KREACIJSKI PATERNI

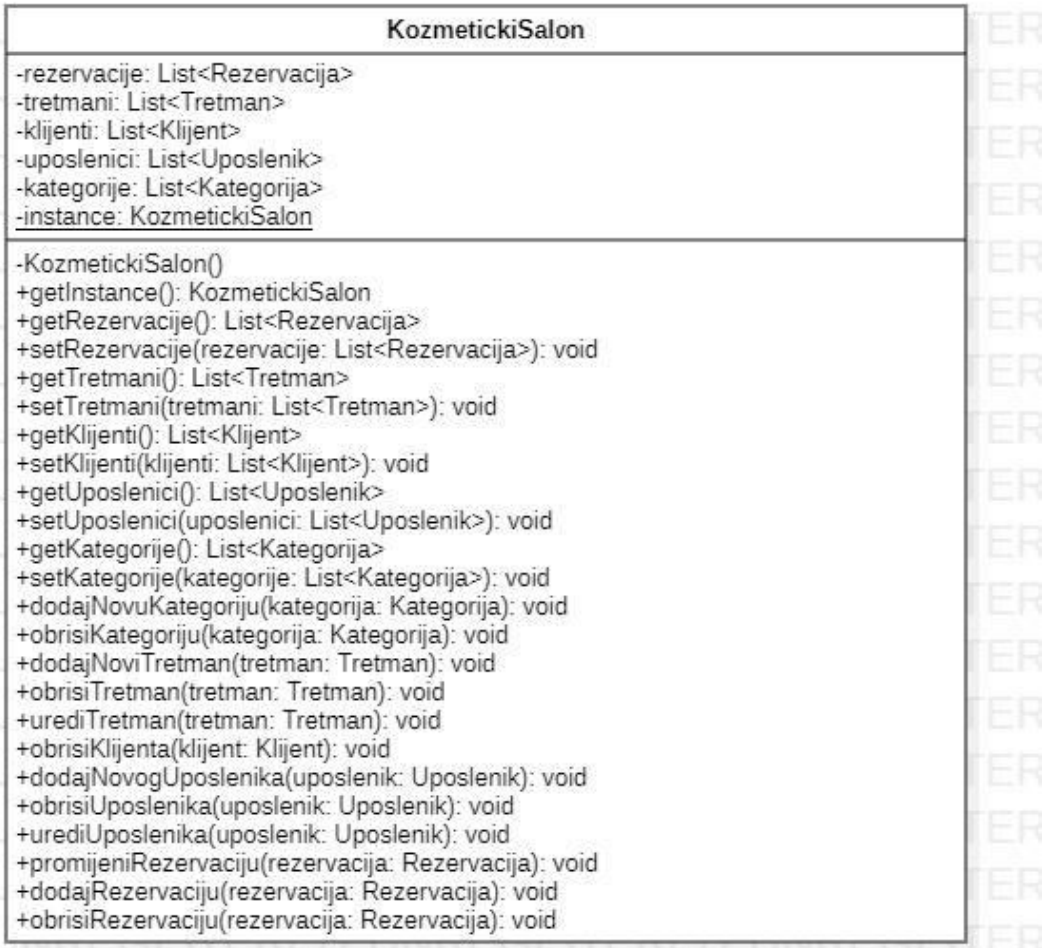
1. Singleton pattern

Singleton patern služi kako bi se neka klasa mogla instancirati samo jednom.

Ovaj patern ćemo primijeniti na našu kontejnersku klasu KozmetickiSalon, tako što ćemo dodati statički privatni atribut instance tipa KozmetickiSalon, privatni konstruktor bez parametara, te metodu getInstance( ).

Razlog zbog kojeg smo primijenile ovaj patern u sistemu je da bismo izbjegli konflikte u slučaju da administrator i uposlenik pristupaju u istom trenutku podacima i modifikuju ih.

Pri prvom pozivu metode getInstance( ) kreiramo objekat tipa KozmetickiSalon pozivom privatnog konstruktora, a pri svakom sljedećem pozivu te metode, ona vraća taj već kreirani objekat.



