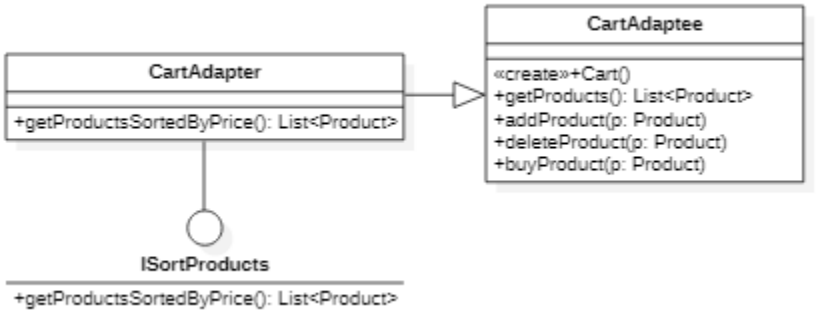
STRUKTURALNI PATERNI

ADAPTER PATERN

Adapter patern služi da se postojeći objekat prilagodi za korištenje na neki novi način u odnosu na postojeći rad, bez mijenjanja same definicije objekta. Na taj način obezbjeđuje se da će se objekti i dalje moći upotrebljavati na način kako su se dosad upotrebljavali, a u isto vrijeme će se omogućiti njihovo prilagođavanje novim uslovima.

U ovom projektu Adapter pattern ćemo omogućiti tako što ćemo u klasi Cart omogućiti prikaz Products sortiranih po cijeni. Tačnije omogućit ćemo da metoda getProducts() vraća sve proizvode ali sortirane po cijeni.
Klasu Cart preimenovat ćemo u CartAdaptee te ćemo dodati klasu CartAdapter koja će implementirati interfejs ISortProducts koja posjeduje metodu getProductsSortedByPrice(). Ova metoda će u sebi pozivati metodu klase CartAdaptee getProducts() međutim mijenjat će je tako da vraća sortiranu listu po cijeni proizvoda.
Ovim smo postigli da je klasa Cart adaptirana i nadograđena. Klasa Cart nije promijenjena samo je unapriđena novim interfejsom. Ovo ne mora biti jedina nadogradnja. Možemo omogućiti sortiranje Products po raznim kriterijima kao što su contidition, color i slično.
U nastavku je prikazano kako je zamišljena primjena ovog paterna na naš dijagram klase.



FACADE PATERN

Fasadni patern služi kako bi se klijentima pojednostavilo korištenje kompleksnih sistema. Klijenti vide samo fasadu, odnosno krajnji izgled objekta, dok je njegova unutrašnja struktura skrivena. Na ovaj način smanjuje se mogućnost pojavljivanja grešaka jer klijenti ne moraju dobro poznavati sistem kako bi ga mogli koristiti.

Facade Patern nećemo implementirati međutim kada bi smo to željeli odabrali bi dio sistema koji je komplikovan no korisnik ne treba znati šta se dešava "ispod haube". Jedan od najkomplikovanijih dijelova ovog projekta jeste dobavljanje informacija iz baze podataka. To zahtjeva veći broj upita koji odgovaraju željenom pozivu. Kako korisnik nema potrebe da zna kako izgledaju ti upiti nego ga samo interesuje krajnji rezultat to je idealno mjesto za korištenje ovog paterna. Primjer kako bi to izgledalo u kodu je dato u nastavku:

```
public class FasadaProduct
{

        public List<Product> getUserProductsForUser(email: String, username: String);
        public List<Products> getCartProductsForUser(email: String, username: String);
        public List<Review> getReviewsForUser(email: String, username: String);
        ...

}
```

DECORATOR PATERN

Decorator patern služi za omogućavanja različitih nadogradnji objektima koji svi u osnovi predstavljaju jednu vrstu objekta (odnosno, koji imaju istu osnovu). Umjesto da se definiše veliki broj izvedenih klasa, dovoljno je omogućiti različito dekoriranje objekata (tj. dodavanje različitih detalja), te se na taj način pojednostavljuje i rukovanje objektima klijentima, i samo implementiranje modela objekata.

