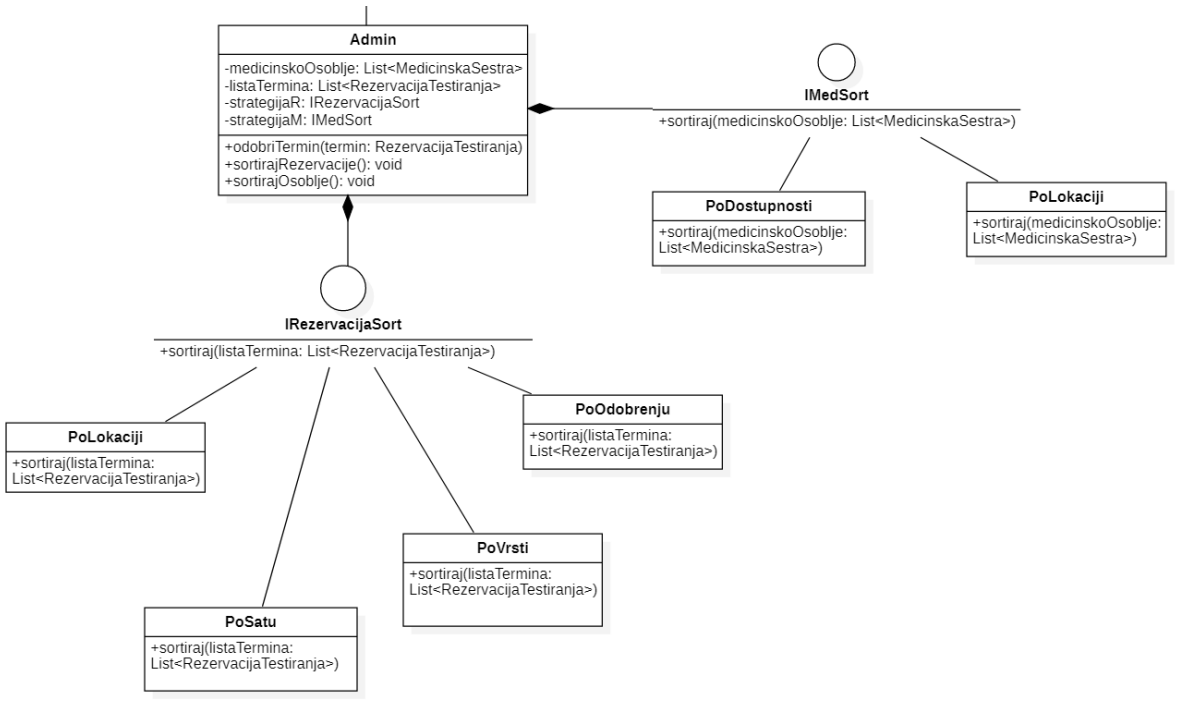
PATERNI PONAŠANJA

1. Strategy patern

Strategy patern služi kako bi se različite implementacije istog algoritma izdvojile u različite klase, te kako bi se omogućila brza i jednostavna promjena implementacije koja se želi koristiti u bilo kojem trenutku. Na ovaj način omogućava se jednostavno brisanje ili dodavanje novog algoritma koji se može koristiti po želji.

Ovaj patern možemo iskoristiti kod realizovanja plaćanja. Kreirali bi Istrategy interfejs za različite vrste plaćanja, na lokaciji i kartično, kao i obračunavanje popusta uz kartično plaćanje. Još jedna ideja jeste da kreiramo interfejs IupravljajBolnicom koji će omogućiti jednostavnije upravljanje sistemom. Ovaj interfejs bi realizovale klase UpravljajKorisnicima, UpravljajRezervacijama, UpravljajOsobljem i slične.
Također, možemo ga primijeniti kod određivanja ozbiljnosti simptoma. Odredili bi vise načina kako preporučiti testiranje na osnovu simptoma, s čime bi samo admin bio upoznat. Jedan način preporučivanja jeste da ako skoči broj zaraženih sa karakterističnim simptomima, počnemo preporučivati testiranje tako da odgovarajući simptomi nose veći dio procenta u algoritmu.
Strategy patern smo implementirali kod prikaza termina testiranja i medicinskog osoblja. Adminu je omogućen sortirani prikaz po odgovarajućim kriterijima, kako bi mu se olakšalo upravljanje sistemom.



