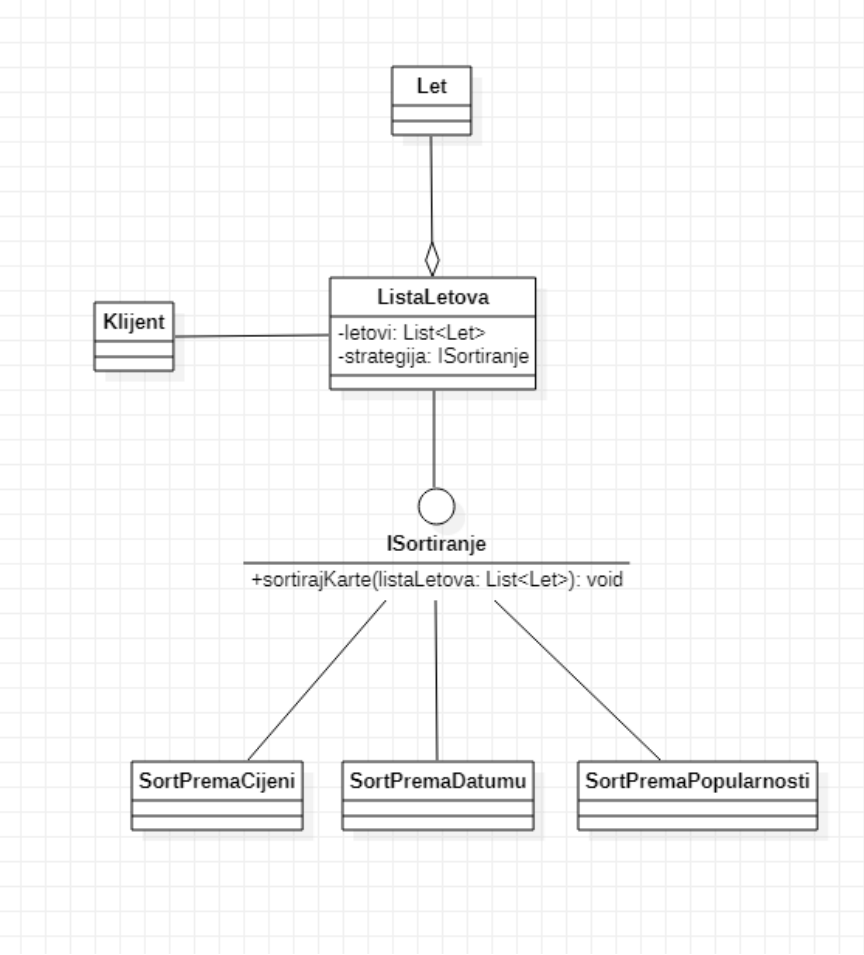
PATERNI PONASANJA

1) **Strategy patern**

Uloga **strategy paterna** jeste da izdvaja algoritam iz matične klase i uključuje ga u posebne klase. Pogodan je kada postoje različiti primjenjivi algoritmi (strategije) za neki problem.
Strategy patern omogućava klijentu izbor jednog od algoritma iz familije algoritama za korištenje. Algoritmi su neovisni od klijenata koji ih koriste.
Podržava *open-closed* princip.

U nasem slucaju, **strategy patern** bi se mogao iskoristiti prilikom odabira nacina sortiranja letova (bilo to po popularnosti, cijeni karte, vremenu polijetanja itd.), gdje bi nam u Strategy podklasama bile metode razlicitih sortiranja, koje bi bile dio IStrategy interfacea.



2) **State patern**

State Pattern je **dinamička verzija Strategy paterna**.
Objekat **mijenja način ponašanja na osnovu trenutnog stanja**.
Postiže se **promjenom podklase unutar hijerarhije klasa**.

U nasem slucaju, mozemo iskoristiti **state patern** kada unosimo jedinstveni kod leta u pretragu, te ukoliko je status leta da je u zraku, metoda pretrage otvara mapu te omogucuje korisniku da prati taj let, dok sa druge strane ukoliko je let u bilo kojem drugom stanju, metoda bi samo ispisala to stanje. Preduslov za ovaj patern bi bio da napravimo podklase aviona za sva njegova stanja, te da ih povezemo sa odredjenim interfejsom.

3) **TemplateMethod patern**

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.

