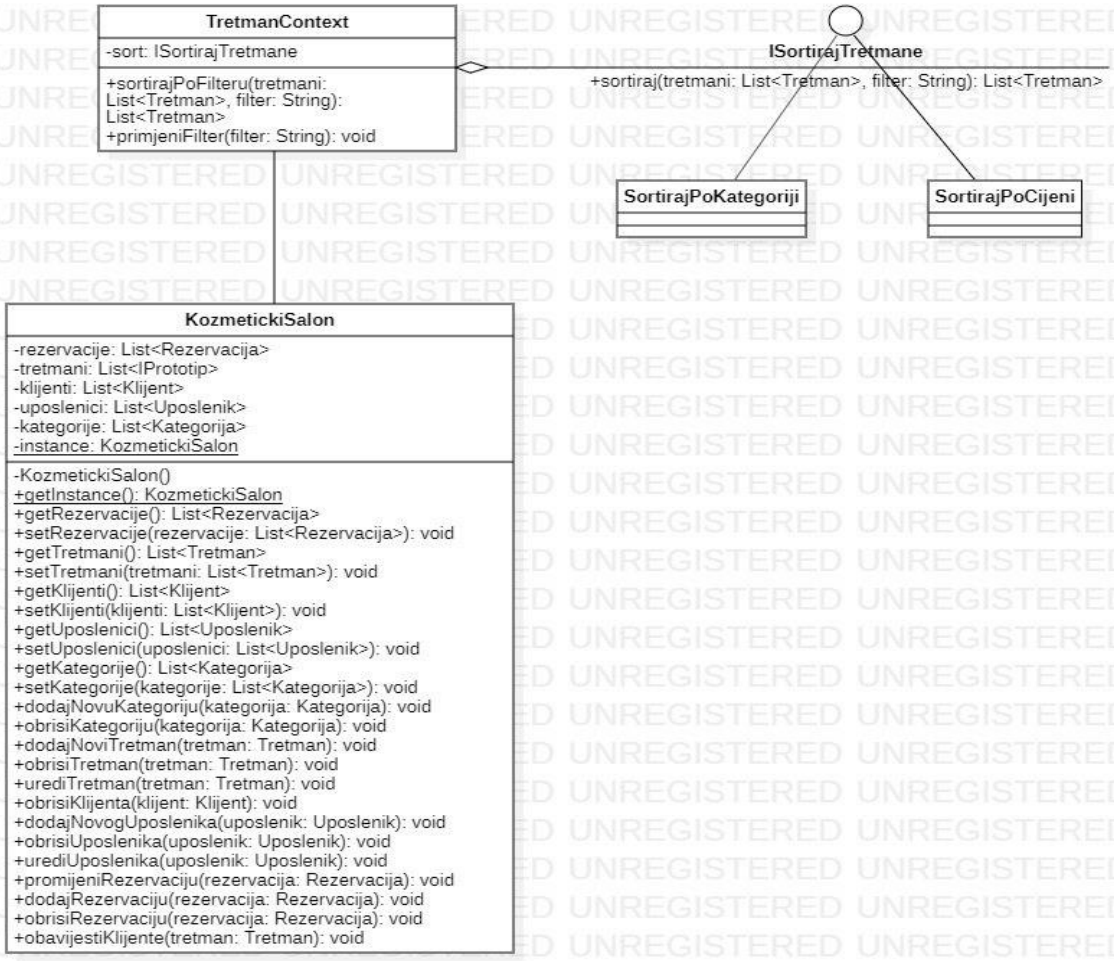
Paterni ponašanja

1. Strategy pattern

Strategy patern 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.

Strategy patern u našem sistemu primjenile smo tako što smo klijentu dodijelile mogućnost prikaza tretmana sortiranih po cijenama ili kategorijama.

Dodale smo klasu TretmanContext koja će interfejsu ISortirajTretmane proslijediti neophodne podatke (listu tretmana i filter po kojem će se vršiti sortiranje tretmana). Ovaj interfejs ima metodu sortiraj(List<Tretmani> tretmani, String filter): List<Tretmani>, koju će implementirati klase SortirajPoKategoriji i SortirajPoCijeni u kojima se vrši sortiranje tretmana po samom tipu filtera. Klasa TretmanContext kao privatni atribut ima objekt tip ISortirajTretmane te metode sortirajPoFilteru(List<Tretman> tretmani, String filter):List<Tretmani> i promijeniFilter(String filter).



