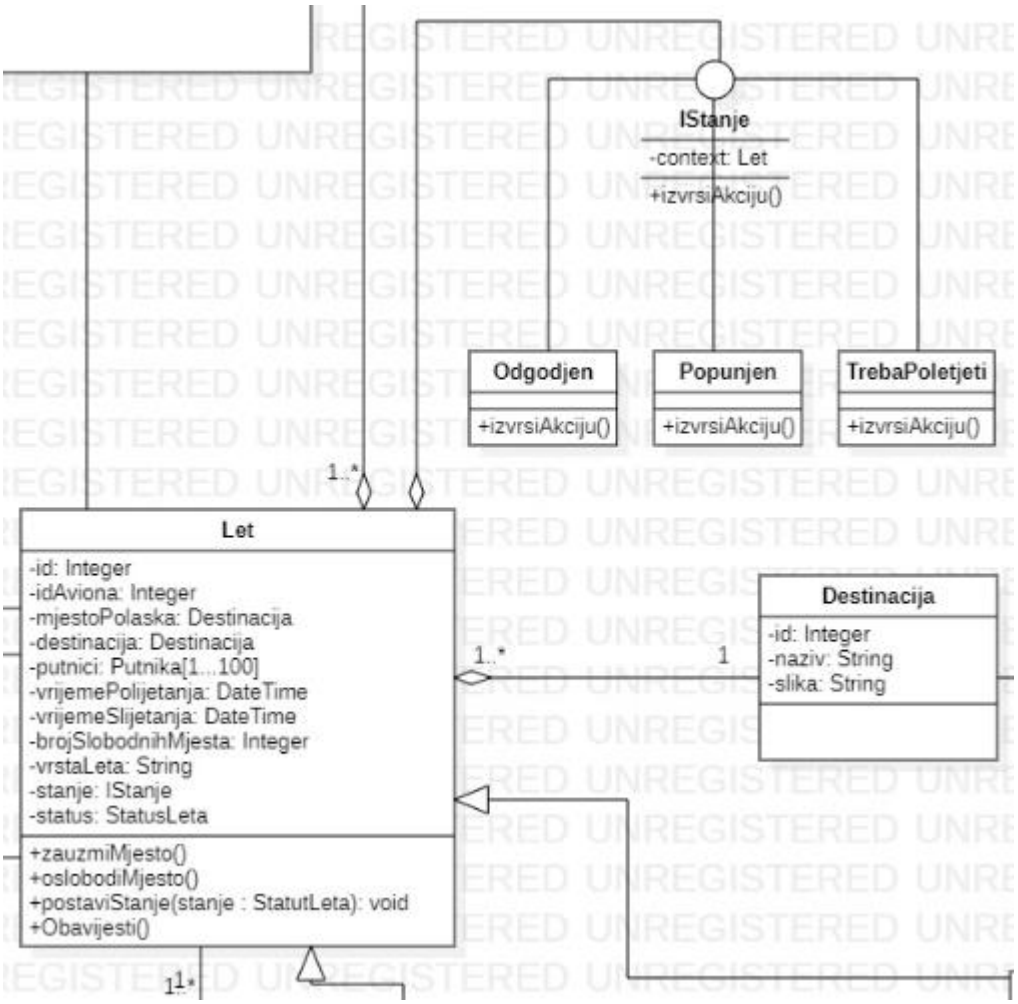
Paterni ponašanja

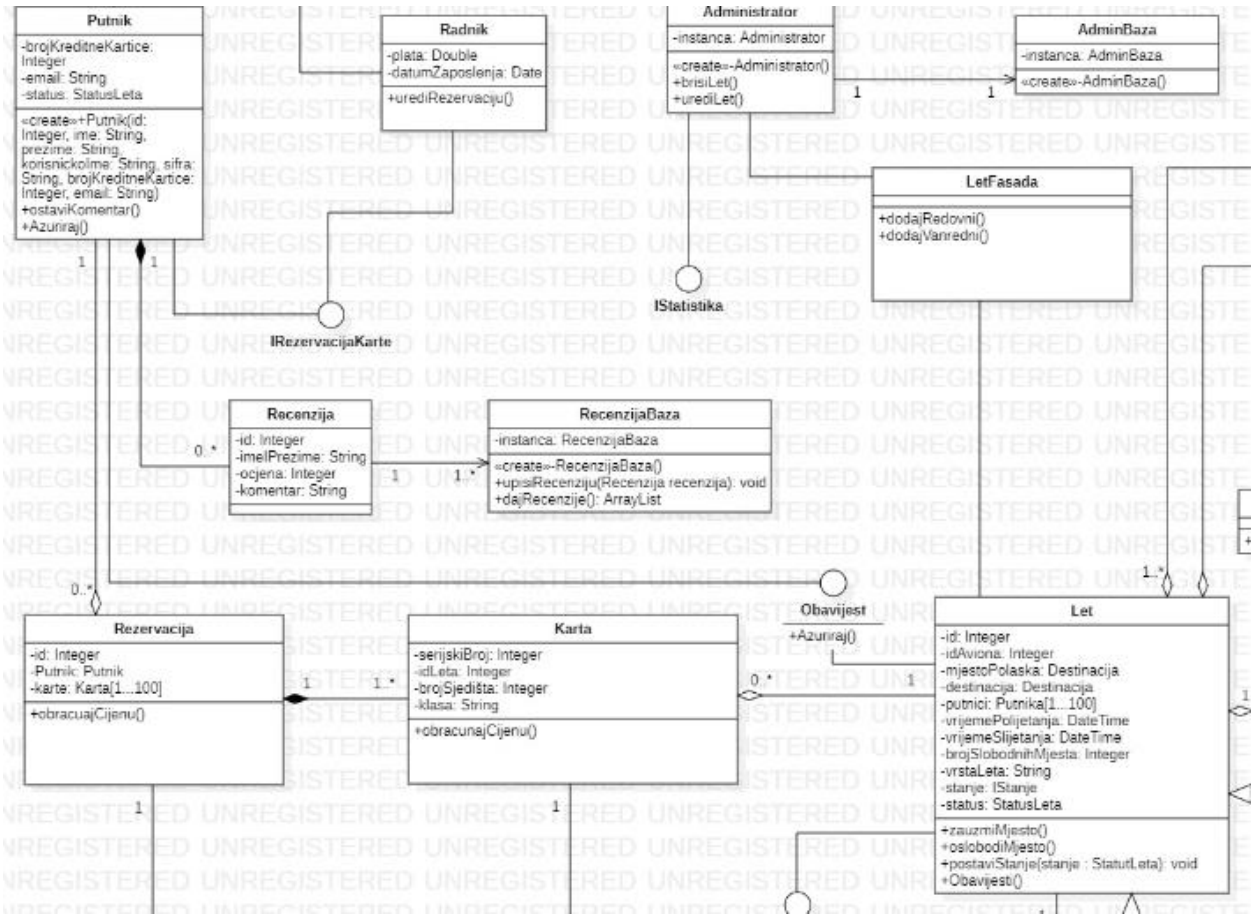
- **Iterator pattern** - ovaj pattern se koristi kada je potrebno imati uniforan način pristupa bilo kojoj kolekciji. Ako recimo želimo iz nekog razloga da primimo ArrayList, Array i HashMapu, možemo iskoristiti iterator interface pomoću kojeg ćemo najbolje omogućiti uniforan pristup, skratiti kod, napraviti bolji polimorfizam. Ukoliko imamo rad sa više vrsta kolekcija implementacija ovog paterna je veoma korisna. Pošto u našoj implementaciji imamo samo listu, upotreba ovog paterna nije od nekog značaja. Mogao bi se iskoristiti u nekoj posebnoj klasi koja bi imala realizovan Iterator interface i imala npr. metodu koja će raditi nešto sa elementima svih kolekcija. Svejedno, patern se mogao iskoristiti za prolazak kroz listu putnika unutar klase Let.

- **Strategy pattern** - Strategy patern izdvaja algoritam iz matične klase i uključuje ga u posebne klase. Strategy patern je koristan kada postoje različiti primjenjivi algoritmi (strategije) za neki problem. Ukoliko imamo ograničeno, ali obavezano ponašanje različitih podklasa u hijerarhiji, tada je ovaj patern zgodno iskoristiti. Strategy patern omogućava klijentu izbor jednog od algoritma iz familije algoritama za korištenje. Prednost ovog paterna je što se mogu definirati dodatne metode za nove strategije neovisno od ostalih klasa! Kako u našem sistemu trenutno nemamo procesa koji se mogu vršiti na više različitih načina, tj. nemamo više algoritama koji rješavaju neki problem, u naš sistem trenutno ne možemo implementirati ovaj patern. Međutim mogli bismo ga implementirati ukoliko bismo u našem sistemu omogućili korisniku prikaz letova sortiranih po određenim kategorijama (cijeni ili datumu polijetanja).

