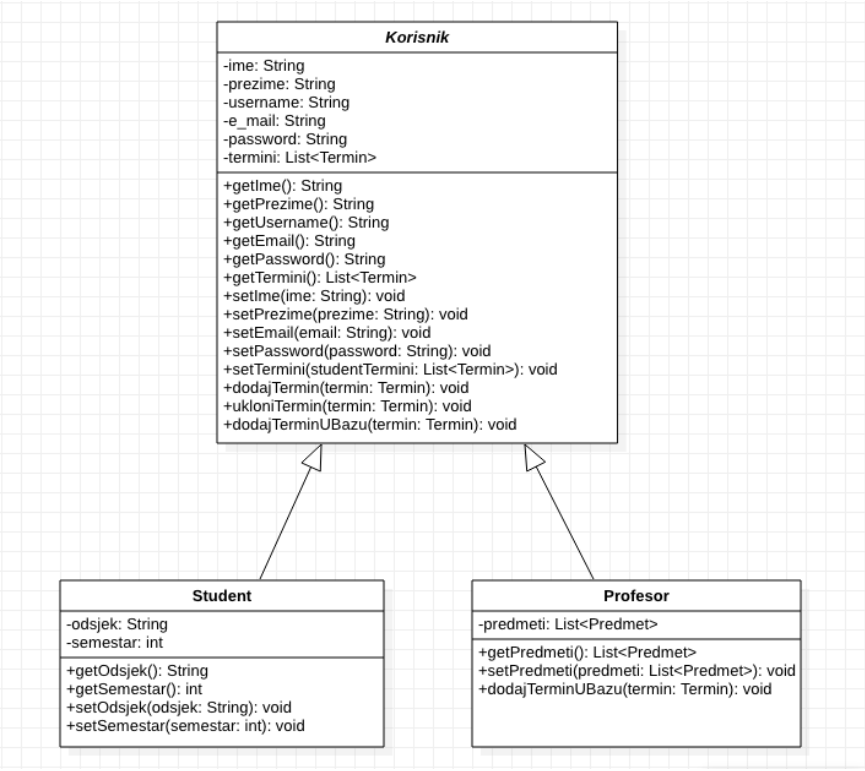
Paterni ponašanja

1. *Strategy* patern

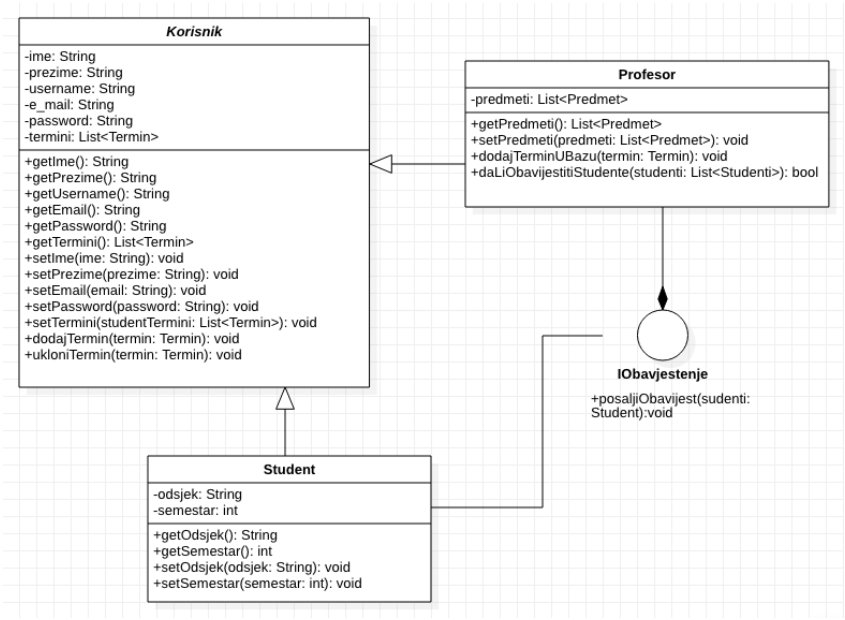
   Ovaj patern služi za razdvajanje različitih implementacija istog algoritma u različite klase.
   Ukoliko bismo imali metodu *optimiziraj* koja bi optimizirala raspored upotrebom različitih algoritama, tada bismo uveli interfejs *IOptimizacija* koji bi imao metodu *optimiziraj*. Tu metodu bi implementirale klase algoritama za optimizaciju (npr. ako bismo primijenili Ford Fulkersonov, Dijkstrin i Bellman Fordov algoritam). U slučaju da aplikacija utvrdi da je jedan algoritam efikasniji u odnosu na druge, s obzirom na broj studenata po semestru, ona će iskorititi određeni algoritam.

