Ayana cvjećara

Strukturalni patterni

*paterni koji će biti implementirani sigurno su boldirani

1. *Adapter pattern*

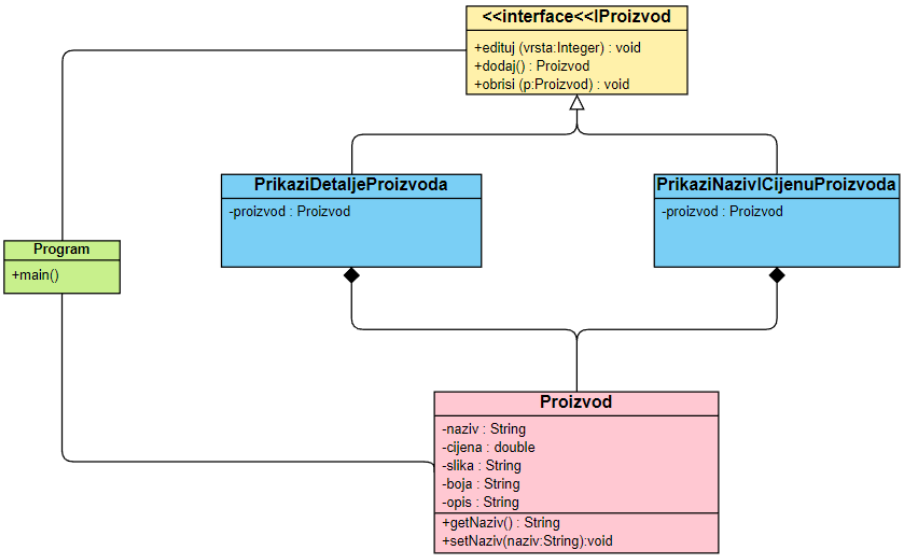
Ovaj pattern se može primijeniti prilikom pretraživanja proizvoda. Pretraga se vrši prema nazivu, cijeni I vrsti cvijeća. Rezultati pretrage se sortiraju prema poretku koji korisnik odabere. Koristimo navedeni pattern zato što nam je potreban drugačiji interface postojeće klase *Proizvod,* a ne želimo mijenjati klasu pa kreiramo *Adapter klasu* koja će služiti kao posrednik kako bismo implementirali funkcionalnost pretraživanja bez izmjene postojeće klase. Konkretnije, radi se o *Class Adapter pattern-u,* kreira se interface *iSearch* s metodom *sortProducts()* koja će vraćati listu sortiranih proizvoda. Definiše se klasa *ProductAdapter* koja će naslijediti navedeni interface.

2. *Facade pattern*

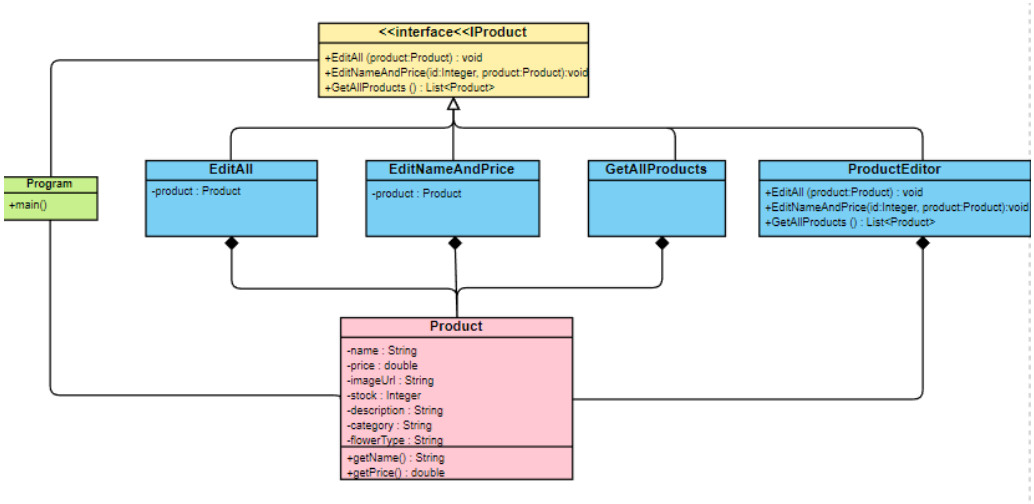
Facade pattern koristimo za prikaz *izvještaja* koji će biti dostupan zaposleniku cvjećare. Sistem aktivno generiše izvještaj prema algoritmu koji će zaposleniku biti dostupan u formi excel tabele, pri čemu zaposlenik bira vrstu izvještaja I ima mogućnost pregleda I preuzimanja izvještaja. Radi se o izvještaju o poslovanju koji nije od interesa korisnicima, a algoritam koji se koristi pri kreiranju upotrebljava metode različitih klasa zbog svoje sveobuhvatnosti, ali nije od interesa zaposleniku kojeg zanima samo krajnji rezultat, odnosno tabela s podacima.