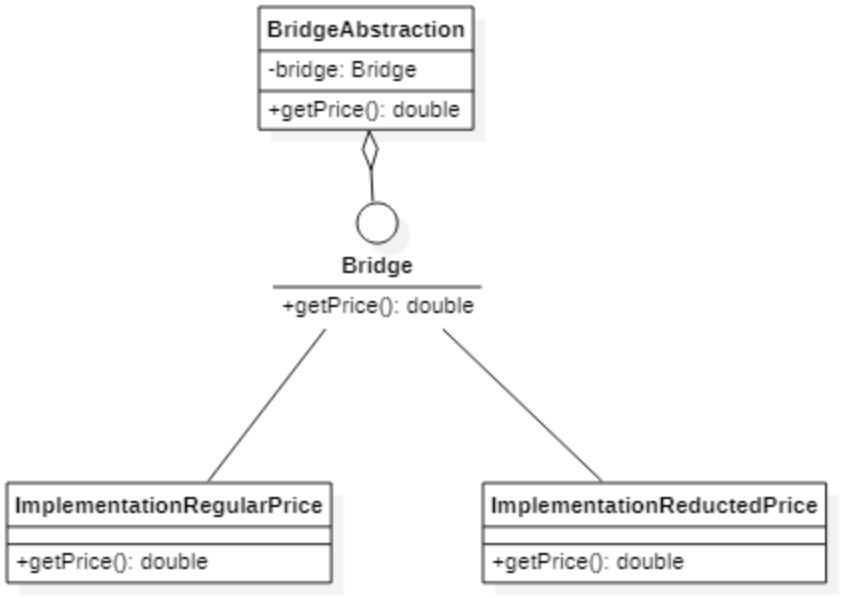
Ovaj Pattern nećemo implementirati no možemo pričati o hipotetskoj implementaciji. Naime bilo bi veoma zgodno iskoristiti ovaj patern na klasi Product da bi se omogućilo editovanje slika Producta koje korisnik želi postaviti. Da bi to omogućili potrebno je izdvojiti klasu Slika kako bi ona mogla implementirati interfejs ISlika koji posjeduje metode dajSliku koja vraca Sliku te postaviSliku. U ovom slučaju će klasa Slika biti Component dok bi Decorator klasa mogla biti klasa SlikaFilteri koja će omogućiti da se slika edituje primjenom

nekog filtera. Takoder vrlo bi se lahko moglo omogućiti i druge vrste manipulacije nad slikom kao što su rotiranje, rezanje i slično.

BRIDGE PATERN

Bridge patern služi kako bi se apstrakcija nekog objekta odvojila od njegove implementacije. Ovaj patern veoma je važan jer omogućava ispunjavanje Open-Closed SOLID principa, odnosno uz poštivanje ovog paterna omogućava se nadogradnja modela klasa u budućnosti te osigurava da se neće morati vršiti određene promjene u postojećim klasama.

Ovaj Pattern nećemo implementirati no ukoliko bismo se odlučili na implementaciju najbolje mjesto za iskoristiti ovaj patern jeste na klasi Product. Ukoliko bi željeli dati određeni popust na Product to bismo uradili preko Bridge Patterna.
Da bi ovo bilo izvodivo odnosno da bi mogli nesmetano izračunati novu cijenu nakon sniženja potrebno je dodati interfejs Bridge koji ima metodu getCijena(). Osim toga omogućili bi dvije implementacije. Jedna bi vraćala regularnu cijenu Product, dok bi druga implementacija vraćala cijenu sniženu u zavisnosti od popusta. Na taj način smo osigurali da će već postojeći sistem regularno raditi.

