KREACIJSKI PATERNI

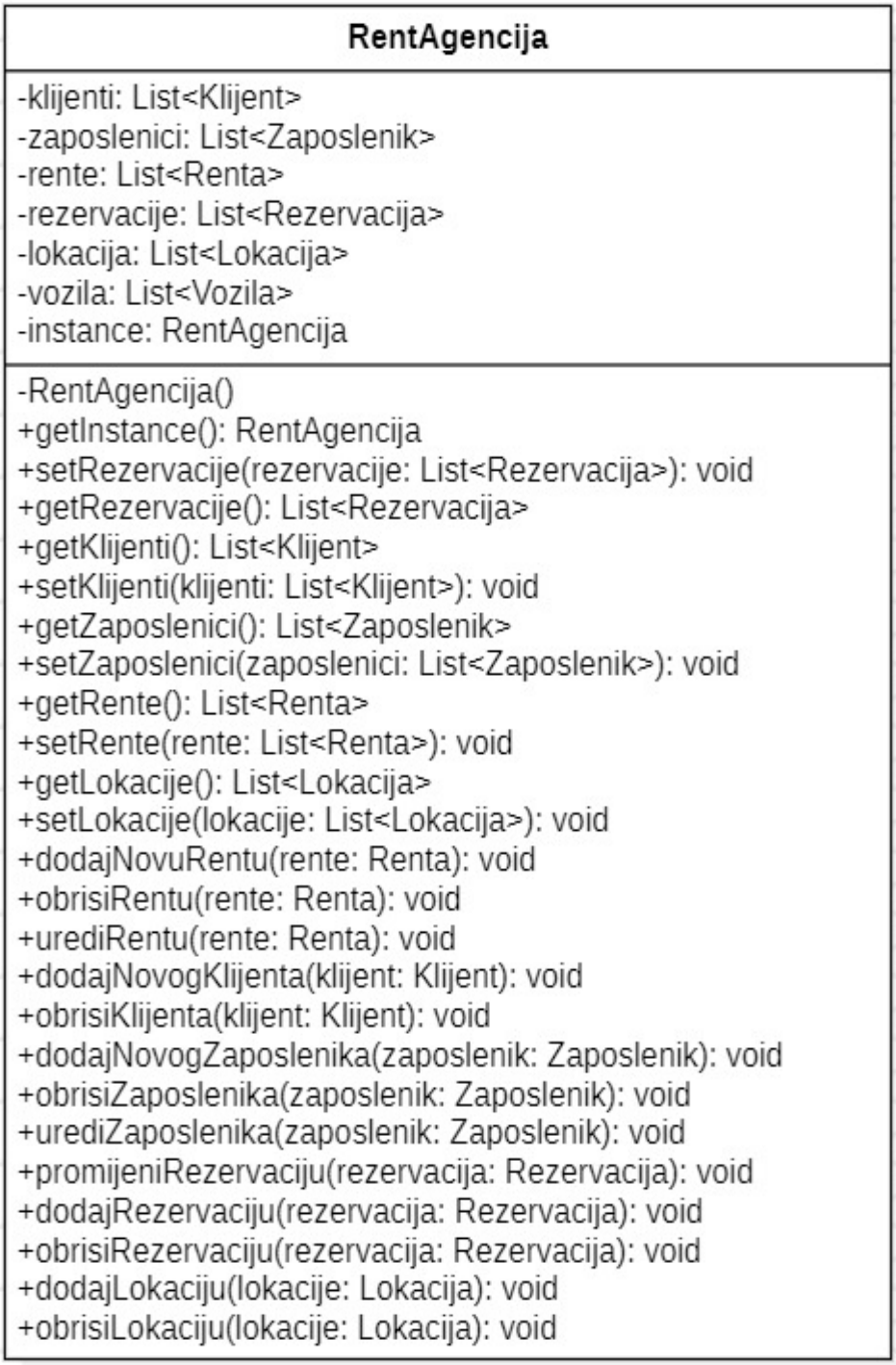
1.Singelton pattern

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

Ovaj patern ćemo primjeniti na našu kontejnersku klasu RentAgencija, tako što ćemo dodati privatni atribut tipa RentAgencija, privatni konstruktor bez parametara,te metodu getInstace().

