Dizajn paterni

Strukturalni paterni

- *Adapter pattern*

  Osnovna namjena ovog paterna je da omogući širu upotrebu već postojećih klasa te ga koristimo kada je potreban drugačiji interfejs već postojeće klase.Pattern nije iskorišten u našem sistemu.

  U našem sistemu je moguće iskoristiti ovaj patern u klasi izvještaj, ukoliko želimo podržati više formata za njegovu izradu. Primjer, generisanje izvještaja preko JSON-a kao osnovna varijanta ali i mogućnost generisanja istog u pdf formatu.Na koji nacin bi ovo moglo biti realizovano u nasem sistemu objasnit ćemo u nasvki, medjutim jos jedno potencijalno mjesto za korištenje ovog paterna vidimo u rezervaciji termina, gdje se ona može realizovati preko online zakazivanja ali i preko Recepcionera u poslovnici. Osnovna varijanta realizacije zakazivanja je online, ali postoji i proširena varijanta.

  Kada bi u klasi BeuatyCentar imali metodu dodajIzvjestaj(JSON izvjestajJSON) koja prima izvještaj u JSON formatu. Te na osnovu tog patametra kreira objekat Izvjestaj te dodajte u listu Izvjestaja. Međutim ukoliko zelimo da imamo mogucnost da kreiramo objekat Izvjestaj na osnovu PDF formata ono sto je potrebno da se uradi jeste:
  - dodati interfejs IFormat() sa metodom dodajPDF(PDF izvjestaj),
  - trebalo bo dodati klasu IzvjestajAdapter koja ce naslijediti interfejs IFormat() te implementirati metodu dodajPDF(PDF izvjestaj), ova metoda bi imala instancu Izvjestaja JSON tipa kojem bi vrijednost bila dodjeljena tako što bi se uradio convert PDF formata u JSON format ,
  - takodjer klasa Adapter je izvedena iz BeautyCentar klase , te implementaciji metode dodajPDF nakon konvertovanja pozivamo metodu dodajIzvjestaj koja prima kao parametar prethodno dobijen objekat (izvjestaj) JSON tipa.

- *Facade pattern*

  Ovaj patern se koristi kada sistem ima više identificiranih podsistema pri čemu su apstrakcije i implementacije podsistema usko povezane. Pattern nije iskorišten u našem sistemu.

  Kako bi ispunili ovaj pattern potrebno je izmijeniti sistem na način da u klasu RegistrovaniKorisnik dodamo metodu getRedniBrojTretmana() koja bi vraćala broj koji govori koji je redni broj tretmana kojeg je korisnik upravo odradio u listi njegovih odrađenih tretmana. Ovo nam je korisno pri naplaćivanju tretmana, jer smo dali mogućnost da je tretman pod određenim rednim brojem bespalatan za tog korisnika, pa bismo pomoću ove metode provjerili redni broj, te ako je on odgovarajući, neće se izvršiti naplata. Klasa Uplatnica bi nam predstavljala fasadu i ona bi u sebi imala metodu naplati() . Iz ove klase bismo mogli izvesti klase koje bi predstavljale različite načine plaćanja, kao npr. plaćanje online, plaćanje preko paypal-a, plaćanje direktno preko kartice, skeniranje uplatnica, itd. U svakoj od ovih klasa bismo implementirali metode koje bi realizovale pojedinačne načine plaćanja, a u metodi naplati() klase Uplatnica bismo provjerili treba li izvršiti naplatu ili je tretman besplatan, te ako treba izvršiti naplatu pozvali bismo odgovarajuću metodu iz izvedenih klasa. Klijent bi mogao koristiti samo metodu naplati() klase Uplatnica, a da ne zna kako su implementirane metode u izvedenim klasama.

- *Decorator pattern*

  Osnovna namjena Decorator paterna je da omogući dinamičko dodavanje novih elemenata i ponašanja postojećim objektima.
  Pattern nije iskorišten u našem sistemu.