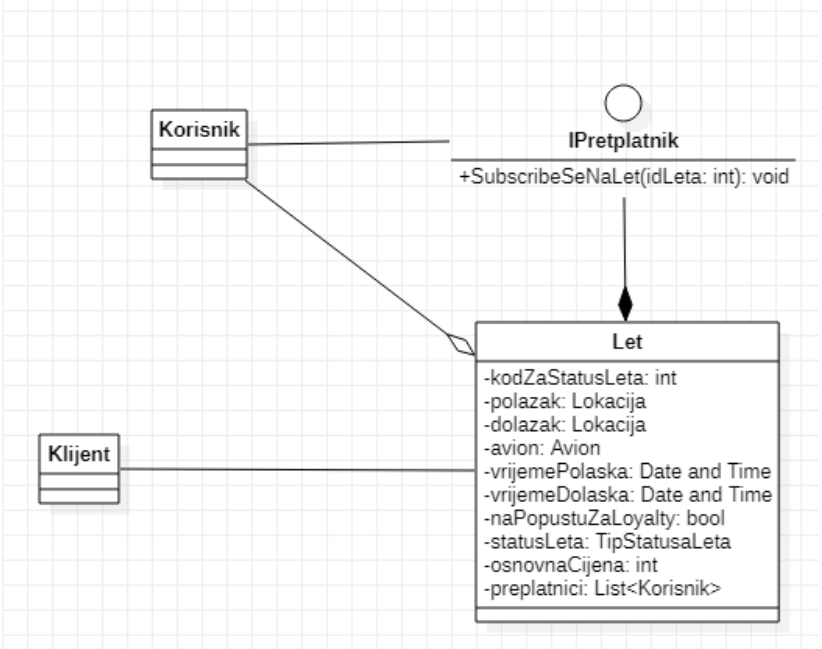
U nasem slucaju, templateMethod patern bi se mogao iskoristiti prilikom kupovine karte, gdje u zavisnosti da li je korisnik prijavljen ili ne, prilikom kupovine karte uracanuvamo popust.
Tok transakcije, prikaz karata i druge mogucnosti ce ostati jednake, jedina razlika ce se ogledati u mogucnosti unosenja koda za popust, zavisno od podklase korisnika.

Osnovna klasa korisnika ima implementirane *default* verzije metoda, a *override* je za popust (koji u opcem slucaju vraca 0) napravljen samo u klasi koja omogucava popust pri kupovini.

4) **Observer patern**

Uloga **Observer paterna** je da **uspostavi relaciju između objekata tako kada jedan objekat promijeni stanje drugi zainteresirani objekti se obavještavaju**.

U nasem slucaju, observer patern bi mogli iskoristiti u slucaju kada bi omogucili korisnicima da se „subsc ribeaju“ na odredjeni let, te da im dolazi notifikacija kada god taj let promijeni svoje stanje.



5) **Iterator patern**

Iterator patern omogućava sekvencijalni pristup elementima kolekcije **bez poznavanja kako je kolekcija strukturirana**.

U nasem slucaju, **iterator patern** bi mogli iskoristiti prilikom korisnikove „posjete“ web shopu. Korisnik bi mogao da bira nacin listanja artikala, bilo kroz *„shuffle“* mode (slucajni redoslijed artikala), *„cijena“* (ide kroz artikle prema nekom algoritmu koji uzima u obzir cijenu) itd.

6) **Mediator patern**

Mediator patern enkapsulira protokol za komunikaciju medju objektima dozvoljavajuci da objekti komuniciraju bez medjusobnog poznavanja interne strukture objekta.

U nasem slucaju, **mediator patern** bi mogli iskoristiti ukoliko bi optimizovali *chat* komunikaciju kupca i programiranog chat-bota u svrhu asistiranja kupcu prilikom koristenja aplikacije.
Poruka se salje medijatoru, koji tu poruku prosljedjuje drugom objektu, u zavisnosti od tipa usluge.

7) **Memento patern**

Memento patern omogucava da spasimo i vratimo prijasnje stanje objekta bez otkrivanja detalja njegove implementacije.

U nasem slucaju, **memento patern** bi mogli iskoristiti prilikom kupovine karte, gdje bi korisnik bio u mogucnosti da se pritiskom na back vrati na prethodno odabrani atribut karte (bilo da je to lokacija polaska, datum leta i sl.). Za implementaciju nam je dovoljna klasa koja bi cuvala prethodna stanja i trenutno stanje, te ukoliko korisnik odluci da predje na transakciju za kupljenu kartu, atributi se proslijedjuju sljedecoj klasi.