STRUKTURALNI PATERNI

1. Adapter patern (implementirano)

Ukoliko želimo nadograditi naš sistem na način da kupcu omogućimo prikaz sortiranih artikala po cijeni (pored metode getArtikli(), koja vraća sve artikle jednog prodavača, treba da imamo i metodu getArtikliSortiraniPoCijeni(), koja vraća sve artikle sortirane po cijeni u rastućem poretku) ili po nazivu(getArtikliSortiraniPoNazivu()), možemo iskoristiti Adapter patern.

Adapter patern koristimo kada je potreban drugačiji interfejs već postojeće klase, a ne želimo mijenjati postojeću klasu. Tim postupkom se dobija željena funkcionalnost bez izmjena na originalnoj klasi.

Ukoliko želimo da svi artikli koje je objavio jedan prodavač budu sortirani po cijeni ili po nazivu mijenjamo interfejs I time nadograđujemo naš sistem. Definišemo interfejs ISortiranje sa metodama *getArtikliSortiranoPoCijeni() I getArtikliSortiranoPoNazivu().* Definišemo klasu ArtikalAdapter, koja implementira interfejs ISortiranje.

Na ovaj način možemo unaprijediti interfejs klase Artikal i sa drugim metodama, a da time ne modificiramo postojeću klasu Artikal. Ovim načinom nadogradnje aplikacije poštujemo OTVORENO ZATVOREN PRINCIP, budući da nadograđujemo entitete softvera bez njihove modifikacije.

1. Decorator patern

Decorator patern služi za omogućavanja različitih nadogradnji objektima koji svi u osnovi predstavljaju jednu vrstu objekta. Umjesto da se definiše veliki broj izvedenih klasa, dovoljno je omogućiti dodavanje različitih detalja, te se na taj način pojednostavljuje i rukovanje objektima klijentima, i samo implementiranje modela objekata. Decorator patern se koristi u slučajevima kada želimo dodijeliti dodatna ponašanja objektu tokom izvođenja programa bez da mijenjamo kod koji je na neki način u interakciji sa datim objektom.

Pošto Artikal posjeduje atribut slika, moguće je u budućnosti da poželimo omogućiti korisniku manipulacije sa slikom (npr. rotacija). U tom slučaju je prvo potrebno u sistem dodati novu klasu (Slika), odnosno odvojiti aktivnosti vezane za sliku, od klase Artikal kako ne bi bio narušen S princip. To u našem slučaju nije bilo potrebno, pošto sistem ne vrši nikakve manipulacije sa slikom. Slika je samo jedan od atributa Artikla. Potrebno je da napravimo interfejs *ISlika*, koji idetificira klase objekata koje želimo rotirati. Interfejs treba sadržavati metodu rotiraj(), dok klasa Slika treba da implementira interfejs ISlika. Ta klasa je osnovna komponenta.

Svaki konkretan dekorator treba da sadrži kao atribut ISlika, potom treba da implementira interfejs ISlika, te u slučaju potrebe dodati nove metode.

1. Bridge patern

Ukoliko želimo uvesti različite implementacije neke funkcionalnosti, npr. dodavanje popusta pri dopuni ETF.

Osnovna namjena Bridge paterna je da omogući odvajanje apstrakcije i implementacije neke klase tako da ta klasa može posjedovati više različitih apstrakcija i više različitih implementacija za pojedine apstrakcije. Bridge patern pogodan je kada se implementira nova verzija softvera a postojeća mora ostati u funkciji.

U našem sistemu postoji klasa Instrukcija getETFKredit() koju bi mogli da proširimo interfejsom Bridge koja bi u sebi imala istoimenu metodu, te bi na osnovu toga da li postoji popust pozivala dvije različite implementacije od kojih bi jedna vraćala iznos ETFKredita bez popusta, a druga iznos sa popustom.

1. Proxy patern (implementirano)

Ukoliko želimo dodatnu sigurnost u sistemu, pri ostavljanju review-a nakon izvršene kupoprodaje, to možemo ostvariti pomoću Proxy paterna. Namjena Proxy paterna je da omogući pristup i kontrolu pristupa objektima ili metodama. Mi imamo klasu Kupac I kupci će moći nakon obavljene kupoprodaje ostaviti komentar na artikal. Taj dio sistema ćemo osigurati sa Proxy paternom da bismo osigurali da samo Kupci koji su obavili kupoprodaju I kupili taj artikal mogu ostavljati komentar na dati artikal.

Da bismo to postigli dodali smo interfejs IproxyKomentarisanje i klasu ProxyKomentarisanje koja kroz metodu odobrenoKomentarisanje(komentar: Komentar, kupac : Kupac) vrši autentifikaciju Kupca koji hoće da ostavi komentar, ako je zaista kupio dati artikal

Na ovaj način možemo unaprijediti sigurnost sistema, a da time ne modificiramo postojeće klase. Ovim načinom nadogradnje aplikacije poštujemo O princip, budući da nadograđujemo entitete softvera bez njihove modifikacije.

1. Composite patern

Osnovna namjena kompozitnog paterna je omogućavanje formiranja strukture stabla pomoću klasa, u kojoj se kompozicije individualnih objekata (korijeni stabla) i individualni objekti (listovi stabla) ravnopravno tretiraju, odnosno moguće je pozvati zajedničku metodu nad svim klasama. Omogućava nam uređenu hijerarhijsku strukturu koja zadržava uniformnost tako što je moguće istu metodu primjeniti nad različitim implementacijama.

U našem sistemu možemo proširiti klasu Artikal korisničkim zahtjevom da postoje dvije vrste artikla, npr. Novi I polovni. Tako bi se stvorila binarna struktura u kojoj je klasa Artikal korijen stabla, a nove klase NoviArtikal I PolovniArtikal su listovi stabla. Da bi ostvarili kompozitni patern potrebno je implementirati interfejs ICijena koji ima metodu dajCijenu() koja vraća cijenu u ovisnosti da li je nov ili polovan artikal. Novi I polovni artikli nemaju ništa zajedničko, te se cijene vraćaju na različit način s različitom programskom logikom.

1. Flyweight patern

Primarna funkcija Flyweight paterna je da dozvoli da više različitih objekata dijele isto glavno stanje, a da im sporedna stanja budu različita. Koristimo ga jer pomoću njega postižemo manju i racionalniju upotrebu resursa kao što je radna memorija i znatno brže izvršavanje programa.

Naš sistem je moguće prosiriti zahtjevom da se omogući svim korisnicima aplikacije da imaju svoju profilnu sliku. U tom slučaju bilo bi potrebno koristiti unaprijed zadanu sliku, ukoliko korisnik ne bude htio da objavi svoju sliku. S obzirom da je moguće da više korisnika zadrži tu početnu sliku potrebno je implementirati flyweight pattern kako bi ti korisnici koristili jedan zajednički resurs.

1. Facade patern

Glavna namjena facade paterna je osiguravanje više pogleda visokog nivoa na podsisteme čija je implementacija skrivena od korisnika.

Naš sistem je moguće proširiti, tako što bi kreirali klasu FakturaFasada koja bi objedinjavala naše podsisteme vezane za klase Artikal I Kupac. Klasa FakturaFasada bi implementirala metode:

* SpasiArtikal I NaruciArtikal
* IzbaciArtikalIzKorpe
* promijeniNacinPlacanja
* promijeniAdresuDostave
* otkaziNarudzbu