  U našem sistemu potencijalnu primjenu Decorator paterna vidimo u rezervaciji termina. Osnovna forma rezervacije je dio sa formalnim unosom podataka i odabirom tretmana dok dodatna pogodnost koja se može realizirati ovim paternom je dodavanje slike željenih rezultata tretmana na datu rezervaciju. Za to su nam potrebne klase Rezervacija, RezervacijaSaOsnovnimPodacima, RezervacijaSaSlikom, te interfejs IRezervacija. Klase RezervacijaSaOsnovnimPodacima i RezervacijaSaSlikom bi implementirale interfejs IRezervacija, te sadržavale atribut tipa klase koja je na sljedećem nivou apstraktnosti. Ovaj interfejs bi sadržavao definiciju metode urediRezervaciju(int nacinUredjivanja), pa ako je nacinUredjivanja = 0, onda rezervacija sadrži samo osnovne podatke, a ako je nacinUredjivanja = 1, onda se dodaje i slika. Pored ove metode, interfejs bi imao i definiciju metode

getRezervaciju(). Metoda getRezervaciju() u klasi RezervacijaSaOsnovnimPodacima bi vraćala rezervaciju kod koje je slika = null, a u klasi RezervacijaSaSlikom bi vraćala rezervaciju koja sadrži i sliku. Metoda urediRezervaciju u klasi RezervacijaSaOsnovnimPodacima ne radi ništa, a u klasi RezervacijaSaSlikom se pomoću ove metode dodaje slika na tu rezervaciju.
Još jedna primjena ovog paterna se može ostvariti ukoliko dodamo mogućnost kreiranja željenog izgleda rezultata tretmana na sistemu (kreiranje slike za izgled noktiju), a također mogao bi se primijenti ukoliko bi dodali opciju za obavještavanje klijenta o njihovim terminima putem poruke/maila.

- *Bridge pattern*

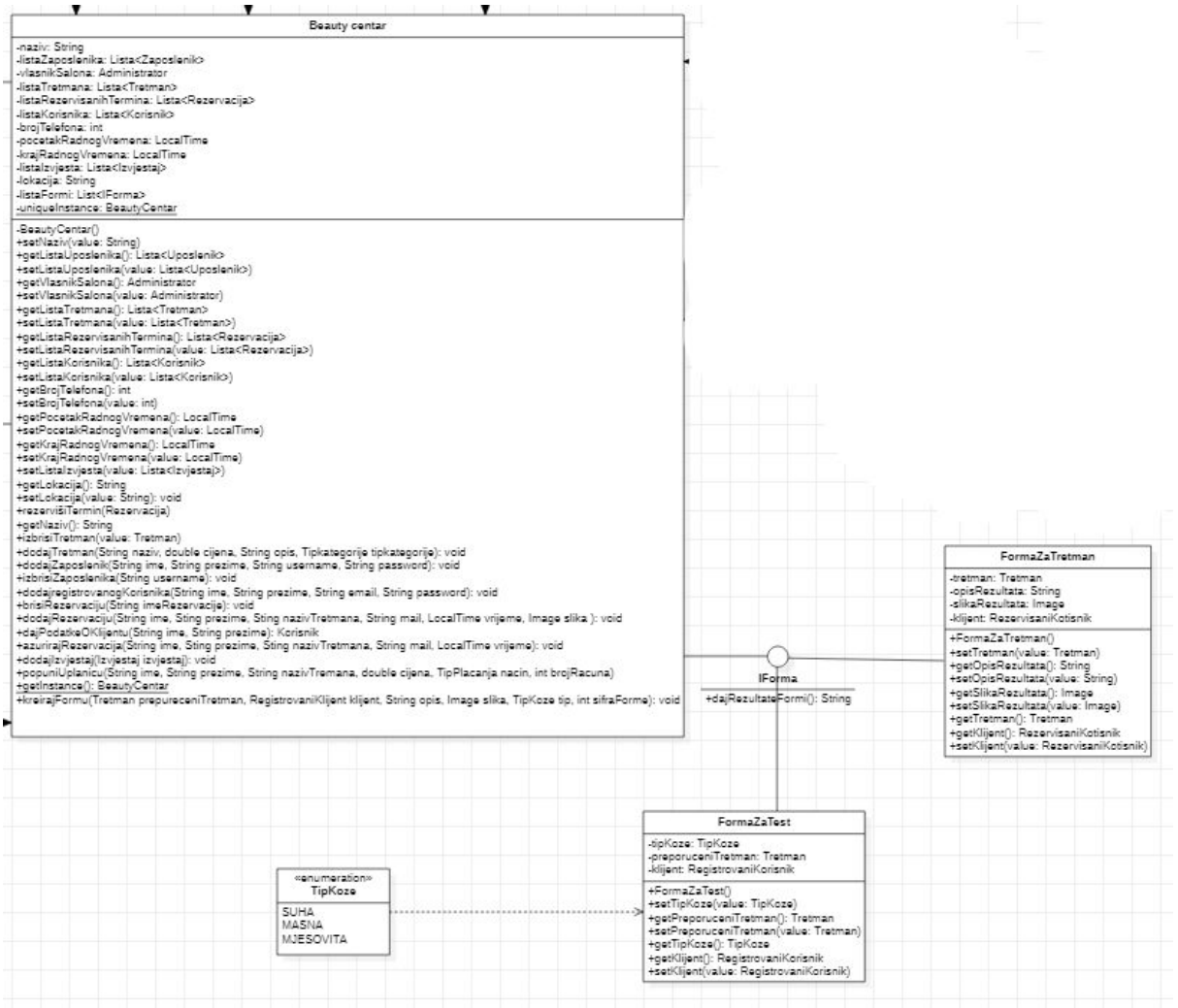
  Osnovna namjena ovog paterna je da se apstrakcija nekog objekta odvoji od njegove implementacije. Pattern nije iskorišten u našem sistemu.

