Kako ne bi mijenjali klasu Korisnik i time naštetili cijelom sistemu, ovdje možemo primjeniti **Decorator patern**, tako što ćemo dodati interfejs IKorisnik i klasu Decorator koja bi sadržavala dodatne metode i atribute koje želimo implementirati.

4. Bridge patern

Svrha Bridge patern-a je omogućavanje da se iste operacije primjenjuju nad različitim pod klasama. Ovim izbjegavamo nepotrebno dupliciranje koda, odnosno i zbjegavamo kreiranje novih metoda za već postojeće funkcionalnosti.

U sklopu našeg sistema, naprimjer u klasama Administrator i Narudžba imamo metode DodajProizvod i ObrišiProizvod pa bi se ovdje mogao iskoristiti **Bridge patern**.

5. Composite patern

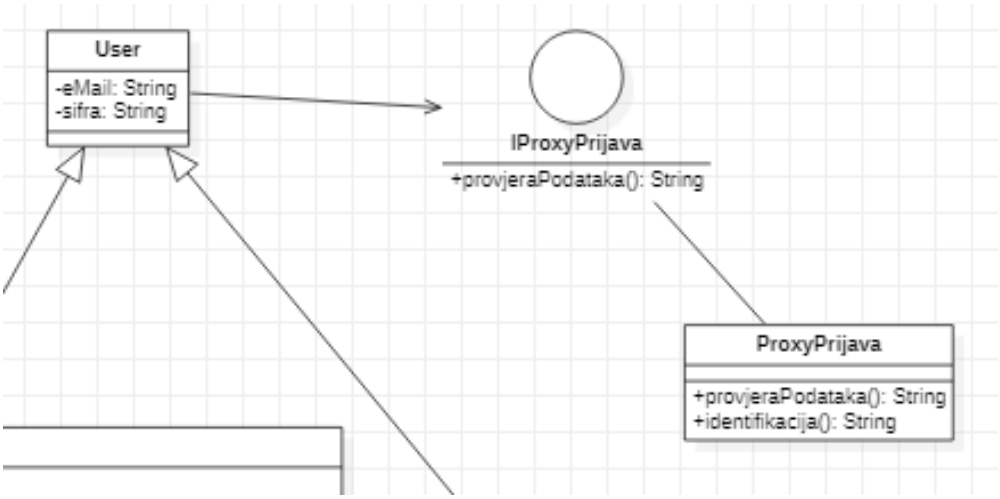
Osnovna namjena Composite patern-a je da omogući formiranje strukture stabla pomoću klasa, u kojoj se individualni objekti (listovi stabla) i kompozicije individualnih objekata (korijeni stabla) jednako tretiraju. Ovo zapravo znači da je moguće pozivati metodu koja je zajednička na nivou svih tih klasa. Osnovna primjena ovog paterna jeste pravljenje hijerarhije klasa i omogućavanje pozivanja iste metode nad različitima objektima sa razičitim implementacijama.

Ukoliko bismo željeli dopustiti administratoru i kuriru da imaju pregled svih narudžbi (onih u pripremi naprimjer ili onih već obavljenih), mogli bismo primijeniti **Composite patern**.

6. Proxy patern

Svrha Proxy patern-a je da omogući pristup i kontrolu pristupa stvarnim objektima. Proxy je obično mali javni surogat objekat koji predstavlja kompleksni objekat čija aktivizacija se postiže na osnovu postavljenih pravila.

U našem sistemu možemo iskoristiti **Proxy patern** tako što bi ograničili prava pristupa određenim funkcionalnostima sistema. Dakle, nakon logina (ukoliko je korisnik uopšte registrovan) vršila bi se provjera podataka i na taj način bi se odredili prava pristupa nekim funkcionalnostima. Nakon logina bi se ustanovilo da li je korisnik admin ili ne, da li je registrovan ili je na stranici kao gost. Naravno, admin ima mnogo više funkcionalnosti od ostalih. Kako bi ovo funkcionisalo, dodat ćemo interfejs 'IproxyPrijava“ i klasu ProxyPrijava koja ima metodu preko koje se vrši provjera podataka i identifikacija korisnika na osnovu unesenih podataka.



7. Flyweight patern

Flyweight patern se može primjeniti da bi se napravio objekat koji bi minimizirao utrošak memorije tako što on dijeli što više podataka sa njemu sličnim objektima. Njegova glavna funkcija je da omogući da više zasebnih objekata dijele isto glavno stanje, a da im sporedna stanja budu različita.

U našem sistemu imamo tri 'glavne' klase Narudžba, Kurir i Korisnik i svaka od njih ima mnogo bitnih atributa koji opisuju te klase i tu jednostavno ovaj patern nije moguće primijeniti. Situacija u kojoj bi mogli iskoristiti **Flyweight patern** jeste u klasi Proizvod gdje imamo jedan od atributa 'slika', koji bi mogli postaviti na defaultnu sliku (naprimjer korisnik odabere restoran i želi da naruči pizzu, slika pizze bi bila defaultna) i na taj način bi imali racionalniju upotrebu resursa.