Strukturalni paterni

1. Adapter pattern

Adapter patern se koristi kada je potreban drugačiji interfejs već postojeće klase, a ne želimo mijenjati postojeću klasu. U našem sistemu, mogli bismo ga koristiti kod integracije eksternog sistema za lokaciju (npr. Google Maps, OpenStreetMap...) koji koristi drugačiji interfejs nego što naša klasa Lokacija očekuje. U našem slučaju klasa Lokacija ima metodu:

`dajKoordinate(): Pair<Float, Float>`

Eksterni sistem koristi metodu **`getCoordinates(city, street)`** koja vraća **`(lat, long)`**.

Rješenje je napraviti LokacijaAdapter koji implementira naš interfejs i poziva metodu eksternog sistema.

2. Façade pattern

Façade patern osigurava više pogleda (interfejsa) visokog nivoa na podsisteme čija je implementacija skrivena od korisnika, tj. iza scene se pozivaju svi potrebni sistemi. U našem sistemu ga možemo koristiti za upravljanje podacima o nekretnini. Da bi se ažurirala neka nekretnina, klijent mora da pozove više različitih metoda: jednu za promjenu lokacije, drugu za promjenu opisa, treću za promjenu cijene itd. Façade patern nudi rješenje za ovaj problem. Kreira se **`NekretninaFacade`** koja predstavlja definciju i implementaciju jedinstvenog interfejsa za skup operacija podsistema.

Ta klasa će sadržavati metodu **`azurirajNekretninu()`,** koju klijent poziva bez potrebe da zna sve pojedinačne korake. Potrebna je interakcija sa eksternim servisima jer se nekretnina veže za više lokacija i servisa. Još jedan primjer upotrebe ovog paterna unutar našeg sistema je interakcija sa eksternim sistemima. Da bi jedna nekretnina bila objavljena, potrebna je komunikacija sa dodatnim servisima kao što su **`EmailService`**, **`GeoLocationService`**, **`NotificationService`** i slični. Potrebno je napraviti klasu koju ćemo nazvati recimo **`PublikacijaFacade`**. Ta klasa kao atribute sadrži objekte već pomenutih podsistema (servisi). Također, sadrži i metode koje su sastavni dio podsistema, tako klijent prilikom objave nekretnine ne mora znati redoslijed poziva niti koji se servisi koriste – sve je namješteno u façade klasi.

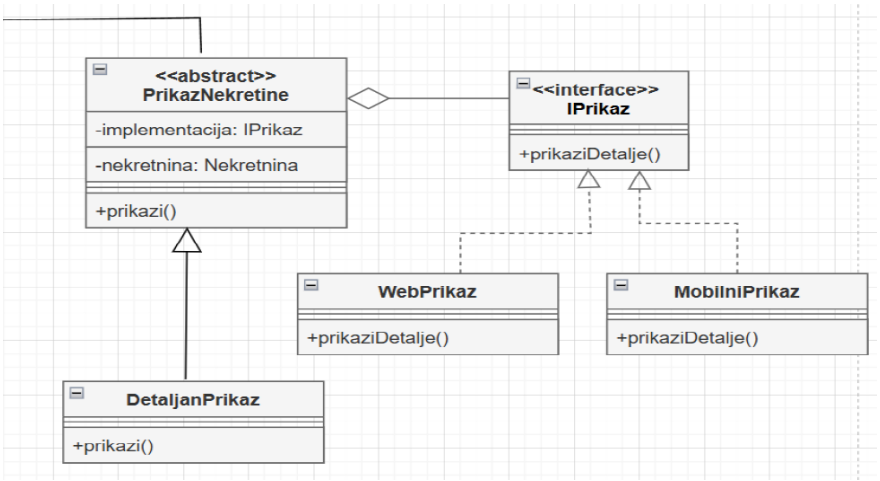
3. Decorator pattern

Decorator patern omogućava dinamičko dodavanje novih elemenata i ponašanja (funkcionalnosti) postojećim objektima, bez mijenjanja postojeće klase. U našem sistemu ovaj patern možemo iskoristiti za slučaj da želimo istaknuti dodatno neke nekretnine prema npr. boji, prioritetu, oznaci, zatim ukoliko želimo naglasiti da neke nekretnine imaju snimljene video ture, ili recimo ako neke nekretnine imaju sponzorisani status. Sva ta dodatna ponašanja možemo dodati bez izmjene osnovne Neretnina klase.

Imamo interfejs **`INekretnina`** koji identificira klase objekata koje trebaju dekoraciju. Nekretnina je originalna klasa koja sadrži nepromjenjivi interfejs. Kreirat ćemo klasu **`NekretninaDecorator`** koja odgovara **`INekretnina`** interfejsu i implementira dinamički proširen interfejs. Na osnovu te klase možemo kreirati konkretne decorator klase, u našem slučaju ih možemo nazvati **`Isticanje, VerifikovanaNekretnina, SponzorisanaNekretnina`**…

4. Bridge pattern

Bridge patern omogućava odvajanje apstrakcije i implementacije neke klase tako da klasa može posjedovati više različitih apstrakcija i više različitih implementacija za pojedine apstrakcije. U našem sistemu ovaj patern ćemo iskoristiti za način prikaza nekretnina na različitim platformama. Klasa PrikazNekretnine je interfejs apstrakcije koji klijent vidi. PrikazNekretnine je apstraktna klasa koja sadrži referencu na objekat tipa IPrikaz. DetaljanPrikaz nasljeđuje PrikazNekretnine i može da pozove metode implementacija.prikaziDetalje(). IPrikaz je interfejs koji definiše metodu prikaziDetalje(). Ovaj interfejs je zadužen za sve konkretne implementacije. WebPrikaz i MobilniPrikaz su konkretne klase koje implementiraju IPrikaz, i one sadrže specifični kod ovisan od platforme.



5. Proxy pattern

Proxy patern se koristi kada je potrebno kontrolisati pristup nekom objektu recimo zbog pristupnih prava, keširanja, autentifikacije i slično. U našem sistemu moguće ga je iskoristiti prilikom pristupa klasi **`NekretninaService`** – npr. samo administratori imaju pravo da mijenjaju nekretnine. Koristi se na način da klijent poziva metodu **`promijeniOpis()`** isključivo u slučaju ako je administrator.

Rješenje je napraviti klasu **`NekretninaServiceProxy`** koji implementira isti interfejs kao **`NekretninaService`**, ali prije svake operacije provjerava dozvole, a zatim poziva originalnu klasu.

6. Composite pattern

Composite patern omogućava formiranje strukture drveta pomoću klasa, u kojoj se listovi stable i korijeni jednako tretiraju. Tačnije, koristimo ga ukoliko želimo da radimo sa

hijerarhijom objekata na isti način, bilo da je objekat list ili korijen. U našem sistemu moguće je iskoristiti ovaj patern prilikom grupisanja više nekretnina u “pakete”.

U slučaju da investitor želi da prodaje 5 stanova zajedno ili da agencija prodaje više kuća kao jednu cjelinu, composite pattern je idealno rješenje. Kreiramo interfejs **`INekretninaComponent`** sa metodama poput **`prikaziDetalje(). Nekretnina`** predstavlja pojedinačnu nekretninu. **`GrupaNekretnina`** sadrži listu nekretnina i omogućava operacije nad grupom. Pojedinačne nekretnine i grupe nekretnina se tretiraju na isti način — pozivom iste metode na interfejsu.