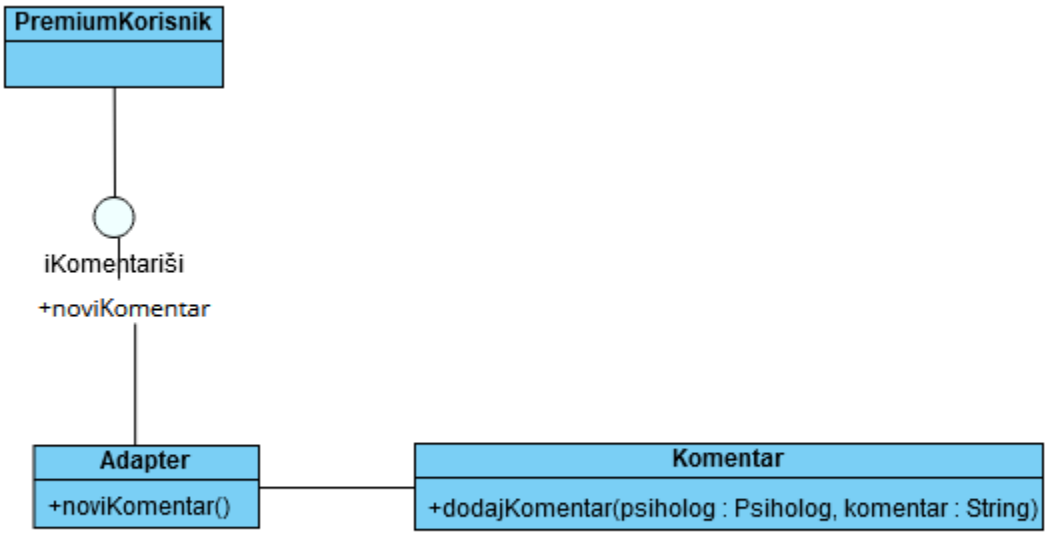
1. Adapter

Adapter patern kreira novu adapter klasu koja služi kao posrednik između originalne klase i željenog interfejsa. Tim postupkom se dobija željena funkcionalnost bez izmjena na originalnoj klasi i bez ugrožavanja integriteta cijele aplikacije.

Naš sistem trenutno nema potrebu za implementacijom adapter strukturalnog patterna, ali ako bismo ga željeli dodati, ne bismo morali mijenjati klase već bismo napravili adaptersku klasu.

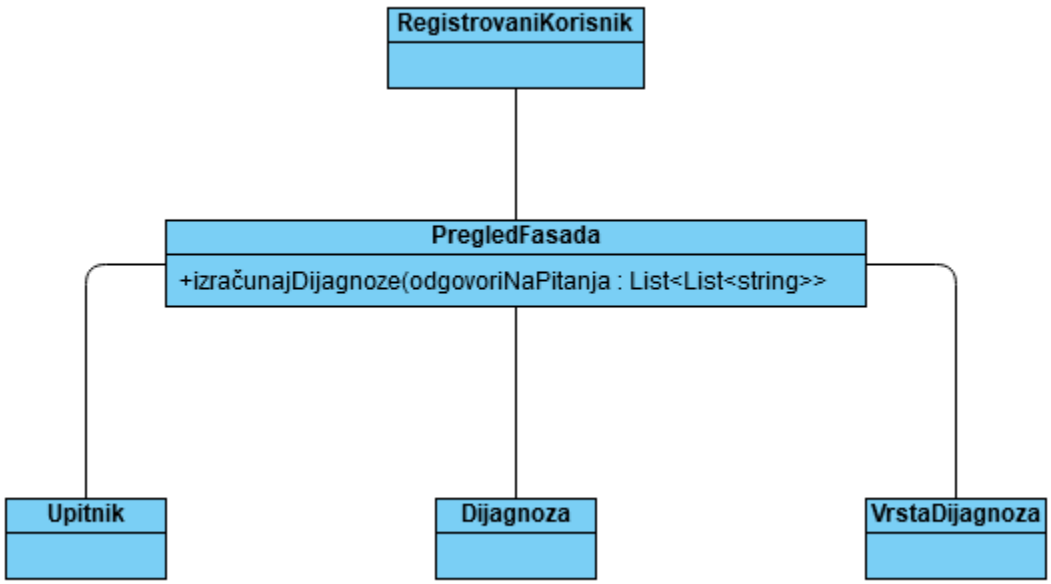
Ukoliko bismo željeli dodati funkcionalnost da PremiumKorisnik može, uz ocjenu, dodavati i komentare nakon termina, implementirali bismo Adapter Pattern



2. Fasadni pattern

Fasadni pattern se implementira kada želimo koristiti mali dio funkcionalnosti velikog i kompleksnog sistema, ili kada nam način implementacije (funkcioniranja) neke funkcionalnosti nije bitan (sa aspekta korisnika).