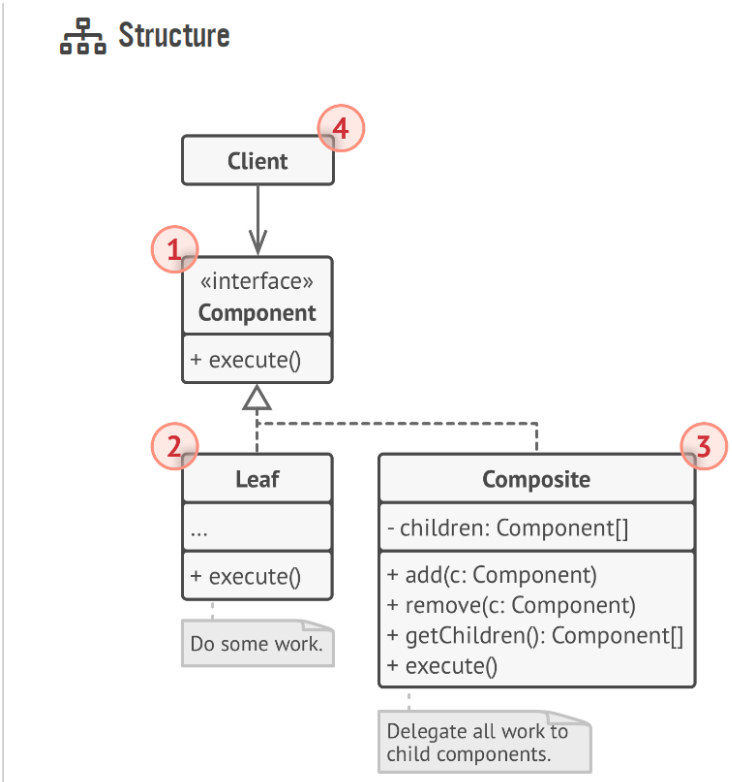


COMPOSITE PATERN

Composite patern služi za kreiranje hijerarhije objekata. Koristi se kada svi objekti imaju različite implementacije nekih metoda, no potrebno im je svima pristupati na isti način, te se na taj način pojednostavljuje njihova implementacija.

Korištenje Composite paterna ima smisla jedino da aplikacija može biti predstavljena kao stablo. On pruža dva osnovna tipa elemenata koja dijele zajednički interfejs: listovi i složeni kontejneri. Kontejner se može sastojati i od listovai od drugih kontejnera. To omogućuje da se konstruiše ugniježdene rekurzivne strukture objekata koja podsjećaju na stablo.

U našem projektu je moguće primijeniti ovaj patern na klasu Cart. Ta klasa je kontejnerske koja posjeduje Products, a iz te klase su izvedene klase Shoes, Clothing i Accessories pa je moguće u interface izdvojiti metodu za računanje cijene ili uklanjanje iz korpe (jer je to isto bez obzira na vrstu Product-a). Ako gledamo donji grafik Composite klasa bi bila naša klasa Cart . Leaf su pojedinačni elementi što predstavlja Product klasu. Component interface bi sadržavao metode za računanje cijene ili uklananje iz korpe, a Client klasa bi mogla biti neka klasa za obračun finalne (ukupne) cijene korpe.



