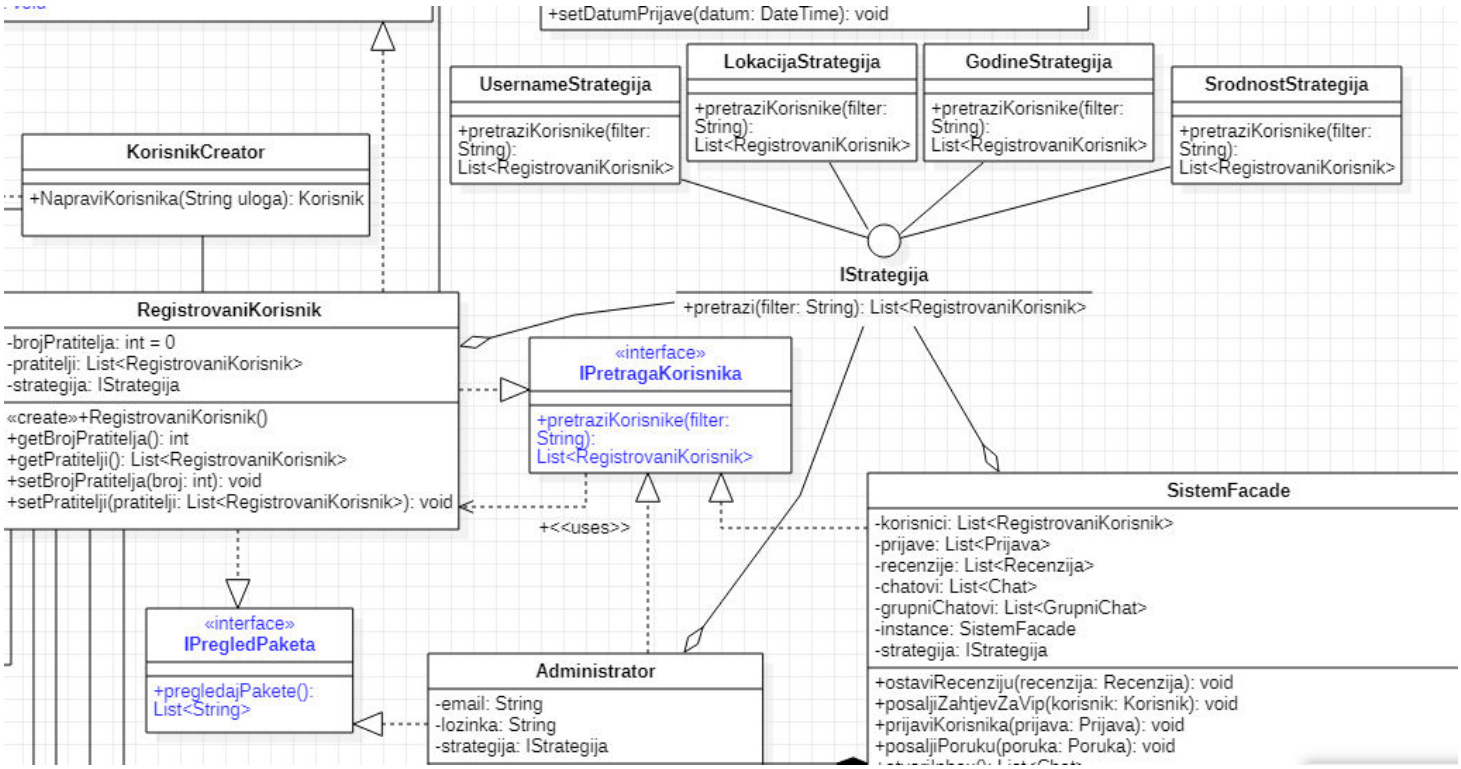
PATERNI PONAŠANJA

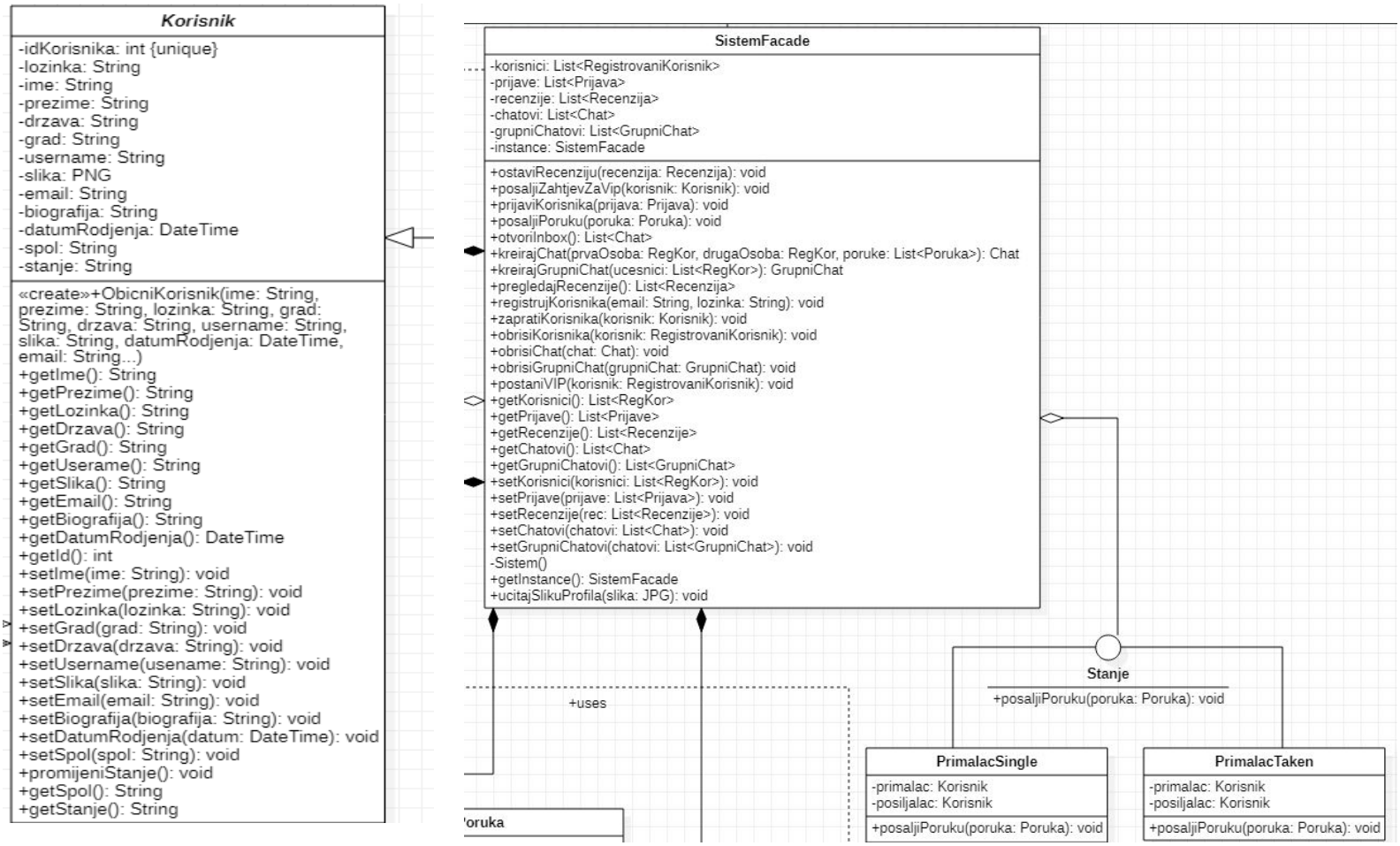
1. *Strategy*

Ovaj patern ćemo iskoristiti kod pretrage korisnika s obzirom da korisnik može imati različite zahtjeve prilikom pretraživanja. Jedan zahtjev može biti prosto pronalaženje korisnika na osnovu željenog username-a ili imena i prezimena. Drugi zahtjev može biti pronalaženje korisnika u blizini odnosno korisnik bi u tom slučaju izabrao u kojem opsegu od njegove trenutne lokacije želi da pronalazi osobe te bi algoritam vratio osobe sortirane od onih sa najbližom lokacijom do onih sa najdaljom. Sljedeći zahtjev bi bio pretraga na osnovu godina gdje korisnik zahtjeva pretragu korisnika sa određenim opsegom njihovih godina. Posljednji zahtjev, kao najgeneralizovaniji, bio bi pronalaženje srodnih osoba. U tom slučaju, algoritmu bi bilo prepušteno da samostalno na osnovu raznih informacija o korisniku izabere one koji bi najviše odgovarali spomenutom korisniku. Za početak bi bili implementirani algoritmi za ove vrste zahtjeva, a svakako da će se patern po potrebi moći nadograđivati sa mnogo više algoritama, npr. pretraga na osnovu na osnovu posla kojim se bavi korisnik, na osnovu obrazovanja i slično. Patern bi bio implementiran na način da se u klasama RegistrovaniKorisnik, Administrator i Sistem bira strategija u okviru atributa „strategija“ jer sve 3 klase realizuju interfejs IPretragaKorisnika. Osim toga dodat će se i interfejs IStrategija kojeg će implementirati sve klase postojećih strategija što se najbolje može prikazati na dijagramu:



