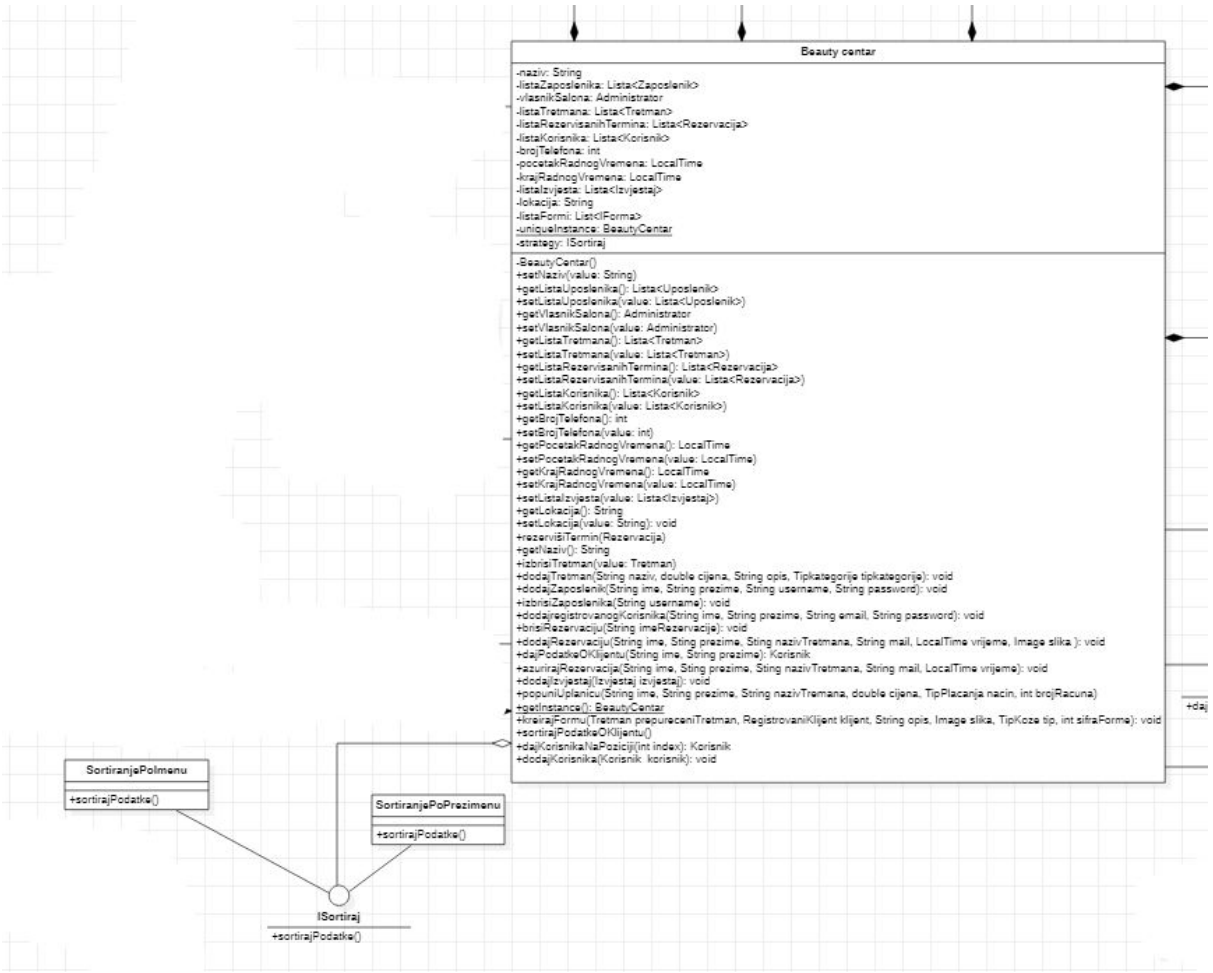
- Dizajn paterni

Paterni ponašanja

- Strategy patern

  Strategy patern izdvaja algoritam iz matične klase i uključuje ga u posebne klase. Pogodan je ukoliko postoje različiti primjenjivi algoritmi za neki problem.
  Potencijalna mjesta za primjenu ovog paterna vidimo pri sortiranju te pretraživanju podataka o klijentu, tretmanima pa čak i izvještajima.
  Ukoliko razmatramo primjenu ovog paterna pri sortiranju podataka o klijentima u sistemu,možemo to realizovati na slijedeći način. Možemo omogućiti dva načina sortiranja, po imenu i po prezimenu.
  Da bi implementirale ovaj patern, potrebno je da imamo interfejs ISortiraj u kojem imamo metodu sortiraj() te konkretne implementacije ovog interfejsa, što su ovom slučaju klase SortiranjePoImenu i SortiranjePoPrezimenu. U ovim klasama se nalazi implementacije metode sortiraj().
  Da bi radnik sistema imao pristup za korištenje date pogodnosti, potrebno je klasu BeautyCentar posmatrati kao Context klasu te u njoj kao atribut čuvati strategiju koja je tipa ISortiraj i metodu sortiraj() koja će uz određene uslove birati koja će se to klasa koristiti. Također još jedna primjena ovog paterna bi se mogla realizovati na veoma sličan način ukoliko se odlučimo da implementiramo pretraživanje korisnika u sistemu po imenu ili prezimenu. Analogno, kreira se ista struktura kao i u slučaju sortiranja, samo u ovom slučaju metoda koje će biti u interfejsu i koja će se implementirati u konkretnim klasama je tražiKorisnika() a unutar BeautyCentra će se odrediti koja će se klasa koristiti. Pozivanje ovih metoda može biti aktivirano pritiskom na dugme ili neku opciju na grafičkom okruženju aplikacije.



- State patern

  State patern koristimo kada imamo objekt koji se ponaša različito ovisno o trenutnom stanju u kojem se taj objekat nalazi.Ovaj pater mozemo realizovati tako sto ćemo napraviti klase koje ce tačno opisivati trenutno stanje objekta.