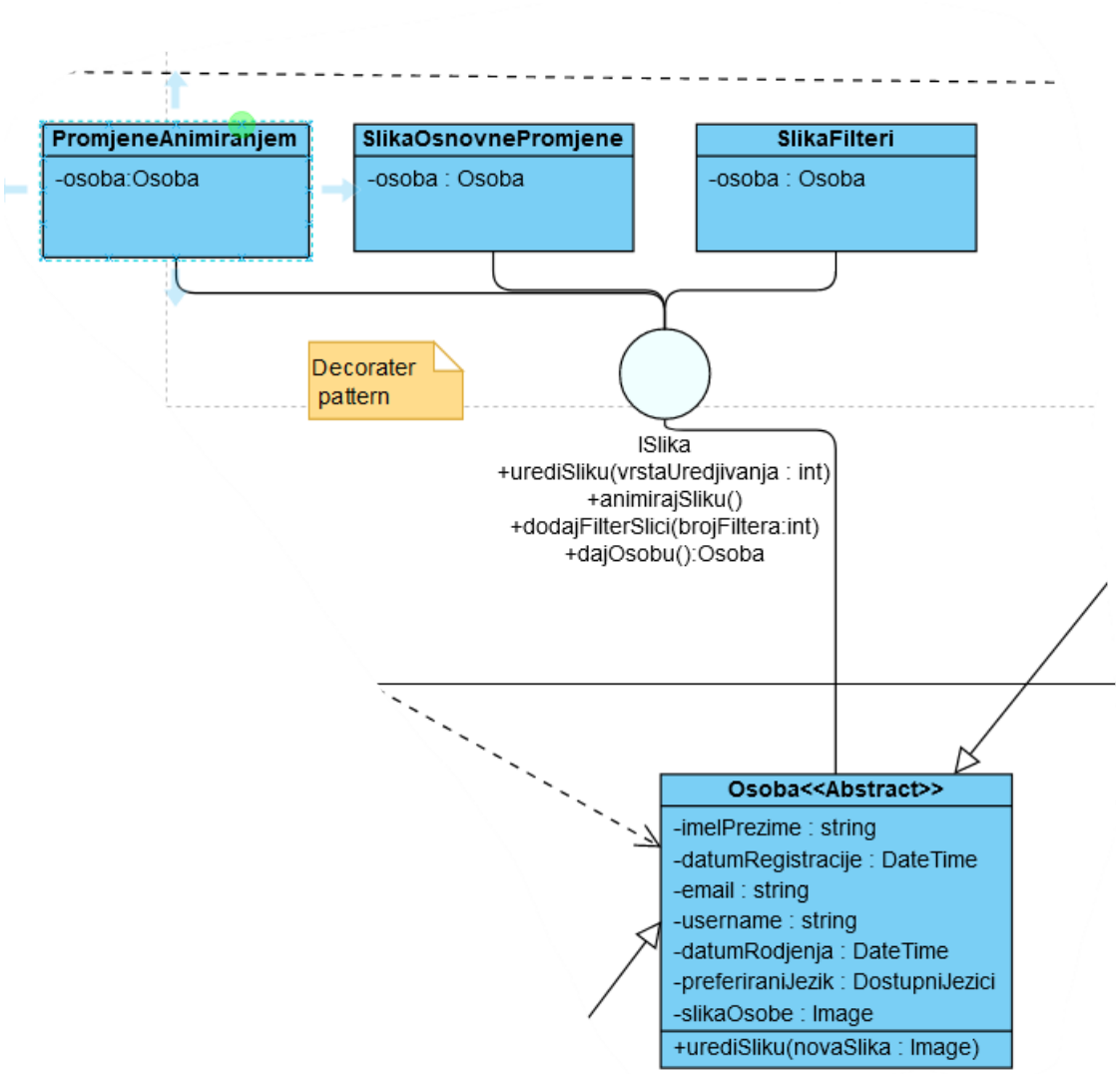
U našem sistemu nije predviđena potreba za implementacijom ovog patterna, ali ukoliko bismo na primjer željeli da pregled dijagnoza nakon popunjenog upitnika implementiramo putem ovog patterna, mogli bismo to izvršiti na sljedeći način, pri čemu bi se sam princip izračunavanja dijagnoza i korištenja više klasa sakrio od korisnika, jer mu je najbitniji finalni produkt a ne njegova izvedba.



3. Dekorater pattern

Dekorater pattern se koristi kada ne želimo praviti veliki broj podklasa koje predstavljaju kombinacije već postojećih klasa, već želimo iskoristiti te postojeće klase.