2. *State* patern
   Ovaj patern se koristi kada se želi omogućiti objektu da mijenja svoja stanja od kojih zavisi njegovo ponašanje. U zavisnosti od stanja, čini se kao da je objekat promijenio klasu, a ta stanja se mijenjaju automatski.
   U našem projektu se ne mogu naći primjene ovog paterna.

3. *Template Method* patern
   Ovaj patern služi za izmjenu ponašanja u jednom ili više dijelova.
   Template method patern je već primijenjen i to u dijelu kada imamo apstraktnu klasu *Korisnik* iz koje su naslijeđene *Student* i *Profesor*. U klasi *Korisnik* imamo metodu *dodajTerminUBazu* koju u suštini samo profesor koristi tj. sudent nema opciju dodavanja i uklanjanja termina u bazu, a profesor ima.



4. *Observer* patern
   Observer patern služi za kreiranje mehanizma pretplaćivanja. Pretplatnici dobiju obavještenje o sadržajima za čije slanje je zaslužna nadležna klasa. Ako bismo u našem projektu dodali funkcionalnost da nakon što su svi profesori unijeli termine predavanja i tutorijala studentima dođe obavijest (npr. na mail) o tome da mogu dobiti optimizirani raspored, tada bismo iskoristili ovaj patern. U klasi *Profesor* bi postojala metoda *daLiObavijestitiStudente* u kojoj bi se vršila provjera je li taj profesor posljednji u tom semestru koji je dodao potrebne termine predavanja i tutorijala i na osnovu toga bi se slalo obavještenje svim studentima u datom semestru da je njihova optimizacija spremna.



5. *Iterator* patern
   Iterator patern služi za omogućavanje prolaska kroz listu elemenata bez da su poznati implementacijski detalji struktrue u kojoj se čuvaju elementi.
   U našem projektu nemamo vidljivu primjenu ovog paterna.

6. *Chain of Responsibility* patern
   Ovaj patern se koristi da se kompleksi procesi obrade razdvoje tako da više objekata na različite načine procesira primljene podatke. Realizuje se tako što svaki objekat koji vrši parcijalnu obradu prosljeđuje idućem objektu u nizu čime se smanjuje kompleksnost individualnih procesa i svaki objekat dobije samo podatke koji su mu potrebni.
   S obzirom na to da u našoj aplikaciji nema potrebe za automatskim dodavanjem određenih dijelova, ovaj patern nije primjenjiv na našu aplikaciju.

7. *Mediator* patern
   Ovaj patern se koristi u svrhu manjenja broja veza između objekata. Objekti se ne spajaju direktno već uz pomoć medijatora koji je zadužen za komunikaciju. Pomoću njega kada jedan objekat želi proslijediti podatke drugom, to se

realizuje preko medijatora koji odlučuje da li će proslijediti te podatke s obziorm na zadovoljenost određenih uslova.
Ukoliko bismo u našoj aplikaciji uveli da i asistent može dodavati termine, ali samo tutorijala, a ne i predavanja onda bismo imali interfejs *IMedijator* koji bi imao metode *provjeri_* koja provjerava da li je osoba student – ne može dodavati nikakve termine, asistent – može dodavati termine samo tutorijala ili profesor – može dodavati termine tutorijala i predavanja.