- **State pattern** - State pattern je dinamička verzija Strategy patterna gdje objekat mijenja način ponašanja na osnovu trenutnog stanja. Postiže se promjenom podklase unutar hijerarhije klasa. Ovaj pattern je jako koristan I iskoristili smog a u našem sistemu smo za određivanje stanja leta, tj da li je let popunjen, odgođen itd. Ovo bismo implementirali dodavanjem interfejsa IStanje sa metodom dajStanje(), koju bi implementirale klase Popunjen, Odgođen i TrebaPoletjeti.



- **Observer pattern** - Observer patern koristimo kada je potrebno uspostaviti relaciju između objekata tako da kada jedan objekat promijeni stanje, drugi zainteresirani objekti dobijaju obavijest o promjeni. Ovaj patern smo primjenili tako da za svaku promjenu stanja leta putnici dobivaju obavijest o tome. Implementirali smo ga dodavanjem interfejsa IObavijest sa metodom Azuriraj(). U klasi Putnik smo dodali metodu Azuriraj() koja azurira atribut za stanje leta.



- **TemplateMethod Pattern** - Omogućava izdvajanje određenih koraka algoritma u odvojene podklase. Struktura algoritma se ne mijenja - mali dijelovi operacija se izdvajaju i ti se dijelovi mogu implementirati različito. Ovaj patern nismo implementirali, ali bismo to mogli kada bi u našem sistemu postojale podjele registrovanog korisnika, na običnog korisnika i VIPKorisnika, kako bi regulisali različite načine plaćanja, s obzirom da bi VIPKorisnik imao neke pogodnosti.