  Ovaj patern bi mogli iskoristiti u našem sistemu ukoliko bismo dodali mogućnost obračunavanja plata za zaposlenike. Dakle, svi zaposlenici imaju istu osnovicu plate, ali im se ostatak plate računa prema vrsti posla.
  Ovo ne bi bilo povoljno za implementaciju u našem sistemu jer se sistem ne bavi kontrolom zaposlenika pa kao takav nema nikakvog razloga da računa platu. Ali ako bismo ipak željeli iskoristiti ovaj patern, onda bi to uradili na način da umjesto klase Zaposlenik dodamo klasu Bridge, koja će biti objašnjena u nastavku, te interfejs IPlata, dok bi klase Kozmetičar, Recepcioner i Administrator idalje postojale, i implementirale bi interfejs IPlata. Klasa Bridge bi sadržavala atribut tipa IPlata, te metodu dajPlatu(). Interfejs IPlata bi sadržavao definiciju metode izracunajPlatu(), a klase Kozmetičar, Recepcioner i Administrator bi sadržavale atribut koeficijent, te implementacije metode izracunajPlatu(). U metodi izracunajPlatu() ovih klasa bi se računala plata prema odgovrajućem koeficijentu, a onda u metodi dajPlatu() klase Bridge bi osnovicu koja je ista za sve sabirali sa rezultatom poziva metode izracunajPlatu() nad atributom klase Bridge.

- *Composite pattern*

  Osnovna namjena ovog paterna je da omogući formiranje strukture stabla pomoću klasa.
  Jedna primjena ovog paterna u našem sistemu bi mogla biti ukoliko odlučimo da galeriju dinamički implementiramo sa albumima i fotografijama za svaku kategoriju.