2. **State patern**

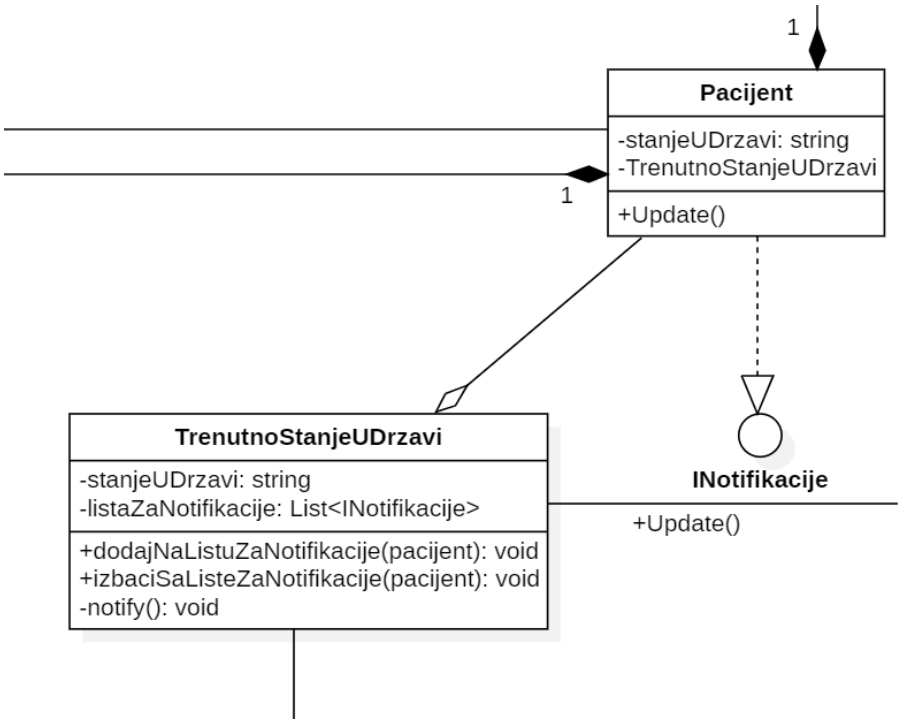
   State patern je ustvari dinamički Strategy patern.
   Ovaj patern bi mogli primijeniti u slučaju da nam karton pacijenata može biti u dva stanja, otvoren i zatvoren. Stanje bi se mijenjalo u zavisnosti od zdravstvenog stanja pacijenta.

3. **Template method patern**

   Template method patern omogućava izdvajanje određenih koraka nekog kompleksnog ili dugog algoritma u određene podklase. Struktura algoritma se ne mijenja odnosno samo mali dijelovi operacija se izdvajaju i ti se dijelovi mogu implementirati različito. Ovaj patern bi mogli iskoristiti pri ograničavanju podataka koje pacijent može sam promijeniti (naprimjer broj kartona, dok podatke kao što su email i broj telefona bi mogao promijeniti), dok bi medicinsko osoblje moglo promijeniti pomenuti podatak.

4. **Observer patern**

   Observer patern uspostavlja relaciju između objekata tako kada jedan objekat promijeni stanje, drugi zainteresirani objekti se obavještavaju. Ovaj patern možemo iskoristiti da kada se stanje pandemije u državi promijeni, ili općenito se desi bilo kakva situacija koja bi korisnika mogla zanimati (dnevna statistika, informacije o testovima), da korisnik dobije obavijest o tome, naravno ako je pretplaćen na primanje istih. Da bi smo implementirali ovaj pattern dodali smo klasu TrenutnoStanjeUDrzavi koja je naš subject. Pacijenti koji odluče da primaju notifikacije dodaju se na listu pretplatnika. Pri promjeni stanja u državi, svi pacijenti koji su pretplaćeni bit će obavješteni.



5. **Iterator patern**
   Iterator patern omogućava sekvencijalni pristup elementima kolekcije bez poznavanja kako je kolekcija strukturirana. Ovaj patern možemo iskoristiti kod prolaska kroz liste medicinskog osoblja i pacijenata. Iz odgovarajućeg interfejsa, mogli bi izvesti klase ZaraženiFirst, OporavljeniFirst, KritičniFirst, DostupniFirst(medicinsko osoblje), koje će predstavljati način prolaska kroz odgovarajuće liste.

6. **Chain of responsibility patern**
   Chain of responsibility patern je namijenjen kako bi se jedan kompleksni proces obrade razdvojio na način da više objekata na različite načina procesiraju primljene podatke. U našem sistemu ne postoji nijedna kompleksna izvedba da bi mogli primijeniti ovaj patern. Međutim, kada bi izvedba kartona bila kompleksnija, pravo rješenje bi bilo primijeniti ovaj patern.

7. **Mediator patern**
   Medijator patern smanjuje broj veza između objekata. Umjesto direktnog međusobnog povezivanja velikog broja objekata, objekti se povezuju sa međuobjektom medijatorom, koji je zadužen za njihovu komunikaciju. Ovaj patern bismo iskoristili pri omogućavanju chat komunikacije između admina i medicinskog osoblja.