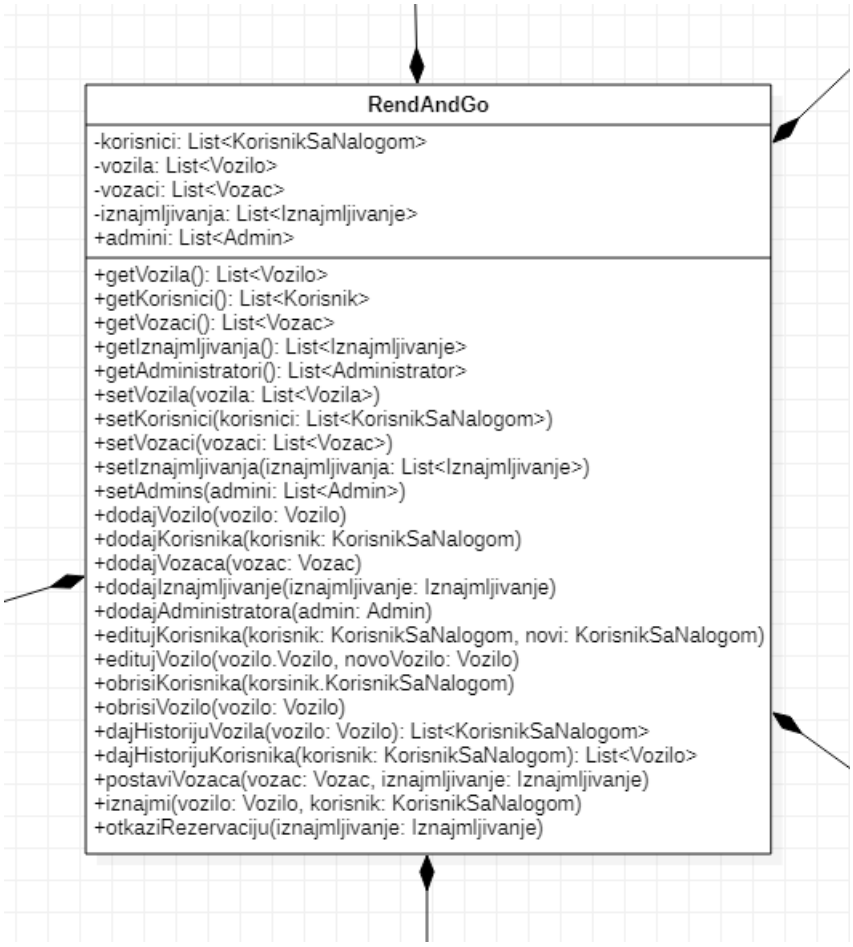
Kreacijski paterni

1.SINGLETON

Uloga singleton paterna je da osigura da se klasa može instancirati samo jednom i da osigura globalni pristup kreiranoj instanci klase. Osigurava se globalni pristup jedin
stvenoj instanci, svaki put kada joj se pristupi dobit će se ista instanca klase. Ovaj patern mogao bi se primijeniti na klasu RentACar, jer bi jedino ispravno bilo da imamo samo jednu instancu ove klase koja bi u biti činila naš sistem, jer kada bi imali više instanci ove klase dolazilo bi do konflikata.



2.PROTOTYPE

Prototype patern nam omogućava smanjenje kompleksnosti kreiranja novog objekta tako sto se uvodi operacija kloniranja. Na taj se način prave prototipi objekata koje je moguće replicirati više puta a zatim naknadno primijeniti jednu ili vise karakteristika, bez potrebe za kreiranjem novog objekta nanovo od početka. Ovime se osigurava pojednostavljenje procesa kreiranja novih instanci, posebno kada objekti sadrže veliki broj atributa koji za većinu instanci isti. U našem projektu bismo mogli implementirati ovaj patern tako što ćemo dodati interface *IPrototip*, koji ima metodu *kloniraj*. Ovim interfejsom smo smanjili kompleksnost kreiranja novog objekta, a implementira ga klasa Vozilo.

3.FACTORY METHOD

Priliku za implementaciju ovog paterna smo uocile kod klase Iznajmljivanje. Kako je taj dio već ranije isplaniran, odlučile smo za sada ne primjenjivati ovaj patern. U slučaju korištenja ovog paterna, implementacija bi bila sljedeca: prvi bi napravile klasu Kreator kojoj bi na neki način rekle koja je vrsta iznajmljivanja (sa ili bez vozača), a zatim bi nam ona vratila instancu tog zahtjeva. Nakon toga, nekom polimorfnom metodom popuniZahtjevZaIznajmljivanje, koja bi se nalazila u klasi Iznajmljivanje, bi se tačno znalo o kojoj vrsti iznajmljivanja je riječ i način na koji se popunjava.

4.ABSTRACT FACTORY

Uloga Abstract Factory paterna je da se kreiraju familije povezanih objekata/produkata. Na osnovu apstraktne familije produkata kreiraju se konkretne fabrike produkata različitih tipova i različitih kombinacija. Važan aspekt Abstract Factory paterna je da se cijela familija produkata može promijeniti za vrijeme izvršavanja aplikacije zbog toga što produkti implementiraju isti apstraktni interfejs. S obzirom da u našem sistemu nemamo familija produkata koji bi se mogli izvesti iz jednog interfejsa, abstract factory nećemo primjenjivati u našem sistemu. Da smo imali složeniji sistem, mogli bi da ga iskoristimo tako da postoje više vrsta vozila kao što su LuksuznoVozilo, EkonomskoVozilo itd., također mogli bi smo mogli imati više vrsta iznajmljivanja koja bi se odnosila na osobu koja iznajmljuje što bi na primjer bilo IznajmljivanjeZaStraneDržavljane, IznamjmljivanjeZaDržavljane itd.

5.BUILDER PATERN

Builder patern služi za apstrakciju procesa konstrukcije objekta, kako bi se kao rezultat mogle dobiti različite specifikacije objekata koristeći isti proces konstrukcije. Ovaj patern se koristi kako bi se izbjeglo kreiranje kompleksnije hijerarhije klasa te kako bi se izbjegao kompleksni programski kod kontsruktora jedne klase koja može imati različite konfiguracije atributa. Različiti dijelovi konstrukcije objekata izdvajaju se u posebne metode koje se zatim pozivaju različitim predosljedom ili se poziv nekih dijelova izostavlja, kako bi se dobili željeni različiti podtipovi objekata bez potrebe za kreiranjem velikog broja podklasa. Ovaj patern možemo implementirati na način da kada korisnik obavlja pretragu automobila, u aplikaciji bira grad iz kojeg želi iznajmiti auto, te bi bila potrebna klasa kojiGradBuilder, a ta klasa bi implementirala IBuilder intefejs te bi se generisali dostupni automobili u tom gradu.