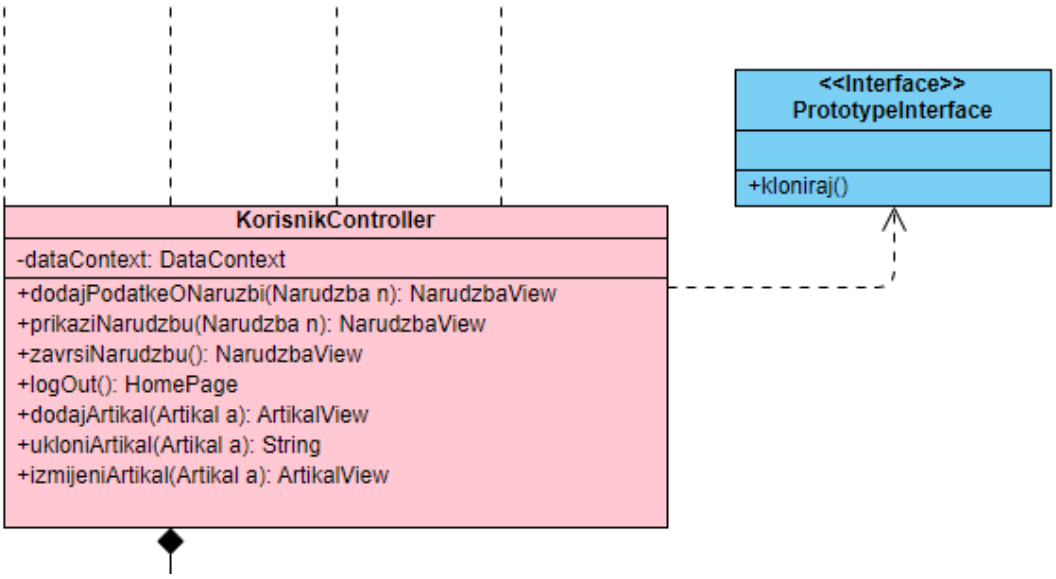
Kreacijski paterni

1. ***Singleton patern*** – osigurava da se klasa može instancirati samo jednom, te globalni pristup kreiranoj instanci klase. Recimo, ako bismo imali samo jednog administratora sistema, ovaj patern bi nam osigurao da može postojati samo jedna instanca ove klase unutar aplikacije. Sve funkcionalnosti koje administrator ima bile bi ograničene na tu jednu instancu klase, samo bismo unutar klase Administrator morali odvojeno implementirati sve potrebne metode i atribute.

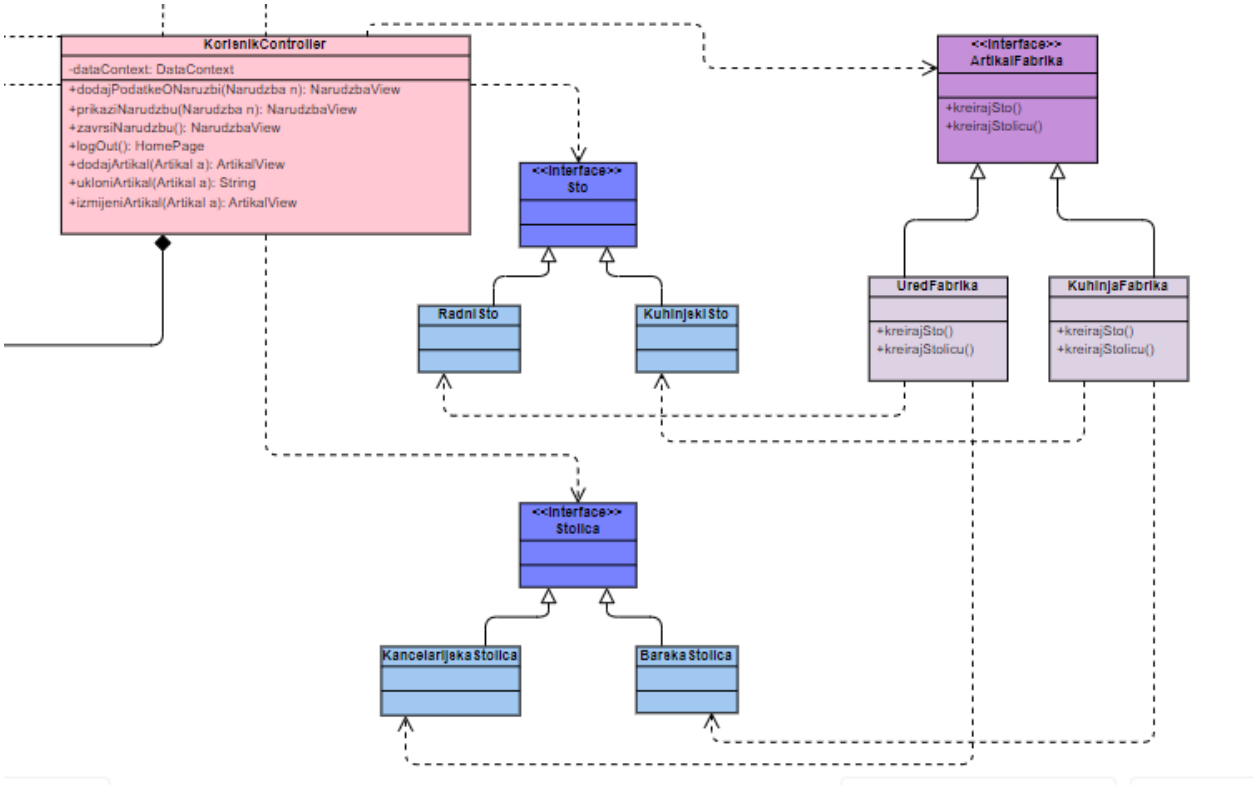
2. ***Prototype patern*** – dozvoljava da se kreiraju prilagođeni objekti bez poznavanja njihove klase ili detalja kako je objekat kreiran, koristi se ako je kreiranje novog objekta resursno skupo ili kompleksno. Ovaj patern umjesto da kreira objekat iz početka, klonira postojeće objekte i modifikuje ih na zahtjev. Definišemo PrototypeInterface, koji će imati metodu kloniraj(), dok će nam klasa Artikal služiti kao konkretan prototip. U sklopu KorisnikController imamo operaciju za modifikaciju artikla, koja se odvija na način da se klonira originalni prototip, zatim se izvrše odgovarajuće izmjene na kloniranom objektu, nakon čega se vrši spašavanje.



