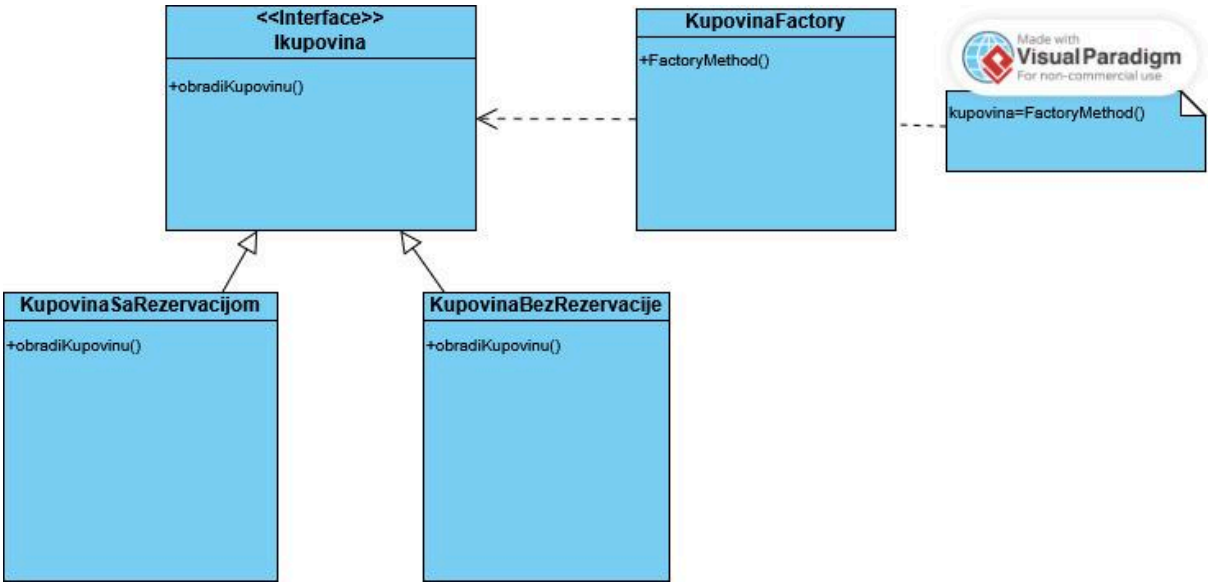
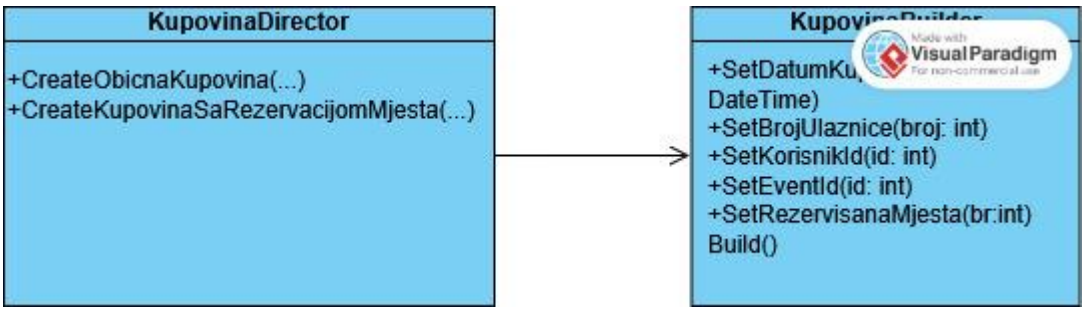
Factory Method pattern

U ovom rješenju primijenjen je Factory Method kreacioni pattern s ciljem da se odvoji logika kreiranja objekata kupovine od samog kontrolera i ostatka aplikacije. Time se postiže fleksibilnost, proširivost i slaba povezanost. Kontroler ne zna ništa o konkretnim klasama kupovine, on koristi samo KupovinaFactory i interfejs IKupovina. U KupovinaFactory imamo metodu FactoryMethod koja donosi odluku koju konkretnu klasu kupovine treba instancirati.



Builder pattern

Kupovina ima više polja i mogu postojati različiti načini kreiranja objekata (npr. sa ili bez rezervacije mjesta), te samim tim korištenje klasičnog konstruktora postaje nepregledno i podložno greškama. Zato se koristi Builder pattern koji omogućava fleksibilno i čitljivo konstruisanje objekata. Builder (KupovinaBuilder) omogućava postavljanje pojedinačnih polja objekta preko imenovanih metoda (SetDatumKupovine(...), SetBrojUlaznice(...) itd.). Završava se pozivom Build(), koji vraća potpuno kreiran objekat klase Kupovina. Omogućava različite reprezentacije istog objekta, ovisno o tome koje metode su pozvane (npr. Kupovina sa ili bez rezervisanih mjesta). Objekat Kupovina je uvijek isti sa strane njegove reprezentacije, ali način kako se do njega dolazi je odvojen i fleksibilan.



Singleton pattern

Singleton pattern mogao bi se primijeniti na klasu DataContext zato što treba postojati jedinstvena, dosljedna tačka pristupa svim DbSet kolekcijama u cijeloj aplikaciji.

Prototype pattern

Prototype pattern bi se mogao primijeniti na klasu Event jer ponekad treba duplicirati već postojeći događaj (isti naziv, lokacija, opis) i zatim promijeniti samo neke atribute, poput datuma ili vremena.

Abstract Factory pattern

Nije moguće primjeniti zbog nedostatka povezanih klasa koje se kreiraju zajedno, postoji samo po jedna konkretna klasa za Event, Kupovinu, Bookmark…