STRUKTURALNI PATERNI

Strukturalne paterne čine: Facade, Decorator, Adapter, Bridge, Composite i Proxy.

Strukturalni paterni se bave kompozicijom i predstavljaju načine za definisanje odnosa među objektima. Strukturalni paterni daju mogućnost da kada je potrebna promjena u jednom dijelu sistema da se ne mora ostatak sistema mijenjati.

Adapter pattern

Osnovna namjena Adapter pattern-a je da omogući širu upotrebu već postojećih klasa. Kada je potreban drugačiji interfejs postojeće klase, a ne želimo da mijenjamo postojeću klasu koristimo Adapter pattern. Tada se kreira nova Adapter klasa koja se koristi kao posrednik između originalne klase i interfejsa.

Primjena u sistemu:

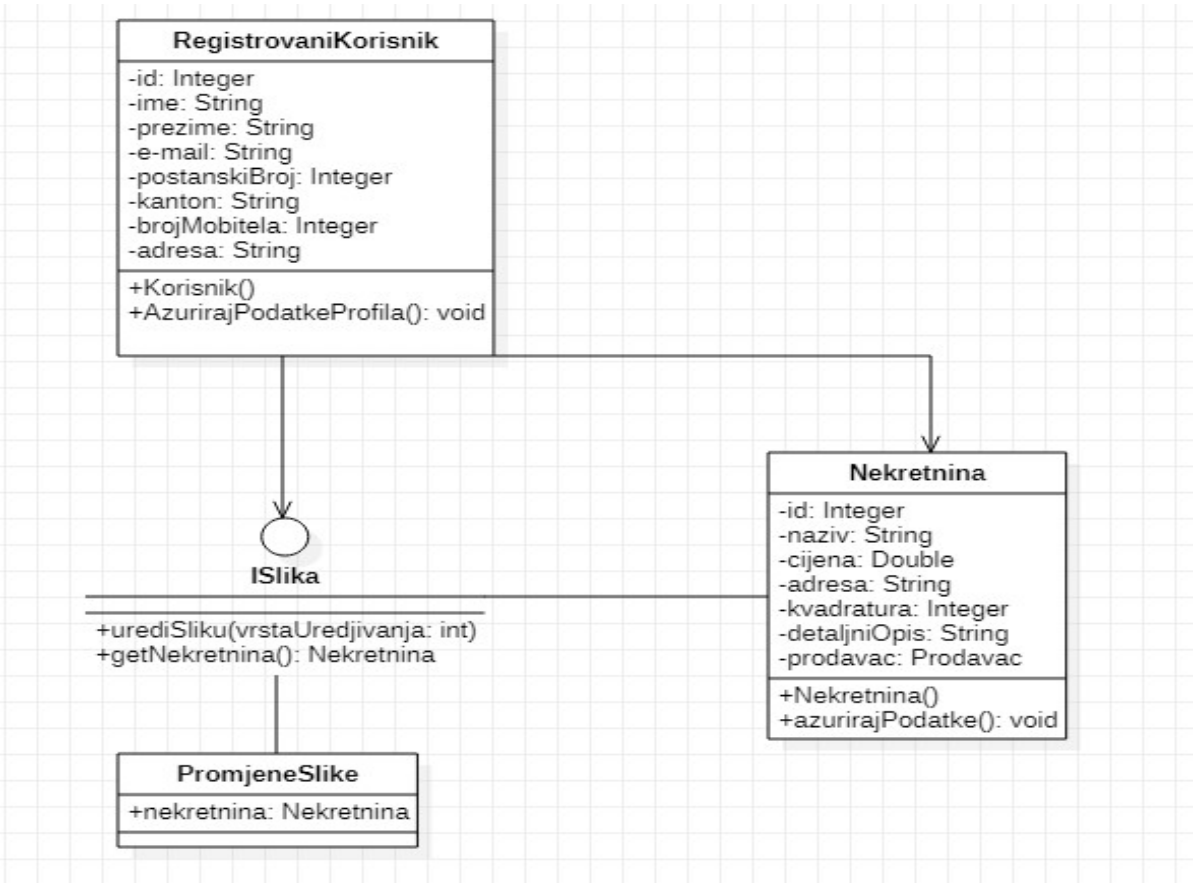
U našem sistemu Adapter patern bi mogli iskoristiti za slike. U našem sistemu trenutni tip atributa Slika je string. Taj tip bi mogli zamijeniti uz pomoć metoda konverzija Adapter klase i omogućiti prihvatanje raznih tipova formata slike.