Razlog primjenjivanja ovog paterna u sistemu je da bismo izbjegli konflikt u slučaju da administrator i uposlenik pristupaju u istom trenutku podacima i da ih modifikuju.

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



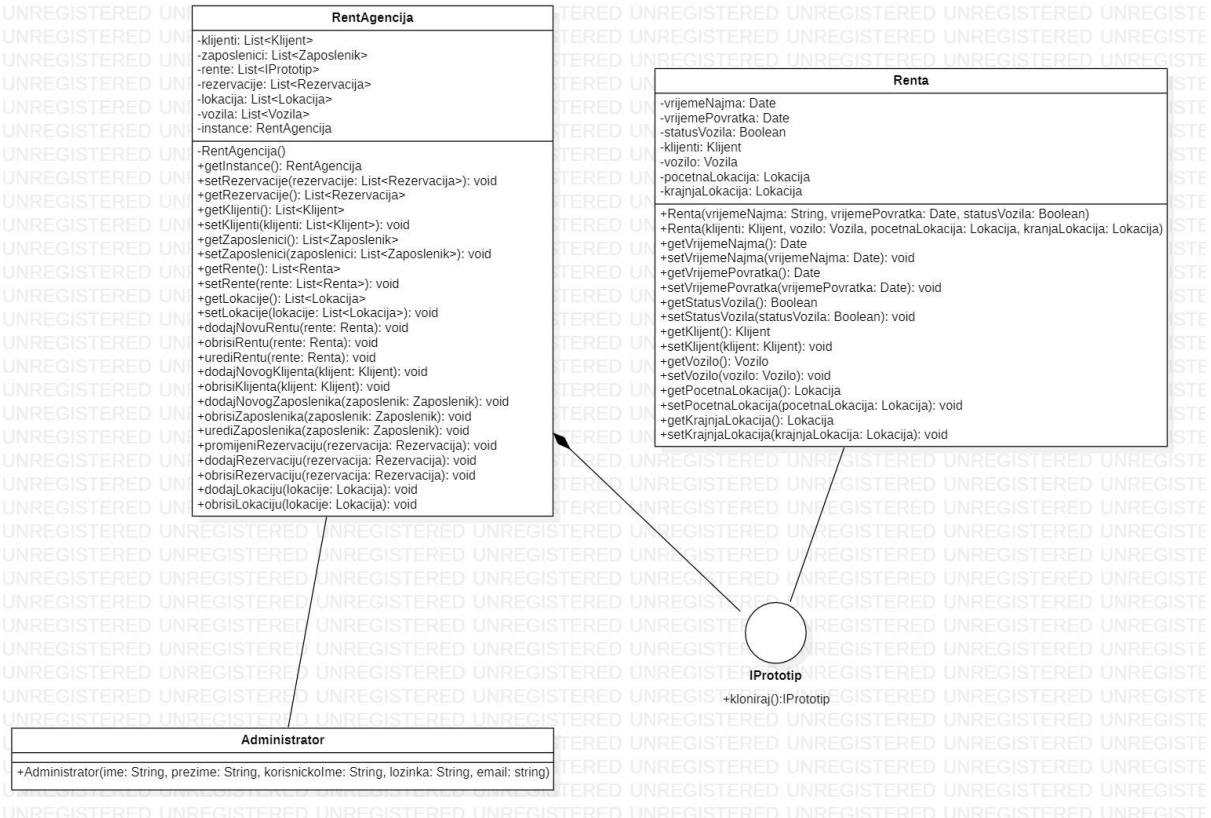
2.Prototype pattern

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

Svi akteri u našem sistemu su jednistveni, pa samim tim se ne mogu klonirati.Međutim ovaj patern smo primjenili za kloniranje rente uvođenjem interfejsa Iprototip, koji sadrži metodu kloniraj().Ova metoda će kreirati novo preuzimanje auta sa istim atributima kao i preuzimanje auta nad kojim je pozvana ova metoda.

Klasa Renta implementira ovaj interfejs , a interfejs je sa klasom RentAgencija spojen kompozicijom (jer su ,prvobitno, klase RentAgencija i Renta bili spojeni kompozicijom).