2. State pattern

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

Ovaj patern nismo primijenili u našem sistemu. Kada bismo imali mogućnost pri rezervaciji tretmana izabrati da li želimo obični, kraljevski ili vip tretman mogli bismo primijeniti navedeni patern. Imali bismo interfejs IState, koji ima metodu promijeniStatus i prima parametar tipa Context.

Dodale bismo i klasu Context koja bi kao atribute imala status tretmana tipa IState, statusTretmana tipa String i metodu operacijaPromjene(String buduceStanje) u kojoj bismo trenutno stanje promijenili u buduće. Stanje tretmana prije rezervacije se smatra trenutnim stanjem i ono je na početku uvijek „slobodno“(statusTretmana). Također, naš sistem bi imao još 3 klase: ObicniTretman, KraljevskiTretman, VipTretman, koje bi implementirale interfejs IState i koje bi unutar sebe mijenjale vrijednost atributa statusTretmana.

3. Template Method pattern

Template method 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 primijenili, ali kada bismo uveli mogućnost plaćanja kroz sistem mogli bismo ga primijeniti na sljedeći način. Dodali bismo klasu Placanje koja bi u sebi sadrzavala metodu void plati(IObracun x), gdje je IObracun interfejs. Ova metoda obračunava cijenu koliko klijent treba platiti za svoj rezervisani tretman. Imali bismo klase Klijent i VipKlijent(koje implemetiraju interfejs IObracun), pa ta cijena ne bi bila ista u ovisnosti koji je klijent u pitanju(VipKlijent uvijek ima popust npr.5%).

Interfejs IObracun bi imao metode String Operacija1() i String Operacija2() koje bi računale cijenu tretmana i popusta respektivno. Ove dvije metode bi se pozivale iz metode *plati* nad njenim parametrom. *Operacija1* bi vraćala cijenu rezervisanog tretmana i ona bi bila ista kod obje vrste klijenata, dok bi *Operacija2* vraćala nulu u klasi Klijent, a u klasi VipKlijent vrijednost 5% cijene.