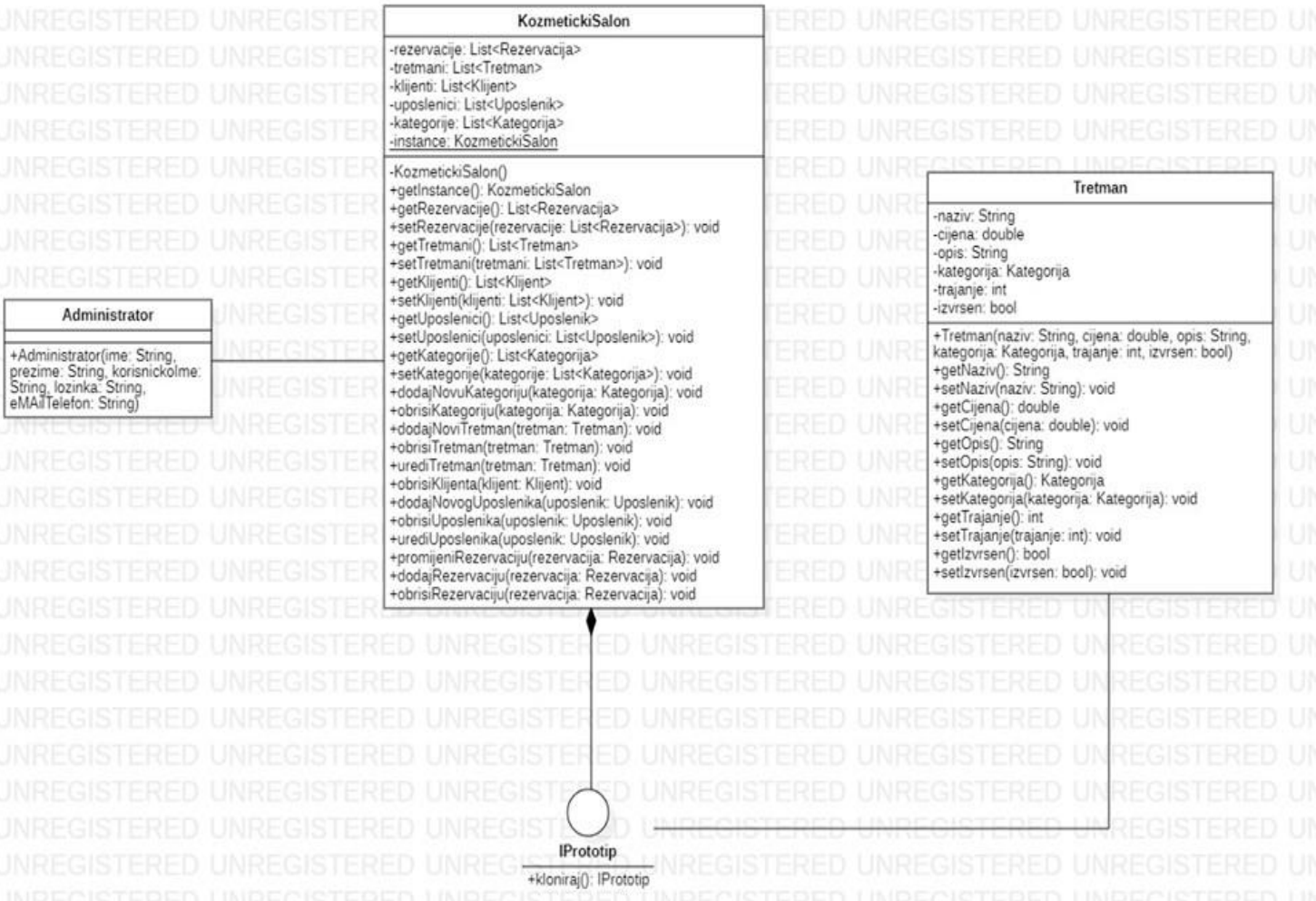
2. Prototype pattern

Prototype patern omogućava smanjenje kompleksnosti kreiranja novog objekta tako što se uvodi operacija kloniranja.

Svi akteri u našem sistemu su jedinstveni, pa samim tim se ne mogu klonirati. Međutim, ovaj patern smo primijenili za kloniranje tretmana uvođenjem interfejsa IPrototip, koji sadrži metodu kloniraj(). Ova metoda će kreirati novi tretman sa istim atributima kao i tertman nad kojim je pozvana ova metoda.

Klasa Tretman implemetira ovaj interfejs, a interfejs je sa klasom KozmetickiSalon spojen kompozicijom (jer su, prvobitno, klase KozmetickiSalon i Tretman bili spojeni kompozicijom).

Razlog zbog kojeg smo primijenili ovaj patern je što administrator prilikom dodavanja novog tretmana, koji ima slične osobine kao neki već postojeći tretman, nema potrebe za kreiranjem novog tretmana nanovo od početka, nego može klonirati već postojeći tretman i izmijeniti ga.



3. Builder pattern

Builder patern služi za apstrakciju procesa konstrukcije objekta, kako bi se kao rezultat mogle dobiti različite specifikacije objekta koristeći isti proces konstrukcije.

Ovaj patern ne možemo primijeniti u našem sistemu, ali kada bismo imali klasu OnlineVisitKartica koja bi kao atribute imala naziv, lokacija, brojTelefona, eMail, vrstaPosla, itd. imalo bi smisla primijeniti navedeni patern. Ako bi se pravila vizit kartica za uposlenika morao bi se postaviti i atribut vrstaPosla, a ako pravimo vizit karticu za kozmetički salon taj atribut ne bi imao smisla.