3. Decorator pattern

Zaposlenik ima mogućnost promjene opisa proizvoda (naziva, cijene, kategorije, podkategorije, opisa, slike I boje). Promjene nad proizvodima možemo podijeli u tri skupine: *EditAll* koja omogućava promjenu svih detalja proizvoda I *EditNameAndPrice* koja će davati uvid samo u naziv I cijenu proizvoda jer su to atributi koji su nam najvažniji I najviše korišteni za razne funkcionalnosti I algoritme te metodu *GetAllProducts* Uvodimo *Decorator* klasu koja je povezana agregacijom s *IProizvod* interface-om I instancira jedan ili više *IProizvod* objekata.



Nakon implementacije, izmijenjeni dijagram izgleda ovako:



4. *Bridge pattern*

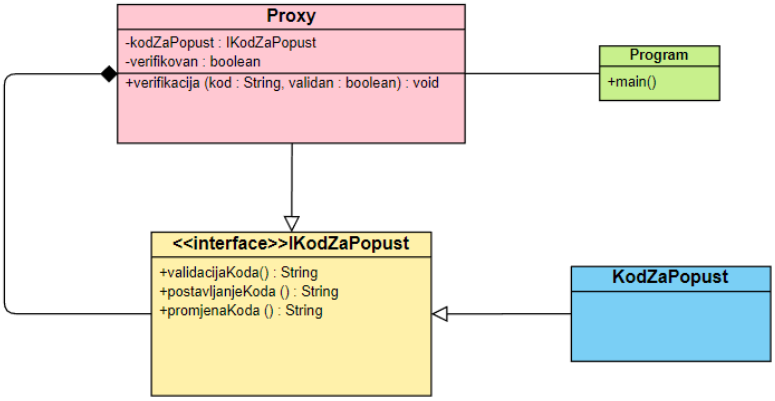
Korisnik koji nije prijavljen može kao I registrovani korisnika vršiti pregled proizvoda I njegovih detalja, kao I best seller proizvoda iz naše ponude. Ova aktivnost spaja dvije vrste korisnika I ona se sigurno izvršava. Međutim, da bi kupac odabrao opciju *Add to cart* I izvršio kupovinu, on mora biti registrovan. Ukoliko to nije slučaj, korisniku se nudi mogućnost *Log in* ili *Register* kako bi mogao izvršiti kupovinu.

5. *Composite pattern*

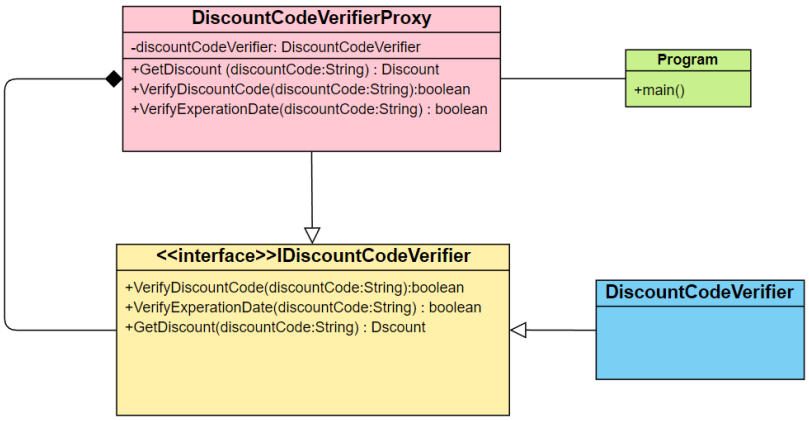
Ovaj pattern može naći primjenu u računanju ukupne cijene koju korisnik plaća za obavljenju narudžbu. Računanje bi se obavljalo, naravno, pri svakom plaćanju. Međutim, ukoliko kupac ostvari popust, cijena se koriguje. Klase će imati interface za istu svrhu

6. *Proxy pattern*

Prilikom plaćanja narudžbe, kupac ima mogućnost unosa koda za popust. Kod za popust koji je dobio putem email-a je potrebno verifikovati, odnosno provjeriti njegovu ispravnost u smislu tačnosti koda I provjere da li kod važi u datom trenutku zbog ograničenog trajanja koda. Dakle, za verifikaciju koda I odobravanje popusta na cijenu narudžbe koristimo *Proxy pattern*.



Nakon implementacije, dijagram izgleda ovako:



7. *Flyweight pattern*

Pri kupovini *subscriptions* paketa, zbog fiksnih unaprijed određenih karakteristika, ne pravimo razliku između paketa iz iste kategorije. Dakle, ako kupac kupuje paket iz određene kategorije koji je već instanciran, mi ćemo iskoristiti podatke od već napravljenog paketa (uz korigovanje poruke za personal card).