  Također još jedno pogodno mjesto za korištenje ovog paterna je realizacija dohvaćanja podataka iz formi. U tom slučaju navedene klase FormaZaTest i FormaZaTretman bi nasljeđivale interface IForma sa metodom dajPodatkeFormi() a u njima bi se implementirali njihovi načini realizacije iste. U ovisnosti o njihovim atributima tj. podaci o odradjenom testu ili tretmanu ce biti vraćeni kao String . Također BeautyCentar nasljeđuje interfejs IForma te implementira metodu dajPodatkeFormi() u kojoj ce kreirati jedan string sa svim podacima o svim formama, kako formama za test , tako i formama za tretman. Način na koji dobijemo sve te podatke u jednoj metodi jeste tako sto se u toj metodi u BeautyCentru za svaku formu koja se nalazi u listi formi koristi njima odovarajuća metoda dajPodatkeFormi() koju nasljedjuju iz interfejsa.



- *Flyweight patern*

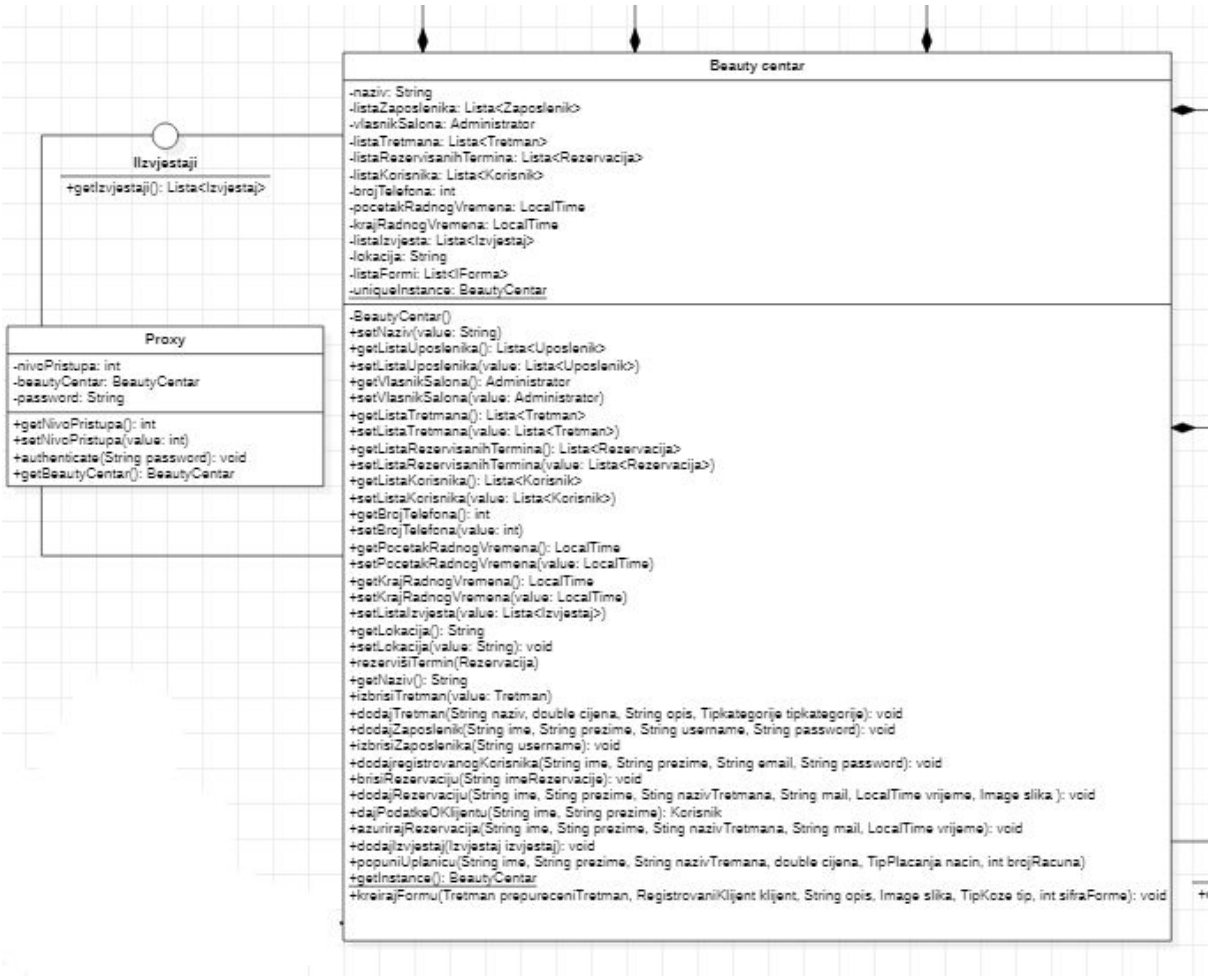
  Flyweight patern koristi se kako bi se onemogućilo bespotrebno stvaranje velikog broja instanci objekata koji svi u suštini predstavljaju jedan objekat. Ovaj pattern podrazumijeva posjedovanje bezličnog i specifičnog stanja.Pattern nije iskorišten u našem sistemu.