Decorator pattern

Osnovna namjena Decorator pattern-a je da omogući dinamičko dodavanje novih elemenata i funkcionalnosti postojećim objektima. Objekat pri tome ne zna da je urađena dekoracija što je veoma korisno za ponovnu upotrebu komponenti softverskog sistema.

Primjena u sistemu:

U našem sistemu mogli bi dodati mogućnost editovanja slika kao što su rezanje, rotacija, uređivanje i povezivanje. To bi uradili na način kada bi dodali interfejs ISlika koji bi imao metode za uređivanje slike.



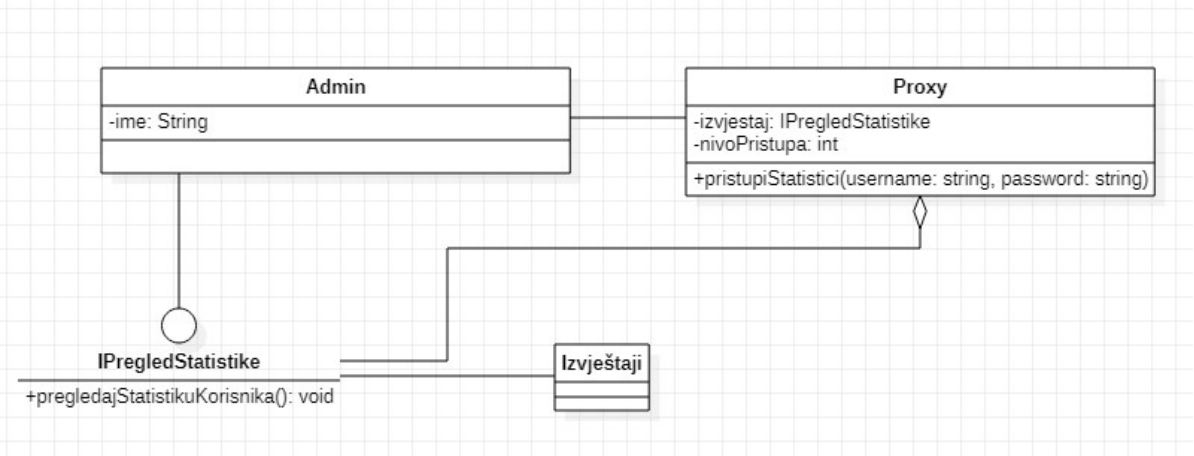
U sistemu imamo klasu Nekretnina koja prikazuje osnovne informacije o objavljenoj nekretnini. Ukoliko u budućnosti dođe do potrebe za proširivanjem klase Nekretnina dodavanjem novih atributa ili metoda, tada ne bi mijenjali tu klasu već bi napravili novu klasu Decorator i interfejs IDecorator u koje bi dodavale nove metode ili atribute koje želimo implementirati.

Proxy pattern

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 sistemu:

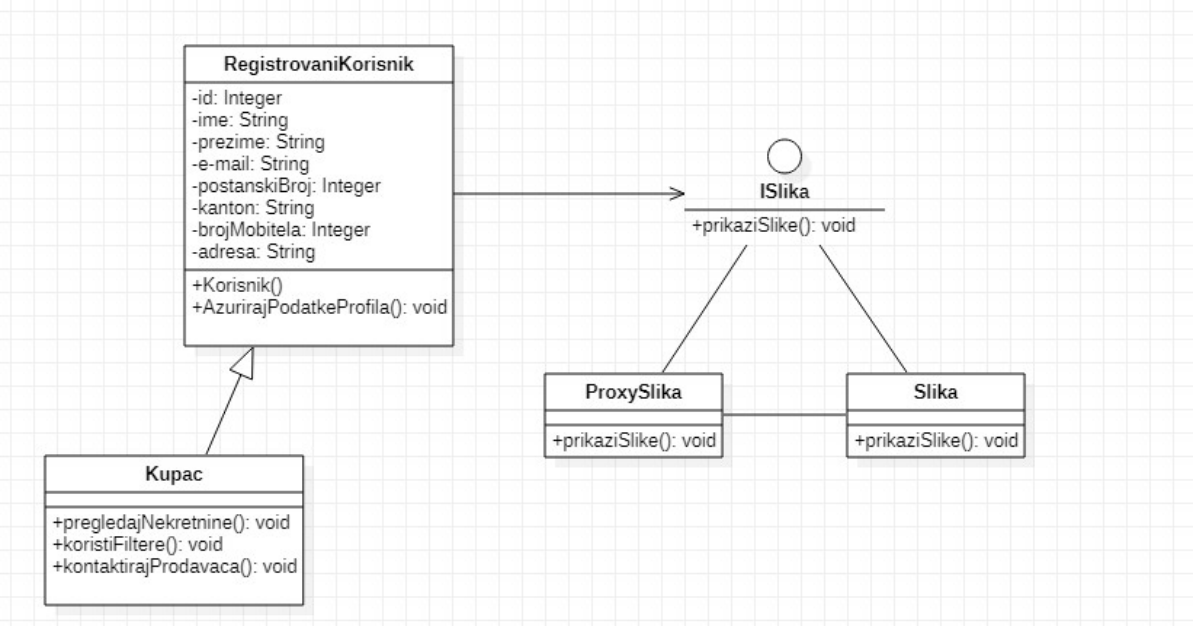
Proxy pattern u sistemu možemo još iskoristiti za pregled izvještaja koji već imamo u sistemu. Možemo ga još osigurati preko proxy paterna tako što bi ograničili pravo pristupa za pregled izvještaja.



