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.

Ukoliko korisnik želi saznati u kojem se stanju trenutno nalazi njegova narudžba (u pripremi, spremna za dostavu ili na putu), možemo kreirati **Adapter patern** koji će ispuniti želju korisnika. Također, ako korisnik obavi narudžbu i naknadno se prisjeti kako želi poručiti još nešto, dodavanjem novog **Adapter paterna** mu je omogućeno da izvrši izmjenu prethodno izvršene narudžbe u određenom vremenskom roku.

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.

Pri implementaciji dijagrama klasa, iskoristili smo **Facade patern** kreiranjem klasa Narudžba i Proizvod i na taj način smanjili kompleksnost sistema.

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.

Ovaj patern bi bilo jako teško implementirati u naš sistem, jer nemamo nijednu razumnu situaciju za njegovu implementaciju.

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 će se recimo klikom na sliku proizvoda, jela ili recimo nekih kućanskih potrepština slika prikazati u većoj i boljoj rezoluciji. Također, preko ovog paterna bi omogućili pristup profilu dostavljača, njegovoj slici i nekim ličnim podatcima kako bi korisnik znao koga da očekuje na svojoj adresi.

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.

Najjednostavniji način primjene **Flyweight paterna** bi bio za recimo praćenje stanja narudžbe. Administrator bi mogao nadgledati stanje svake narudžbe i u slučaju nekog problema, informisati korisnika na vrijeme. Svaka narudžba bi imala neki tekstualni atribut koji bi opisivao njeno trenutno stanje. Također, možemo i pratiti trenutni položaj narudžbe u toku dostave, odnosno dostavljača. U tom slučaju bi svaki od njih imao neki atribut za grafički prikaz i na taj način bi se svi oni mogli pozicionirati na mapi.