  Pogodno mjesto za korištenje ovog paterna je pri kreiranju korisnika ,ukoliko pri pokušaju rezervacije termina se ne nalazi u sistemu. Ovaj patern podrazumjeva posjedovanje bezličnog i specifičnog stanja. S obzirom da su nam potrebni podaci o specifičnom stanju svakog korisnika, potreban nam je interfejs koji ih vraća (IPodaci). U ovom slučaju to su jedinstveni podaci klijenta. U BeautyCentru više nemamo listu RegistrovanihKorisnika već listu

IPodataka te kad se pozove funkcija rezervišiTermin() poziva se neka metoda iz interfejsa IPodaci koja vraća string sa informacijama klijenta. U funkciji rezervisiTermin() ukoliko se podaci klijenta koji pokušava izvršiti rezervaciju poklapa sa nekim od postojećih podataka, onda se vraća jedna od postojećih instanci iz liste IPodaci a ako ne onda se dodaje nova instanca u listu.

- *Proxy pattern*

  Primjenom ovog paterna omogućava se kontrola pristupa objektima,te se onemogućava manipulacija objektima ukoliko neki uslov nije ispunjen, odnosno ukoliko korisnik nema prava pristupa traženom objektu.
  U našem projektu to ćemo iskoristiti za pristup izvještajima u sistemu , gdje će pravo pregleda svih izvještaja imati akter ukoliko je administrator, u suprotnom to neće biti moguće.
  Ovaj patern smo realizovali tako što smo u sistem dodali novu klasu Proxy i interfejs IIzvještaji.S obzirom da je u klasi BeautyCentar lista izvještaja koju želimo da zaštitimo od neautorizovanog pristupa, u klasu Proxy dodajemo instancu BeautyCentar klase te metodu pomoću koje vraćamo istu u zavisnosti od toga ko pristupa sistemu,getBeautyCentar() . Ukoliko se radi o osobi koja nije zaposlenik u sistemu, metoda getBeautyCentar() vraća null (nivo pristupa 3) , svim ostalim zaposlenicima osim admina se vraća instanca BeautyCentra bez liste izvještaja (nivo pristupa 2) dok jedno Administrator ima pristup i izvještajima u BeautyCentru(nivo pristupa 1). Metoda kojom provjeravamo nivo pristupa u sistemu je authenticate(String password), a kao atribut u klasi Proxy čuvamo password koji pripada administratoru sistema. Ukoliko password što se nalazi kao atribut se poklapa sa passwordom što je parametar ove funkcije,nivo pristupa je 1 (Administrator),ukoliko password koji dođe kao parametar funkcije sadži password što se nalazi kao atribut plus ima dodatne karaktere nakon,radi se o zaposleniku sistema koji nije Administrator,nivo pristupa je 2 (naknadno će se utvrditi u kojem tačno obliku će biti dati password) te ukoliko nije zadovoljen niti jedan od ova dva slučaja,nivo pristupa je 3.