Proxy pattern u aplikaciji još možemo iskoristi npr. kada korisnik želi da iznajmi nekretninu, ali tada će svaki korisnik imati pristup iznajmljivanja nekretnine. Tada ćemo imati autentifikacijski proxy koji će proslijeđivati neki password adminu koji samo on zna.

Proxy pattern možemo još iskoristiti i na način da kreiramo Proxy klasu koja će izvršiti kreiranje instance klase Admin, tj. klasa Admin treba biti privatna. Metoda za autentifikaciju će se pozvati samo kada su uneseni ispravni korisnički podaci.

Jedan od najvažnijih dijelova sistema je prikazivanje slika nekretnine. Da bi imali veliki kvalitet slike možemo preko Proxy patterna omogućiti veliku rezoluciju slike. To bi implementirali na način da kreiramo interfejs ISlika koji će imati metodu za prikaz slike i dvije klase Slika i ProxySlika.



Composite pattern

Composite pattern ima namjenu da omogući formiranje strukture stabla pomoću klasa u kojoj se individualni objekti i kompozicije individualni objekata jednako tretiraju. Koristi se za kreiranje hijerahije objekata, tj. Kada svi objeti imaju različite implementacije nekih metoda kojima je potrebno pristupati na isti način.

Primjena u sistemu:

U sistemu imamo više metoda koje služe za ažuriranje podataka nekretnine ili korisnika. Tada bi napravili interfejs IComposite koji bi imao metodu azuriranje() koje bi služilo za ažuriranje svih klasa koje imamo u sistemu.

Facade pattern

Facade pattern služi kako bi se klijentima pojednostavilo korištenje kompleksnih sistema. Klijenti vide samo krajnji izgled objekta, dok je njegova unutrašnja struktrura skrivena. Na ovaj način se smanjuje mogućnost pojavljivanja grešaka jer klijenti ne moraju dobro poznavati sistem da ga koriste.

Primjena u sistemu:

Ovaj patern bi mogli iskoristiti na način da napravimo klasu Facade koja bi bila fasadna klasa. Ta klasa bi sadržavala druge klase kao atribute te klase. Pozivom neke metode ove klase će biti pozvano više različitih medota iz drugih klasa koje korisnik ne može vidjeti.

Flyweight pattern

Flyweight patern se koristi 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, vrši se njegova instantacija, dok se u svim ostalim slučajevima koristi postojeća opća instance objekta.

Primjena u sistemu:

Ovaj patern bi mogli iskoristiti na način da ukoliko se Gost prijavljuje više puta, umjesto instanciranja više objekata tog tipa, biti će instanciran jedan objekat tog tipa.

Bridge pattern

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 je pogodan kada se implementira nova verzija softvera, a postojeća mora ostati u funkciji.

Primjena u sistemu:

Ovaj pattern se može iskoristiti za sisteme za razmjenu poruka. Kada bi u sistemu napravili da korisnici komuniciraju međusobno, tada bi morali dodati klasu Abstraction koja će imati svoje metode i interfejs Implementation koji će raditi zajedno sa klasom Abstraction.