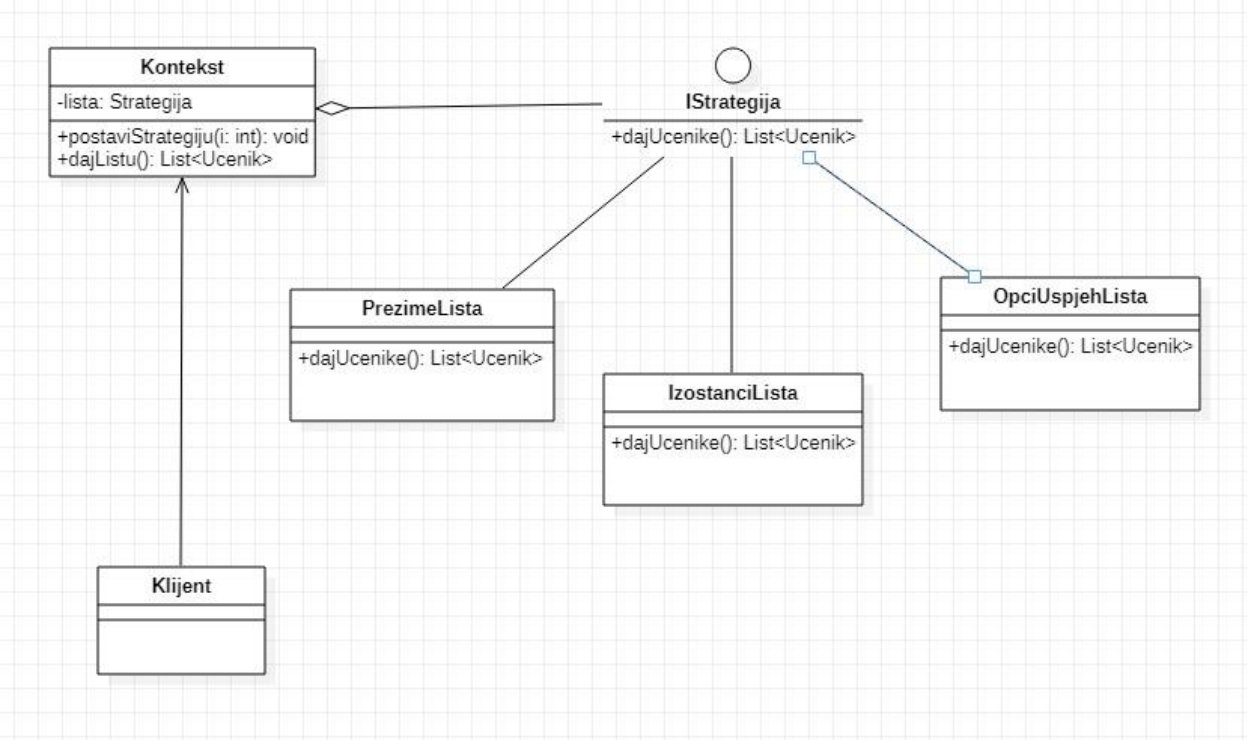
Paterni ponašanja

Strategy pattern (implementiran)

Strategy patern izdvaja algoritam iz matične klase i uključuje ga u posebne klase.

Razrednik ima mogućnost pregleda liste učenika iz svog odjeljenja po različitom kriteriju. Dakle, on može zahtijevati listu učenika sortiranih abecedno po prezimenu, zatim listu učenika sortiranu po općem uspjehu na kraju polugodišta ili na kraju školske godine, zatim listu učenika sortiranu po ukupnom broju izostanaka. Primjenom ovog paterna omogućeno je da se u budućnosti doda još kriterija za sortiranje učenika jednog odjeljenja. Klasa *Kontekst* ima atribut *lista* tipa interfejs *IStrategija*. Njega implementiraju klase *ProsjekLista*, *PrezimeLista* i *IzostanciLista* koje preko metode *dajUcenike*():list<Ucenik> vraća listu učenika sortiranu po odgovarajućem kriteriju.



State pattern

State patern mijenja način ponašanja objekta na osnovu trenutnog stanja.