Razlog zbog kojeg smo primjenili ovaj patern je što administrator prilikom dodavanja novog rentanja koji ima slične osobine kao neki već postojeci tip rentanja pa nema potrebe za kreiranjem novog rentanja nanovo od početka , nego može klonirati već postojeći Renta 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 primjeniti na naš sistem, ali kada bismo imali klasu OnlineVisitKartice koja bi imala atribute naziv,lokacija,brojTelefona,email,vrstaPosla itd.,imalo bi smisla primjeniti navedeni patern.Ako bi se pravila vizit kartica za uposlenika morao bi se postaviti i atribut vrstaPosla , a ako pravimo vizit karticu za rent agenciju taj atribut ne bi imao smisla.

Da bismo konstruktor klase OnlineKartica učinili jednostavnijim dodali bi dvije nove klase : ManuelnaVisitKartica i AutomatskaVisitKartica, te interfejs IBuilder,kojeg implementiraju navedene klase.Novokreirane klase bi imale atribut tipa OnlineVisitKartica(npr. Void PostaviKontakte(String email,string brojTelefona),te metodu dajOnlineVisitKarticu. Uloga klase AutomtaskaVisitKartica je da pomoću metoda iz interfejsa IBuilder kreira vizit karticu sa podrazumijevanim vrijednostima za date atribute klase OnlineVisitKartica(npr. naziv=“Rent a car Agencija“...),dok ManuelnaVisitKartica incijalizira 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 AutomtskaVisitKartica. Klasa Uposlenik implementira interfejs IBuilder.

4.Abstract Factory patern

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

U našem sistemu nije bilo potrebe za uvođenjem ovog paterna. Recimo samo da smo imali složeniji sistem, i da želimo da sastavimo posebne pakete izdavanja auta za Klijent, PosebanKlijent, VIPKlijent. Sada pretpostavimo da imamo više klasa koje predstavljaju rente a, razlikuju se po nekim luksuznijim vozilima.Vip klijentu se pravi paket LuksuznaRenta i ExtraLuksuznaRenta, Posebnom klijentu se pravi paket PosebnaRenta i PorodicnaRenta, a klijentu se pravi paket koji sadrži ObicnaRenta i DvodnevnaRenta.

Za realizaciju Abstract factory paterna potrebno je dodati interfejs IpaketFactory te klase koje implementiraju intefejs IpaketFactory.

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

5.Factory Method patern

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

Patern nije korišten u našem sistemu,za slučaj da jeste , morali bismo zamisliti sljedeći scenarij.

Administrator na kraju svakog mjeseca ima izvještaj koji mu daje na uvid podatke o radu uposlenika rent-a-car agencije. Svi izvještaji do sada kreirani su koristeći AnketaKlijenata(klijent koji nakon završenog rentanja automobila u aplikaciji ima mogućnost da popuni kratku anketu o usluzi uposlenika koji je vršio tretman) i GodisnjaStatistika(prati se promet klijenata u rent-a-car agenciji, broj rentanja ,broj otkazivanja itd.)

Administrator je izrazio želju da se sada izvještaj kreira koristeći AnketaNovihKlijenata(anketiraju se klijenti koji imaju izvršeno jedno rentanje automobila) i MjesecnaStatistika.

Potrebne klase:

a. Izvjestaj(atributi:statistika(Statisktika),anketa(Anketa))

b. Statistika(izvedene klase:GodisnjaStatistika,MjesecnaStatistika)

c. Anketa(izvedene klase:AnketaKlijenata,AnketaNovihKlijenata)

Da bi realizovali facotry method patern potrebno je dodati: intefejs Iizvjestaj(sadrzi metodu napraviIzvjestaj()), te klase StariIzvjestaj i NoviIzvjestaj koje su izvedene iz klase Izvjestaj , a implementiraju intefejs Iizvjestaj.

Na ovaj način bilo bi omogućeno kreiranje izvještaja korištenjem metode napraviIzvjestaj koristeći različite podklase Atributa iz klase Izvjestaj,a izbjegli bi korištenje različitih metoda(koje bi se nalazile u klasi izvjestaj) za kreiranje različitih izvještaja.