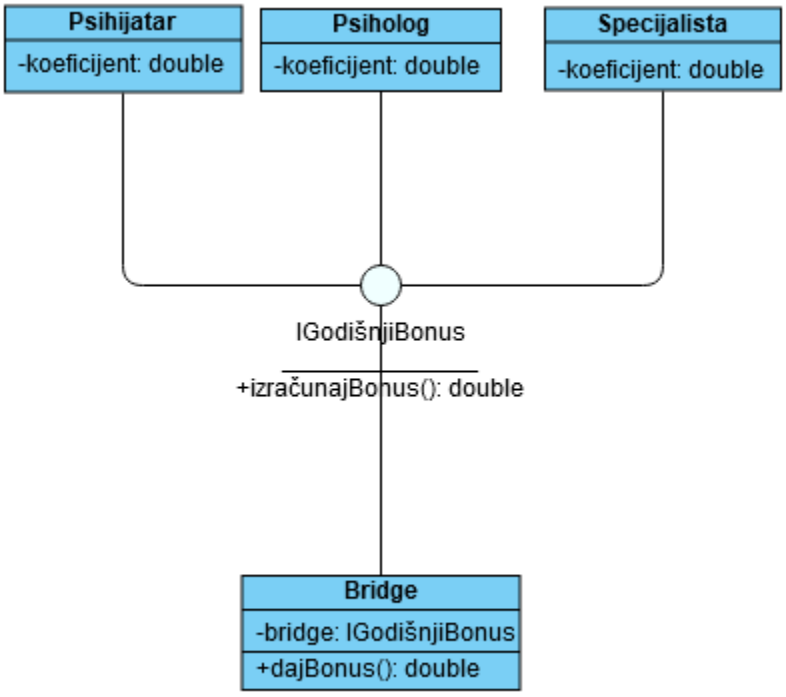
Dekorater pattern je već implementiran u naš sistem. Uočili smo potrebu za implementacijom ovog patterna kod dodavanja i uređivanja korisničkih slika. Predvidjeli smo mogućnost više načina uređivanja slika pa smo zato implementirali interfejs ISlika.



4. Bridge pattern

Bridge pattern omogućava da se iste operacije primjenjuju nad različitim podklasama. Također se koristi kod izbjegavanja kreiranja novih metoda za postojeće funkcionalnosti.

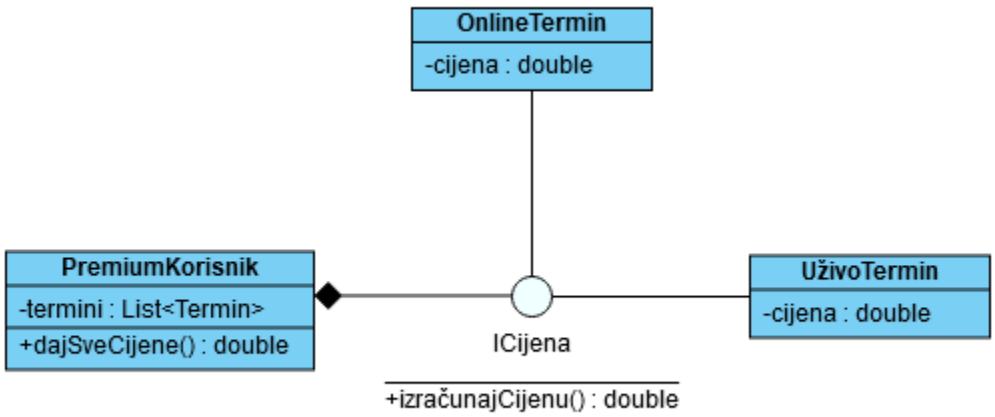
Što se tiče našeg sistema, ukoliko bi se javila potreba za uvođenjem dodatnih zaposlenika sistema, način računanja bonusa bismo mogli izvesti putem Bridge patterna:



5. Kompozitni pattern

Kompozitni pattern se koristi za pravljenje hijerarhije klasa, ili za omogućavanje pozivanja iste metode nad različitim objektima sa različitim implementacijama.

Ovaj pattern bi se u našem sistemu mogao implementirati na način da uvedemo interfejs koji bi vraćao cijenu termina, a kojeg bi implementirale klase Termin i PremiumKorisnik. Ako bi željeli uvesti novi način održavanja psihoterapeutskih sesija, cijene bi se razlikovale u zavisnosti od načina održavanja. Očigledno je da će implementacije metode tog interfejsa biti različite, jer za klasu PremiumKorisnik se mora vratiti ukupna cijena svih termina koje je korisnik imao.



6. Proxy pattern

Proxy pattern se koristi kod zaštite pristupa resursima u sistemu. On također ubrzava pristup korištenjem cache objekata.

Ovaj pattern je već implementiran u našem sistemu. Izveden je zbog kontrolisanja pristupa prethodnim terapijama korisnika. Ukoliko psihoterapeut nije ranije komunicirao sa određenim korisnikom, ne može imati uvid i uređivati njegove terapije.

7. Flyweight

Flyweight pattern se koristi kod ponovne upotrebe istih objekata, a u cilju smanjivanja RAM potrošnje.

Ovaj pattern nije predviđen za implementaciju u našem sistemu, ali bi se mogao izvesti na sljedeći način. Ukoliko novi korisnik, koji tek popunjava upitnik vezan za dijagnoze, da iste odgovore kao neki korisnik koji je prije njega popunio taj upitnik, program neće ponovo računati procentualni spisak mogućih dijagnoza, nego će korisniku proslijediti spisak dijagnoza od prethodnog korisnika, jer su im odgovori identični.