

## Strukturalni patterni

\*patterni koji će biti implementirani sigurno su boldirani

### 1. *Adapter pattern*

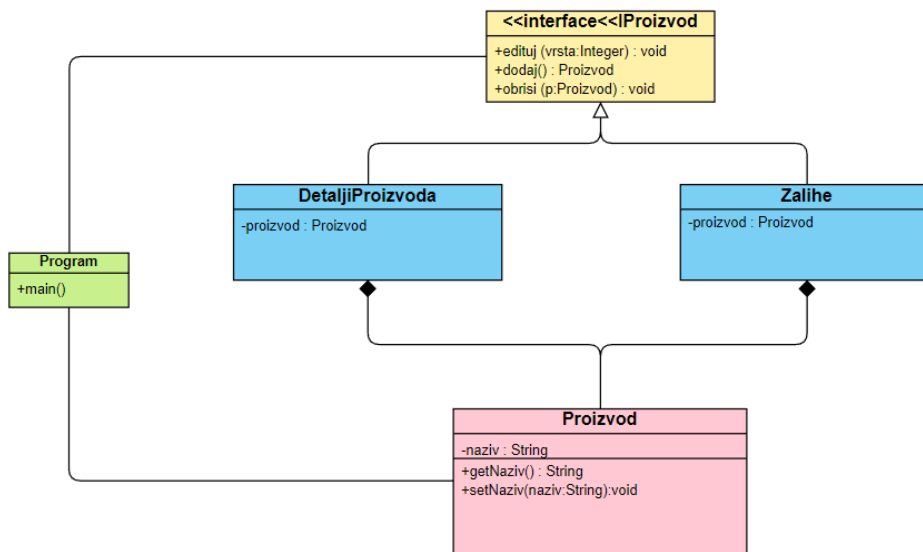
Ovaj pattern se može primijeniti prilikom pretraživanja proizvoda. Pretraga se vrši prema nazivu, cijeni i vrsti cvijeća. Rezultati pretrage se sortiraju prema poretku koji korisnik odabere. Koristimo navedeni pattern zato što nam je potreban drugačiji interface postojeće klase *Proizvod*, a ne želimo 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.

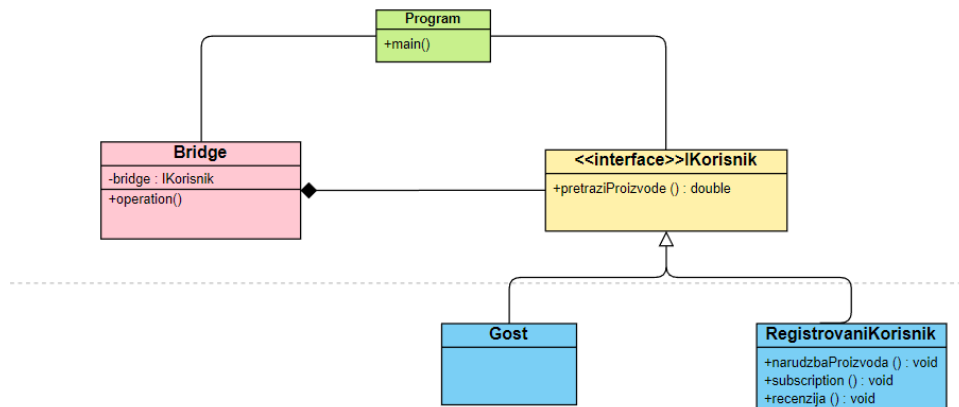
### 3. *Decorator pattern*

Zaposlenik ima mogućnost promjene detalja o proizvodu (naziva, cijene, kategorije, podkategorije i opisa). Sistem na osnovu toga da li se konkretni proizvod nalazi u nekoj narudžbi povećavati broj prodanih primjeraka i smanjivati zalihe, da bismo imali uvid u to da li je proizvod na stanju. Promjene nad proizvodima možemo podijeliti u dvije skupine: *DetaljiProizvoda* (naziv, cijena, kategorija, podkategorija, opis) i *Zalihe* (listaNarudžbi, historijaProdaje, zaliha). Uvodimo *Decorator* klasu koja je povezana agregacijom s *IProizvod* interface-om i instancira jedan ili više *IProizvod* objekata.



#### 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.

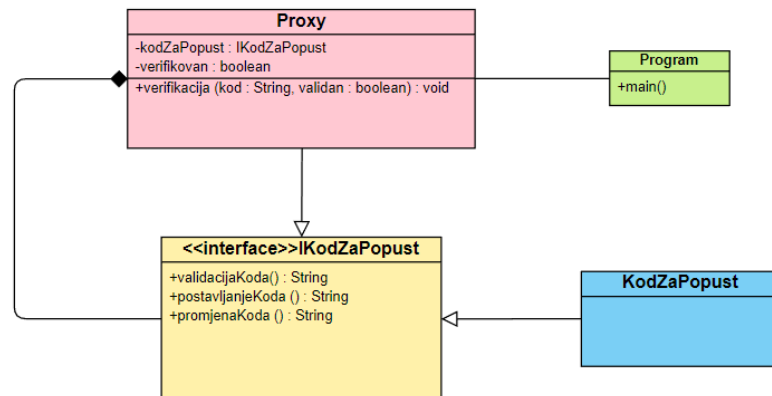


#### 5. Composite pattern

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*.



## 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).