PROXY PATERN

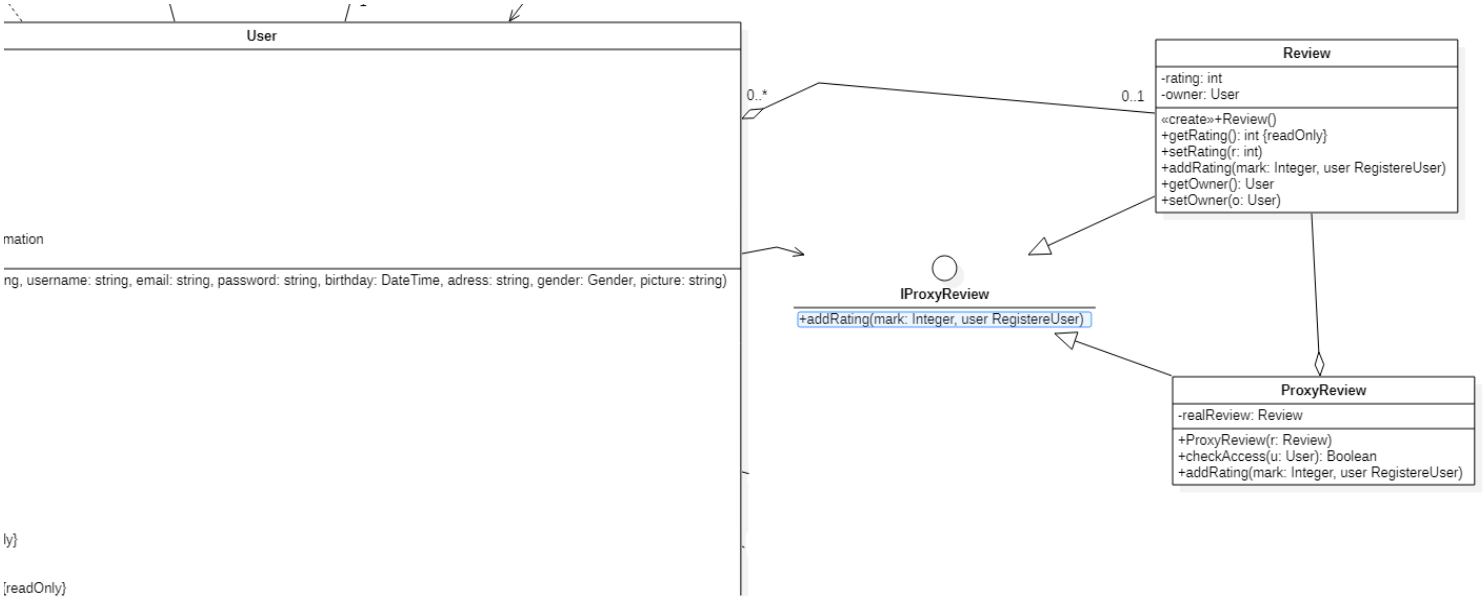
Proxy patern služi za dodatno osiguravanje objekata od pogrešne ili zlonamjerne upotrebe. Primjenom ovog paterna omogućava se kontrola pristupa objektima, te se onemogućava manipulacija objektima ukoliko neki uslov nije ispunjen, odnosno ukoliko korisnik nema prava pristupa traženom objektu.

Primjena u praksi:

Zaštita pristupa resursima

Ubrzavanje pristupa korištenjem cache objekata

Mi ćemo se u svojoj aplikaciji ograničiti na zaštitu pristupa resursima. U sistemu postoje klase User i Review. Recenzije želimo da ograničimo tako da je moguće ocijeniti samo User-a sa kojim je kupac poslovao. Za tu akciju potrebno je dodati interface IProxyRecenzija,koji sadrži metodu za ocjenjivanje, i klasu ProxyReview koja će provjeriti da li je moguće dodijeliti recenziju odabranom korisniku.



FLYWEIGHT PATERN

Flyweight patern koristi se kako bi se onemogućilo bespotrebno stvaranje velikog broja instanci objekata koji svi u suštini predstavljaju jedan objekat. Samo ukoliko postoji potreba za kreiranjem specifičnog objekta sa jedinstvenim karakteristikama (tzv. specifično stanje), vrši se njegova instantacija, dok se u svim ostalim slučajevima koristi postojeća opća instanca objekta (tzv. bezlično stanje).

Flyweight je dizajn patern koji omogućuje da "stavimo" više objekata u dostupnu količinu RAM memorije dijeljenjem zajedničkih dijelova između više objekata, umjesto da čuvate sve podatke u svakom objektu. Dakle, ideja je da se iskoristi ono što je zajedničko za više objekata i izdvoji gdje će biti dostupno za korištenje svim objektima koji su sačinjeni od tih elemenata (atributa). Pošto ovaj patern čini kod nečitljivim mi ga nećemo implementirati u svom projektu, al on bi se dobro morao iskoristi npr za klasu gost gdje bi uvijek imali samo jednu instancu tog objekta, ili bi se moglo izdvojiti neki atributi iz Cloatnig, Accessories i Shoes klase koji bi bili svima zajednički (npr atribut boja se može ovako izdvojiti jer je zajednička za sve 3 klase).