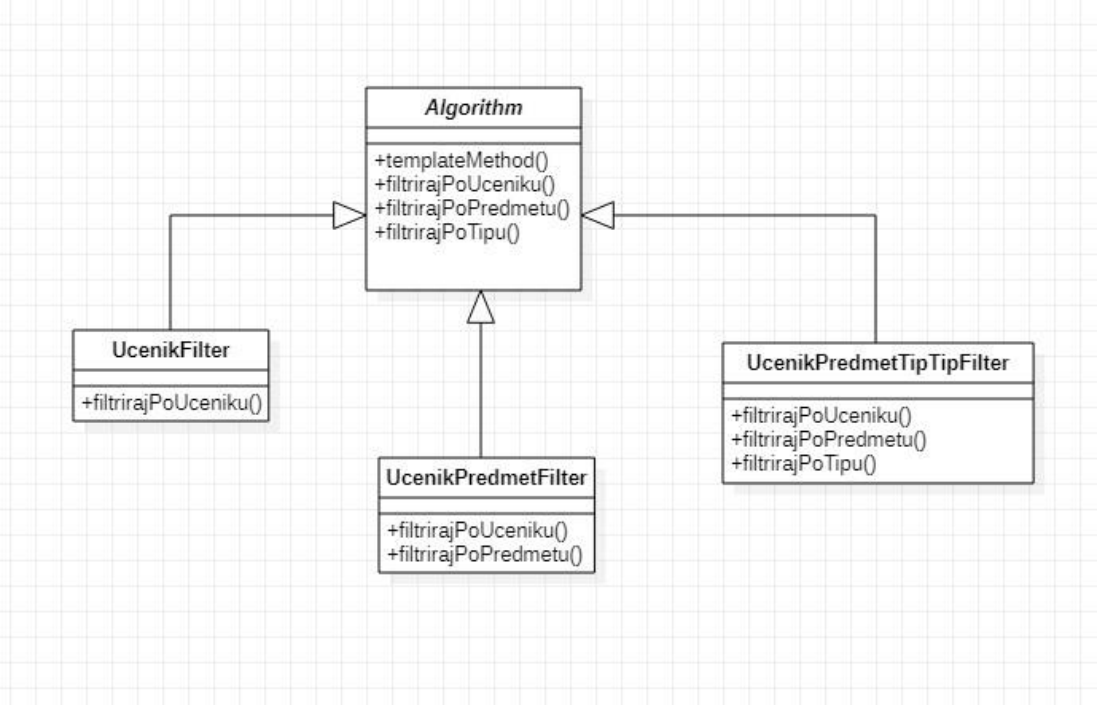
Ovaj patern nismo primijenili u našem sistemu. Mogli bismo ga primijenit na sljedeći način. Nastavnik može imati dva stanja u odnosu na jedno odjeljenje: samo nastavnik ili razrednik tom odjeljenju. Ukoliko je samo nastavnik, ne bi imao mogućnost pravdanja izostanaka, evidentiranja izvještaja sa roditeljskog sastanka, zaključivanja općeg uspjeha itd., dok bi kao razrednik to mogao. Vrijednost ovog atributa bi se mijenjala sa promjenom odjeljenja sa kojim radi. Da bismo primijenili ovaj patern, trebali bismo u klasu *Nastavnik* dodati atribut *stanje*:Stanje i

metodu *promijeniStanje*(). Zatim, bismo dodali interfejs *Stanje* sa metodama iz klase *Nastavnik* koje su vezane za upis ocjene, upis časa, pravdanje izostanaka itd. Taj interfejs bi implementirali klase *NastavnikOdjeljenju* i *RazrednikOdjeljenju*. Ukoliko bi se pozvala metoda *opravdajIzostanak*, a nastavnik je u stanju *NastavnikOdjeljenje* ne bi se desilo nista, a ukoliko je *RazrednikOdjeljenju* metoda bi se ispravno izvršila. Ovaj patern nismo primijenili, jer smo zamislili da naš sistem drugačije funkcioniše.

Template Method pattern (implementiran)

Template Method patern omogućava izdvajanje određenih koraka algoritma u odvojene podklase.

Primijenili smo Template Method patern kako bi omogućili izdvajanje određenih ocjena iz liste ocjena jednog odjeljenja. Recimo, želimo pregled svih ocjena nekog učenika, zatim možemo specificirati predmet iz kojeg želimo vidjeti ocjene, zatim odabrati tip ocjene (PISMENA ZADACA, TEST, AKTIVNOST itd.). Kako bi primijenili ovaj patern u naš model dodali smo klase *Algorithm, UcenikFilter, UcenikPredmetFilter* i *UcenikPredmetTipFilter*. Klasa *Algorithm* je apstraktna klasa koja ima metode *templateMethod(), filtrirajPoUceniku(), filtrirajPoPredmetu()* i *filtrirajPoTipu()*. Dakle, ukoliko želimo ocjene samo jednog učenika primijenit će se klasa *UcenikFilter*, ukoliko odredimo i predmet, primijenit će se klasa *UcenikPredmetFilter* i ukoliko navedemo sva tri parametra primijenit će se klasa *UcenikPredmetTip*.



Observer pattern

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

Ovaj patern nismo primijenili u našem sistemu, ali bismo ga mogli primijeniti ukoliko želimo da se roditelj i razrednik obavijeste ukoliko izostanak učenika nije opravdan u periodu od 7 dana od dana kada se desio. U tom slučaju bi trebali definisati interfejs *IPratilac* koji bi imao metodu *update*(). Klase *Razrednik* i *Roditelj* bi implementirali interfejs *IPratilac*. U klasi *Izostanak* pored navedenih atributa bi imali atribut *pratioci*:list<Osoba> u kojem bi čuvali listu pratioca (u našem slučaju to bi bili samo razrednik i roditelj, pa bi ih mogli čuvati odvojeno, ali kako bi ostavili mogućnost za dodavanje još pratioca koristili bi listu), zatim metode *obavijestiPratioce()* koja je privatna metoda, *dodajPratioca*(osoba:Osoba) i *izbrisiPratioca*(osoba:Osoba) (u slučaju da se promijeni razrednik) i metodu *provjeriStanje*() koja bi provjerila da li je prošao zadani period i u slučaju da jeste poziva se metoda *obavijestiPratioce*().

Iterator pattern

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

Ovaj patern nismo primijenili u našem sistemu.

U klasi *Odjeljenje* čuvamo listu učenika tog odjeljenja. Primjenom ovog paterna bismo omogućili iteriranje foreach petljom kroz listu učenika abecedno po prezimenu. Dodali bismo novu klasu *UcenikIterator* i dva nova interfejsa *Iterator* i *Aggregate*. Interfejs *Aggregate* bi imao metodu *createIterator*(), a klasa *Odjeljenje* bi implementira taj interfejs. Klasa *UcenikIterator* bi implementira interfejs *Iterator* koji bi imao metode *hasNext*() i *next*().