2. *State*

Ovaj patern ćemo iskoristiti u našem projektu s obzirom da se objekat tipa Korisnik može nalaziti u stanjima slobodan i zauzet. Pri slanju poruka želimo da zabranimo slanje poruka zauzetim osobama suprotnog spola. Za implementaciju ovog paterna, najprije je potrebno dodati u klasu Korisnik atribute spol i stanje. Potrebno je dodati i metodu promijeniStanje. Dalje, interfejs Stanje sa metodom posaljiPoruku iz klase Sistem implementiraju klase PrimalacSingle i PrimalacTaken. Ako se pozove metoda posaljiPoruku, provjerava se atribut „primalac“ iz klase Poruke odnosno njegovo stanje i atribut „posiljalac“. U slučaju da je stanje „slobodan“, metoda će poslati poruku primaocu. U slučaju da je stanje „zauzet“, bit će pozvana metoda implementirana u klasi PrimalacTaken gdje će se provjeravati spolovi primaoca i pošiljaoca. Ako su im isti spolovi, bit će poslana poruka, u suprotnom neće.



3. ***Template Method***

   Ovaj patern nećemo implementirati ali bi se mogao primijeniti na način da se ostvari jednostavno pronalaženje najboljeg match-a za datog korisnika. Za implementaciju ovog paterna bit će nam potrebna apstraktna klasa PerfectMatch sa metodama templateMethod, filtrirajOpcenito, filtrirajPoInteresima, odaberiTop3. U ovisnosti od toga kakav je status veze korisnika, ove metode će biti različito implementirane. Za konkretan status bit će primijenjena konkretna klasa od 3 moguće: Single, Taken, Divorced. U klasi Single, metoda filtrirajOpcenito bi prvenstveno izdvojila ljude koji nisu u vezi, koji se nisu nikad ni vjenčavali i koji su približnih godina kao korisnik za kojeg se traži najbolji match. Metoda filtrirajPoInteresima bi izdvojila ljude čija se pitanja u vezi braka, religijska, politička uvjerenja slažu sa korisnikom koji se razmatra. I na kraju metoda odaberiTop3 bi izabrala one 3 osobe koje se najviše poklapaju sa datim korisnikom u smislu raznih interesa kao što su hobiji, sportska aktivnost, muzika koju slušaju, filmovi i serije koje gledaju itd. U klasi Taken bi metoda filtrirajOpcenito izdvojila ljude istog spola koji također traže prijatelje. Metoda filtrirajPoInteresima bi izdvojila ljude koji imaju slične poslove, obrazovanje i slično. I odaberiTop3 bi bila ista kao i u klasi Single. Kad je u pitanju Divorced klasa, metoda filtrirajOpcenito bi izdvojila ljude koji su također razvedeni. Metode filtrirajPoInteresima i odaberiTop3 bi bile iste kao i u klasi Single.

4. ***Observer***

   Ovaj patern nećemo koristiti ali bismo ga mogli iskoristiti ukoliko bismo nadogradili našu aplikaciju sa mogućnošću objavljivanja postova ili fotografija od strane korisnika. U tom slučaju, svi oni korisnici koji zaprate korisnika A, mogli bi na jednostavan način dobijati obavještenje kad god korisnik A objavi novi post ili fotografiju. Uvodimo nove klase Subject klasu i Observer klasu. Instanca klase Subject mijenja svoje stanje i obavještava Observer klase koje predstavljaju konkretne klase koje obezbjeđuju konkretnu implementaciju za IObserver interfejs(sadrži samo jednu metodu - Update koja se poziva nakon promjene stanja neke Subject instance). Subject klasa sadrži privatni događaj Notify(Event mehanizam za pozivanje Update metode za sve posmatrače). Dakle, Notify će prolaziti kroz listu svih pratitelja određenog korisnika odnosno potrebno je u klasu Korisnik dodati tu listu kao atribut da bi to bilo moguće. Subject bi bio korisnik koji objavljuje postove i ima pratitelje koji žele da dobiju obavještenje pri objavljivaju svakog novog posta, a Observer su upravo ti pratitelji.

5. ***Iterator***

   Iterator patern omogućava sekvencijalni pristup elementima kolekcije bez poznavanja strukture kolekcije. U našem projektu nije potreban ali bismo ga mogli implementirati na sljedeći način. S obzirom da klasa Chat sadrži listu svih

poruka između dvije osobe, zahvaljujući ovom paternu, mogli bismo omogućiti iteriranje foreach petljom kroz ovu listu tako da prvi element bude posljednja poslana poruka u chatu, drugi element pretposljednja itd. (morali bismo u klasu Poruka dodati i atribut koji će predstavljati datumi vrijeme slanja poruke). U klasu Chat bismo dodali instancu klase Iterator koja će sadržavati trenutni indeks i broj elemenata kao atribute, a metoda koju će implementirati - getNext(). Također, klasa Chat bi implementirala interfejs IterableCollection koja sadrži metodu createIterator za kreiranje iteratora.