  U našem modelu ovaj patern nije primjenjen. Međutim način na koji bi to uradile i gdje bi mogao biti primjenjen bit će opisan u nastavku. Mjesto na koje smo mi odlučile da bi primjenile ovaj patern jeste davanje mogućnosti rezervacije da mijenja stanje tj. u našem slučaju rezervacija mijenja ponašanje u ovisnosti o njenom trenutnom stanju. Ukoliko je rezervacija aktivna i jos nije došao trenutak za njenu realizaciju ona se ponaša kao aktivna rezervacija, međutim ukoliko rezervacija bude otkazana ili ukoliko rezervacija bude realizovana tj. da je završen tretman u salonu koji je bio zakazan preko određene rezervacije , rezervacija u tom slučaju mijena stanje i ponaša se kao neaktivna rezervacija.
  Pošto se ovaj patern sastoji od Contexta, interfejsa ili apstraktne klase koju nasljeđuju određene klase koje ustvari opisuju trenutno stanje, te metoda preko kojih se ustvari osvtaruju funcionalnost ovog paterna, u našem modelu ono sto je potrebno da bi bila moguća realizaija paterna jeste: klasu Rezervacija proglasiti za apstraktnu klasu , te dodati klasu AktivnaRezervacija i NeaktivnaRezervacija koje ce naslijediti

klasu Rezervacija. Klasa BeautyCentar bi bila ustvari Context klsa, klasu bi izmijenile na način da dodamo metode koje će omogućiti pozivanje metoda iz izvedenih klasa kako bi određena rezervacija mijenjala način ponašanja u ovisnosti o trenutku .
Dodana metoda u Beauty centar bi bila oznaciRezervaciju() kao i otkaziRezervaciju() koje imaju parametar koji označava indeks rezervacije u listi.
Klase koje su izvedene bi imale implementiranu metodu odrediStanjeRezervacije() koja kao parametar prima BeauyCentar kao Context i koja bi imala parametar tipa **bool** na osnovu kojeg bi određivali da li je potrebno promijeniti stanje objekta tj. rezervacije.
Npr. ukoliko je odabrana jedna od rezervacija iz liste rezervacija te zelimo da otkazemo tu rezervaciju a ona je je tipa Aktivna rezervacija prvo se poziva metoda otkaziRezervaciju(int i) zatim u ovoj metodi se nad tom rezervacijom poziva metoda odrediStanjeRezervacije() sa patametrom bool koja će ukoliko bool ima vrijednost false promijeniti stanje te klase te staviti da ta rezervacija ima stanje NeaktivneRezervacije . Te promjene će biti sačuvane u listiRezervacija u BeauyCentru.