Da bismo konstruktor klase OnlineVisitKartica učinili jednostavnijim dodali bismo dvije nove klase: ManuelnaVisitKartica i AutomatskaVisitKartica, te interfejs IBuilder, kojeg implemetiraju navedene klase. Novokreirane klase bi imale atribut tipa OnlineVisitKartica, a interfejs bi imao metode pomoću kojih bi se inicijalizirali atributi klase OnlineVisitKartica(npr. void PostaviKontakte(String eMail, String brojTelefona)), te metodu dajOnlineVisitKartica(). Uloga klase AutomatskaVisitKartica je da pomoću metoda iz interfejsa IBuilder kreira vizit karticu sa podrazumijevanim vrijednostima za date atribute klase OnlineVisitKartica(npr. naziv = „Kozmetički salon“, vrstaPosla = “ “, ...), dok ManuelnaVisitKartica inicijalizira atribute na vrijednosti koje su parametri metoda interfejsa IBuilder(namijenjena za pravljenje vizit kartica uposlenicima).

Veza između klasa OnlineVisitKartica i ManuelnaVisitKartica je agregacija, kao i između klasa OnlineVisitKartica i AutomatskaVisitKartica. Klasa Uposlenik implementira interfejs IBuilder.

4. Abstract Factory patern

Abstract factory patern služi kako bi se izbjeglo korištenje velikog broja if-else uslova pri kreiranju različitih hijerarhija objekata.

U našem sistemu nije bilo potrebe za uvođenjem ovog paterna. Recimo da smo imali složeniji sistem, i da želimo da sastavimo pakete tretmana za Klijent, PosebanKlijent, VIPKlijent. Sada pretpostavimo da imamo više klasa koje predstavljaju tretmane, a razlikuju se po kategoriji (radi jednostavnosti zamislimo da imamo 2 klase: TretmaniLica - iz koje su izvedene klase CiscenjeLica, AntiAgingTretmanLica, VitaminskiTretman i TretmaniKose – iz koje su izvedene klase KraljevskiTretman, KeratinskiTretman, ObicniTretman). Vip-klijentu se pravi paket sastavljen od KraljevskiTretman i VitaminskiTretman, posebnom klijentu paket sastavljen od KeratinskiTretman i AntiAgingTretmanLica, a klijentu se pravi paket koji se sastoji od ObicniTretman i CiscenjeLica.
Za realizaciju Abstract Factory paterna potrebno je dodati interfejs IPaketFactory u kojem se nalaze metode dajTretmanLica() i dajTretmanKose(), te klase:

a. KlijentFactory (atributi: tretmanLica tipa VitaminskiTretman i tretman kose tipa KraljevskiTretman),
b. PosebanKlijentFactory (atributi: tretmanLica tipa AntiAgingTretmanLica i tretman kose tipa KeratinskiTretman)
c. VIPKlijentFactory (atributi: tretmanLica tipa CiscenjeLica i tretman kose tipa ObicniTretman)

Sve novododane klase implementiraju interfejs IpaketFactory.
Na ovaj način izvršili bismo kreiranje paketa za različite klijente, a pri tome izbjegli brojne if-else provjere u okviru jedne metode (koja bi se, da naš sistem posjeduje takve mogućnosti, nalazila u klasi Rezervacija).

5. Factory Method patern

Factory method patern služi za omogućavanje instanciranje različitih vrsta podklasa koristeći factory metodu koja odlučuje koja će se podklasa instancirati i koja programska logika izvršiti

Patern nije iskorišten u našem sistemu, za slučaj da jeste, morali bismo zamisliti sljedeći scenarij.
Administrator na kraju svakog mjeseca ima izvještaj koji mu daje podatke o radu uposlenika kozmetičkog salona. Svi izvještaji do sada kreirani su koristeći AnketaKlijenata (klijent nakon izvršenog tretmana u aplikaciji ima mogućnost da popuni kratku anketu o usluzi uposlenika koji je vršio tretman) i GodisnjaStatistika (prati se promet klijenata u kozmetičkom salonu, broj zakazanih, otkazanih i izvršenih tretmana).
Administrator je izrazio želju da se sada izvještaj kreira koristeći AnketaNovihKlijenata (anketiraju se klijenti koji imaju izvršen jedan tretman) i MjesecnaStatistika.
Potrebne klase:

a. Izvjestaj (atributi: statistika(Statistika), anketa(Anketa))
b. Statistika (izvedene klase: GodisnjaStatistika, Mjesecna Statistika)
c. Anketa (izvedene klase: AnketaKlijenata, AnketaNovihKlijenata)

Da bi realizovali factory method patern potrebno je dodati: interfejs IIzvjestaj (sadrži metodu napraviIzvjestaj()), te klase StariIzvjestaj i NoviIzvjestaj koje su izvedene iz klase Izvjestaj, a implementiraju interfejs IIzvjestaj.
Na ovaj način bilo bi omogućeno kreiranje izvještaja korištenjem metode napraviIzvjestaj koristeći različite podklase atributa iz klase Izvjestaj, a izbjegle bi korištenje različitih metoda (koje bi se nalazile u klasi Izvjestaj) za kreiranje različitih izvještaja.