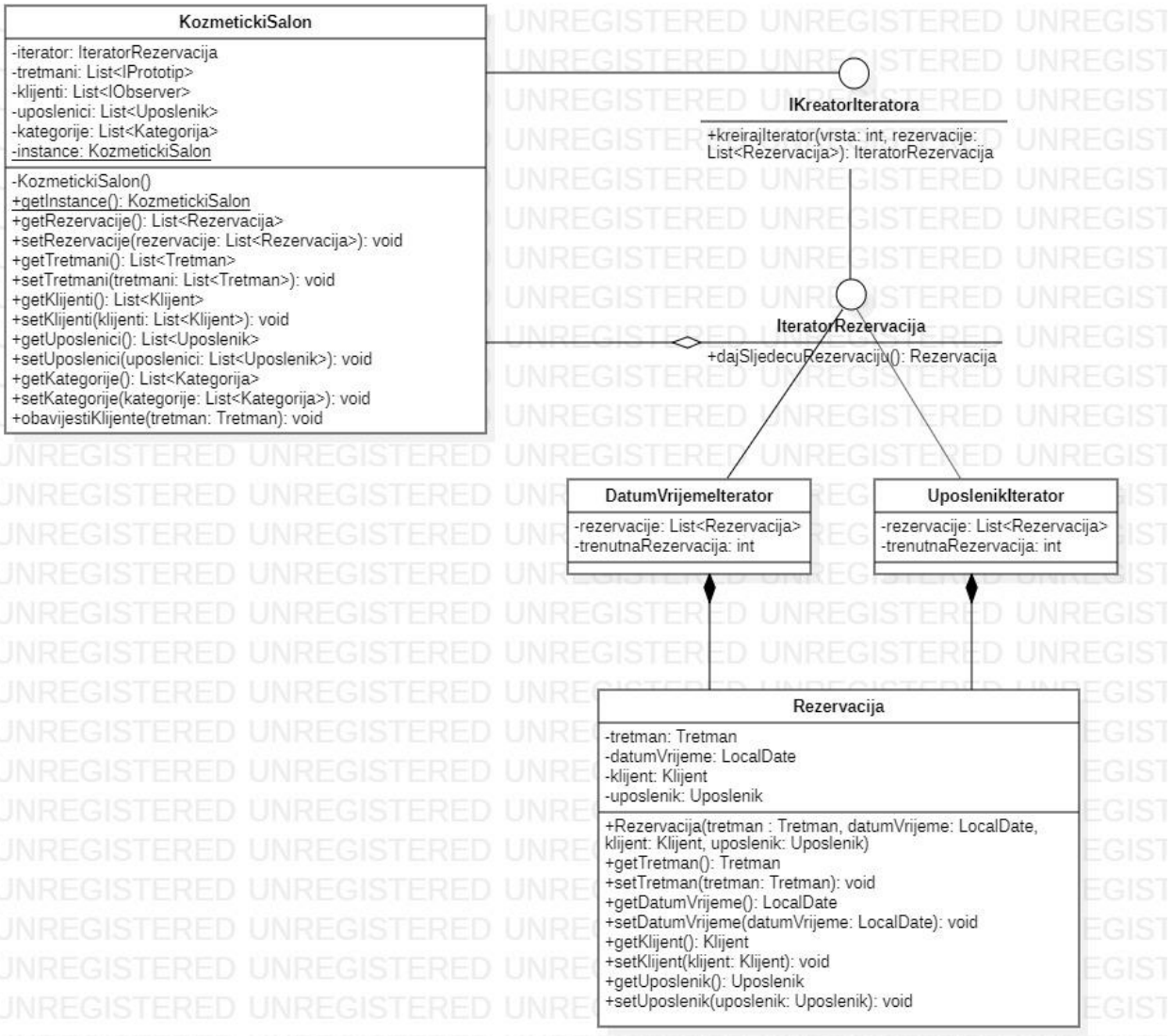
U metodi *plati* bismo mogli pisati npr. Double ukupno = x.Operacija1() - x.Operacija2();

4. Iterator pattern

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

Ovaj patern iskoristili smo u našem sistemu na sljedeći način: klasa KozmetickiSalon kao jedan od atributa sadrži listu rezervacija. Želimo omogućiti prolazak kroz listu prema uposlenicima koji će vršiti rezervisani tretman i prema vremenu rezervacije.

Dodale smo interfejs IIteratorRezervacija koji ima metodu dajSljedecuRezervaciju():Rezervacija koji nasljeđuju klase UposlenikIterator i DatumVrijemeIterator čiji su atributi rezervacije:List<Rezervacije>, tretnutnaRezervacija: int. Klasa KozmetickiSalon kao atribut sadrži objekat tipa IiteratorRezervacije. Dodale smo i interfejs IKreatorItretatora (metoda: kreirajIterator(int vrsta, List<Rezervacija> rezervacije): IIteratorRezervacija) koji omogućava promjenu načina iteriranja. U klasi KozmetickiSalon sada umjesto liste rezervacija imamo atribut iterator tipa IteratorRezervacija.



5. Observer pattern

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

U našem sistemu observer patern iskorišten je na sljedeći način. Kada administrator doda novi tretman u listu tretmana (u klasi KozmetickiSalon), klijenti dobiju obavještenje o dodanom tretmanu (Dostupan je novi tretman).

U klasu KozmetickiSalon dodale smo metodu void obavijestiKlijente(Tretman tretman) koja će se pozivati svaki put kada administrator doda novi tretman. Definisale smo i interfejs IObserver koji ima metodu void update(), a koji implementira klasa Klijent. . U klasi KozmetickiSalon smo atribut tipa List<Klijent pretvoili u List<IObserver>, u kojem će se nalaziti svi klijenti koji će dobiti obavještenje kada tretman bude dodan.

Na ovaj način, u našem sistemu postigli smo da sve klijente u isto vrijeme obavijestimo o dodavanju novog tretmana.