Kreacijski patterni

- *Singleton pattern*

  Uloga ovog paterna je da se klasa može instancirati samo jednom i da osigura globalni pristup kreiranoj instanci klase. Potencijalna mjesta primjene ovog paterna u našem sistemu su prijava na sistem,pristup bazi podataka te osiguravanje da se klasa Beauty centar može samo jednom instancirati. U našem sistemu smo odlučile iskoristi ovaj patern za treći navedeni slučaj, tj.osiguravanje da se klasa BeautyCentar može samo jednom instancirati. To smo realizovale dodavanjem privatnog statičkog atributa uniqueInstance tipa BeautyCentar, zatim proglašavanjem konstruktora privatnim te dodavanjem statičke metode getInstance() koja vraća instancu klase BeautyCentar.

- *Prototype patern*

  Uloga ovog paterna je da kreira nove objekte klonirajući postojeći objekat. Ukoliko bi dodali opciju slanja letaka sa obavještenjima o aktuelnim ponudama i popustima na određene usluge, mogli bi iskoristiti ovaj patern za kreiranje istih. Pattern nije iskorišten u našem sistemu.

  Mjesto na kojem bi također ovaj pattern mogao biti iskorišten jeste za rezervacije članova salona tj. stalne klijente. Ukoliko salon ima određeni broj svojih stalnih klijenta on ima u listi rezervacija sacuvan broj rezervacija koje su rezervisane za njihove klijente , gdje su ustvari elementri te liste klonirani od jedne glavne istance za određenog klijenta. Dakle prilikom rezervacije tretmana bilo bi rezervisano mjesto za stalnog klijenta, te bi bilo potrebno samo da se promijene odredjeni podaci te iste rezervacije.

Potrebno bi bilo dodati interfejs IRezervacija() koji mi imao metodu kloniraj(). Ovaj interface bi trebao biti naslijeđen od strane klase Rezervacija () koja će implementirati metodu kloniraj() kako bi bila omogućena kopija instance. Ova metoda bi bila iskoristena prilokom rezervacije ,tj. prilikom dodavanja rezervacije u listu IRezervacija koja se nalazi u klasi BeautyCentru te bi prilikom dodavanja rezervacije u metodi dodajRezervaciju(Rezervacija r) mogla biti iskoristena metoga kloniraj() gdje bi eventualno doslo do promijena nekih atributa klijenta kaosto su npr. vrijeme retervacije ili mozda odabrani tetman.