- Iterator patern

  Iterator patern omogućava sekvencijalni pristup elementima kolekcije bez poznavanja kako je kolekcija struktuirana. Struktura iterator paterna se sastoji iz nekoliko dijelova. Prvi je kolekcija objekata kroz koju se želi iterirati. U našem sistemu, kolekcija objekata kroz koju se želimo kretati, lista korisnika se nalazi u klasi BeautyCentar. Zatim postoji interfejs IterableCollection, koji sadrži definiciju metode createIterator(). Klasa BeautyCentar implementira ovaj interfejs, i sadrži implementaciju metode createInterator(), koja vraća instancu klase Iterator, gdje se prilikom kreiranja instance proslijeđuje kao parametar cijela klasa BeautyCentar (new Iterator(this, 0)). U ovu klasu ćemo dodati i metode dajKorisnikaNaPoziciji(int indeks) koja vraća instancu klase Korisnik, te dodajKorisnika(Korisnik korisnik). Naredni dio je kreiranje interfejsa InterfejsIterator koji sadrži definicije metoda hasNext(), isCompleted() i getNext(). Postoji i klasa Iterator, koja mapira kolekciju objekata(u našem slučaju korisnika) iz klase BeautyCentar kao svoj atribut. Pored toga, ova klasa sadrži i atribut tretnutniIndeks tipa int. Klasa Iterator implementira interfejs InterfejsIterator, te sadrži implementacije metoda iz ovog interfejsa.
  Metoda getNext() poziva metodu dajKorisnika(tretnutniIndex) klase BeautyCentar i vraća korisnika koji se u listi nalazi na poziciji trenutniIndex koji smo inicijalizirali pri kreiranju iteratora. Metoda hasNext() bi nam mogla vraćati Korisnika koji je na idućoj poziciji. Metoda isCompleted() nam omogućava da provjerimo da li je doslo do kraja kolekcije.

- TemplateMethod pattern

  Ovaj patern omogućava izdvajanje određenih koraka algoritma u odvojene klase. Sama struktura algoritma se ne mijenja, nego se mali dijelovi operacija izdvajaju i oni se mogu različito implementirati. Ovaj patern bismo mogli iskoristiti u našem sistemu npr. za pretraživanje korisnika. To bismo uradili na način da imamo klasu Pretrazivanje i u toj klasi metodu pretrazi(IPretrazivanje obj, int nacinPretazivanja, int brojOdradjenihTermina = 0). Zatim bismo dodali interfejs IPretrazivanje koji bi sadržavao definicije metoda pretražiPoImenu(), pretraziPoPrezimenu, pretraziPoBrojuOdradjenihTretmana(), itd. Pošto nam se lista korisnika nalazi u klasi BeautyCentar, mogli bismo staviti da klasa BeautyCentar implementira interfejs IPretrazivanje, i u njoj dodati implementacije metoda koje su navedene u interfejsu. Metoda pretrazi(IPretrazivanje obj, int nacinPretrazivanja, , int brojOdradjenihTermina = 0) bi bila implementirana na način da ako je npr. nacinPretrazivanja=0, onda se iz nje poziva obj.pretraziPoImenu(), ako je nacinPretrazivanja = 1, onda se poziva obj.pretraziPoPrezimenu(), te ako je nacinPretrazivanja = 2, onda se

poziva obj.pretraziPoBrojuOdradjenihTermina(brojOdradjenihTermina). Svaka od navedenih metoda vraća listu korisnika. Klijent bi ovo mogao koristiti na način:

Pretrazivanje p = new Pretrazivanje()

List<Korisnik> lista = p.pretrazi(new BeautyCentar(), 0).

- Observer patern

  Glavna uloga ovog paterna je da uspostavi relaciju između objekata tako da kada jedan objekat promijeni stanje, drugi zainteresirani objekti se obavještavaju. U našem sistemu, pogodan slučaj za primjenu ovog paterna bi bio ukoliko želimo da svaki put kada se doda novi tretman u sistem, šalje poruka svim registrovanim korisnicima u vidu obavještenja da postoji nova usluga koju bi mogli isprobati i informisati se više o pogodnostima iste. Također slična obavještenja bi se mogla slati ukoliko se promijeni cijena nekih usluga ili kreira specijalna ponuda. Ukoliko bi se odlučile za prvu opciju, tj.da se šalje poruka korisniku svaki put kada se doda novi tretman u sistem, to bi implementirali na sljedeći način. Klasa BeautyCentar bi bila Subject klasa u kojoj se čuva lista registrovanih korisnika kojima se treba poslati obavještenje. Također u BeautyCentar klasu bi se trebale dodati metode registrujNovogClanaNaListiZaObavjestenja(RegistrovaniKorisnik), Notify(), ukloniČlanaSaListeObavjestenja(RegistrovaniKorisnik) te privatni event pošaljiObavijest() koji kada se aktivira šalje svoje stanje kao parametar Update() metodi unutar klase RegistrovaniKorisnik. Potrebno bi bilo dodati interfejs IObserver u kojem se nalazi metoda Update() a konkretna implementacija istog bi bila klasa Registrovani korisnik u koju bi dodali metodu Update().