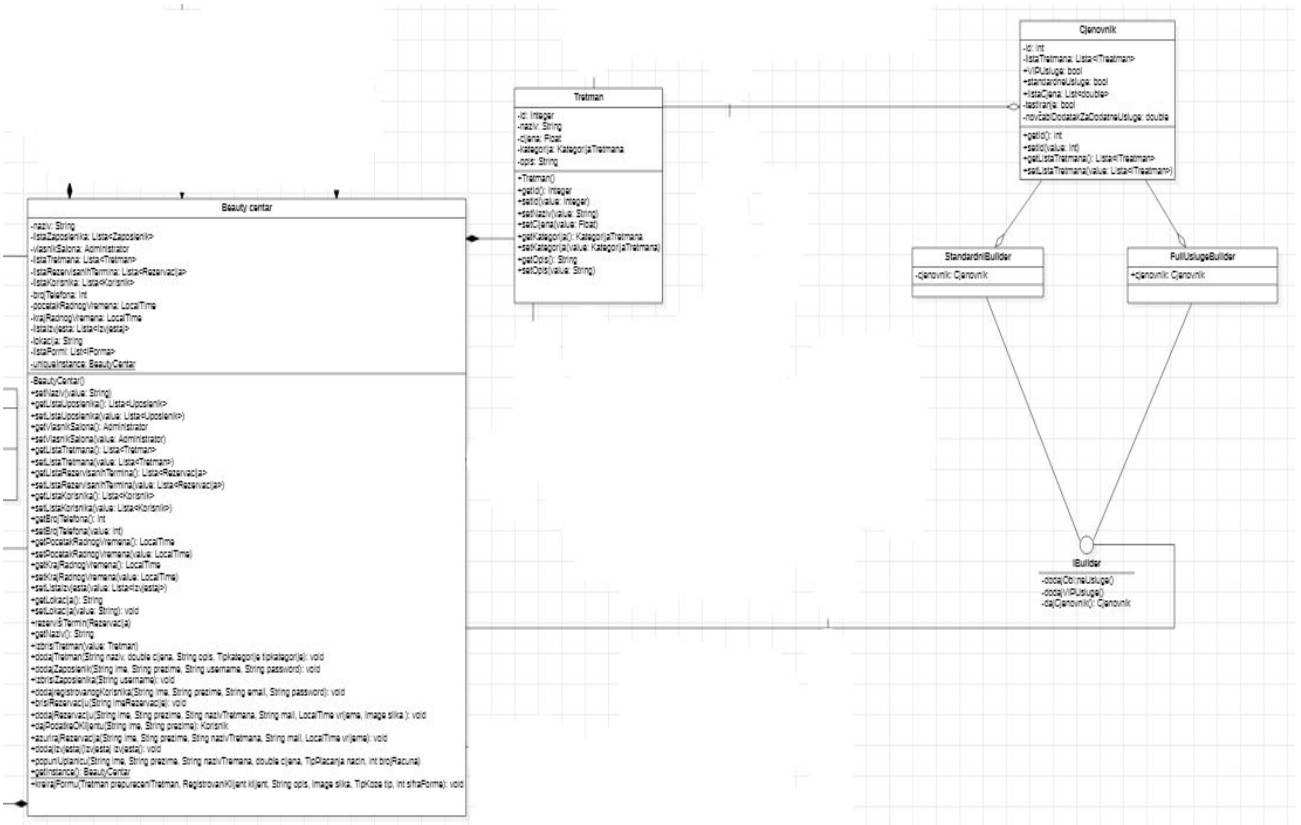
- *Factory Method patern*

Uloga ovog paterna je da omogući kreiranje objekata na način da podklase odluče koju klasu instancirati.Pattern nije iskorišten u našem sistemu.
Potencijalna primjena ovog paterna u našem sistemu bi bila ukoliko bi uvele prodaju kozmetičkih preparata u poslovnici. U tom slučaju naručivanje proizvoda bi se realizovalo preko različitih dobavljača ali isporuka bi uvijek bila na isti način. Također ukoliko bi se dodala funkcionalnost da se na sistemu vodi evidencija o stanju robe i materijala mogli bi iskoristiti ovaj pattern.

Da smo odlučile da ovaj pattern primijenimo u nas sistem ono što bi bilo potrebno dodati bit će objasnjeno u nastavko.
Potrebno bi bilo napraviti klasu NarudzbaPreparata, koja bi imala atribut kao što je primjenaPreparata cij je tip apstraktna klasa PrijmenaPreparata koju nasljeđuju klase PreteratZaKosu , klasa PreparatZaLice, klasa PreparatZaTijelo. Takodjer jedan od atributa bi bio tipProizvodjača cij je tip ustvari abstraktna klasa ProizvođacPreparata koju nasljedjuju klase PrirodniProizvodjac i MixProizvodjac (proizvodjac koji ne radi sve na prirodnoj bazi) .
Ono sto je jos potrebno dodati jeste interfejs INarudzba koji ce imati metodu napraviNarudzbu(). Interfejs će biti naslijeđen od strane klase NarudzbaZaKosu() , klasa NarudzbaZaLice, klasa NarudzbaZaTijelo koje naslijediti klasu Narudzba i njezine atribute, te će implementirati klasu napraviNarudzbu() na način koji ce napraviti instancu narudžbe koja je potrebna.

- *Builder patern*

  Uloga ovog paterna je odvajanje specifikacije kompleksnih objekata od njihove stvarne konstrukcije.
  U našem sistemu bi se ovaj patern mogao realizovati ukoliko bi se odlučili da dinamički kreiramo različite vrste cjenovnika.Realizacija ove ideje bi se omogućila ukoliko dodamo klasu Cjenovnik koja kao atribute ima sve karakteristike i opise koje bi jedan cjenovnik trebao da ima . U klasi Cjeovnik imamo listu tretmana i njihove cijene koje se odreduju na osnovu ostalih atributa u ovisnosti kako su incijalicirani, tj. da li je su ukljucene vip uspuge ili su ukljucene standardne usluge, da li je ukljuceno testiranje , pregled glijenta itd. Potrebno je dodati i interfejs IBuilder koji ima metode za različite dijelove izgradnje cjenovnika. Kao primjer možemo uzeti metode dodajObicneUsluge(), dodajVipUsluge() koje ce popunjavati atribute cjenovnika na određen način .U našem sistemu smo dodale klasu StandardBuilder i FullUslugeBuilder koje kao svoj atribut imaju Cjenovnik i koje koriste različite kombinacije metoda,tj.grade različite cjenovnike.Ove klase implementiraju metode interfejsa koje ce na razlicit nacin incijaliziraci atribute Cjenovnika ,ti atributi ce uzimati razlicite vrijednosti atributa u ovisnosti o kojoj vrsti cjenovnika se radi.



- *Abstract factory patern*

  Ovaj patern omogućava da se kreiraju familije povezanih objekata/produkata. Na osnovu apstraktne familije produkata kreiraju se konkretne fabrike različitih tipova i različitih kombinacija.Pattern nije iskorišten u našem sistemu.
  Primjena ovog paterna u našem sistemu bi mogla biti npr. pri rezervaciji termina. Najprije bismo kreirali različite klase korisnika, koje bismo naslijedili iz klase RegistrovaniKorisnik. To mogu biti kategorije korisnika prema njihovim godinama. Npr. klase KorisniciIzmedjuPetnaestIDvadesetPetGodinaFactory, KorisniciIzmedjuDvadestPetITridestPetGodinaFactory, KorisniciIzmedjuTridestPetIPedesetGodinaFactory, itd.
  Svakoj od navedenih kategorija bismo omogućili rezervaciju različitih tretmana.
  Kreiranje familije povezanih objekata bi počeli sa osnovnim tipom tretmana te bi se dalje dijelili u klase i kreirale bi se razne kombinacije istih. Npr.
  kreirali bismo apstraktne klase Manikir, Pedikir, TretmanZaTrepaviceIObrve, Depilacija, itd.
  Kao podklase klase Manikir bismo imali klase OsnovnaManikura, DizajnILakiranjeNoktiju, French, itd.
  Kao podklase klase Pedikir bismo imali klase OsnovniPedikir, KraljevskiPedikir, FrenchLakiranjeNoktiju, itd.
  Kao podklase klase TretmanZaTrepaviceIObrve bismo imali klase FarbanjeTrepavica, FarbanjeObrva, KorekcijaObrva, itd.
  Kao podklase klase Depilacija bismo imali klase DepilacijaRuku, DepilacijaNogu, itd.
  Također bismo kreirali interfejs IFactory koji bi sadržavao definicije metoda:
  -getManikir(): Manikir
  -getPedikir(): Pedikir
  -getTretmanZaTrepaviceIObrve(): TretmanZaTrepaviceIObrve
  -getDepilaciju(): Depilacija
  Klase koje predstavljaju kategorije korisnika bi implementirale ovaj interfejs.
  A onda bismo u klasu KorisniciIzmedjuPetnaestIDvadesetPetGodinaFactory dodali atribute tipa French, FrenchLakiranjeNoktiju, KorekcijaObrva, DepilacijaNogu.
  U klasu KorisniciIzmedjuDvadestPetITridestPetGodinaFactory bismo dodali atribute tipa DizajnILakiranjeNoktiju, KraljevskiPedikir, FarbanjeObrva, DepilacijaRuku.
  U klasu KorisniciIzmedjuTridestPetIPedesetGodinaFactory bismo dodali atribute tipa OsnovnaManikura, OsnovniPedikir, FarbanjeTrepavica, DepilacijaNogu.
  U svaku od ovih klasa bismo dodali i gettere za ove njihove atribute.