3. ***Factory method patern*** – omogućava kreiranje objekata na način da podklase odlučuju koju će klasu instancirati. Kada bismo imali klasz Zaposlenik, ovaj patern bismo mogli iskoristiti kada bismo razlikovali različite tipove zaposlenika, kao npr. menadžer, skladištar i sl. Ovisno o situaciji, tj. ovisno o tipu zaposlenika, mogli bismo imati ZaposlenikFactory, koja bi bila zadužena za kreiranje objekata tipa Zaposlenik, sa različitim kriterijima. Ovako osiguravamo umjesto da kontroleri ili klase direktno instanciraju objekte klase Zaposlenik, koristili bismo ZaposlenikFactory za dobijanje različitih tipova zaposlenika.

4. ***Builder patern*** – odvaja specifikaciju kompleksnih objekata od njihove stvarne konstrukcije, koristi se ako se objekti mogu podijeliti u skupove objekata koji se razlikuje samo po permutaciji njihovih dijelova. U našem slučaju, ovaj patern bismo mogli iskoristiti za kreiranje objekata klase Artikal, budući da je poprilično efikasan kada su u pitanju kompleksni objekti sa mnogo atributa i metoda, što je kod nas klasa Artikal. Također je pogodan kada u trenutku kretiranja objekta, svi atributi nisu poznati. Builder interface povezujemo sa svim kontrolerima koje i sama klasa Artikal koristi.

5. ***Abstract factory patern*** – odvaja definiciju klase proizvoda od klijenta, u smislu da se na osnovu apstrakne familije proizvoda kreiraju konkretne fabrike produkata različitih tipova i različitih kombinacija. Idealna primjena ovog paterna bi bila za kreiranje različitih tipova namještaja, odnosno artikala, koje su dostupne salonu. Kreiramo apstraktnu fabriku ArikalFabrika unutar koje definišemo metode za kreiranje različitih tipova artikala. To bi recimo bile metode kreirajSto, kreirajStolicu, kreirajKrevet i sl. Svaka konkretna fabrika (UredFabrika, KuhinjaFabrika...) bi implementirala ove metode i vraćala odgovarajuće objekte. Ova ideja je prikazana na sljedećoj slici.


KorisnikController
-dataContext: DataContext
+dodajPodatkeONaruzbi(Narudzba n): NarudzbaView
+prikaziNarudzbu(Narudzba n): NarudzbaView
+zavrsiNarudzbu(): NarudzbaView
+logOut(): HomePage
+dodajArtikal(Artikal a): ArtikalView
+ukloniArtikal(Artikal a): String
+izmijeniArtikal(Artikal a): ArtikalView
<<interface>>
Sto
<<interface>>
ArtikalFabrika
+kreirajSto()
+kreirajStolicu()
RadniSto
KuhinjskiSto
UredFabrika
+kreirajSto()
+kreirajStolicu()
KuhinjaFabrika
+kreirajSto()
+kreirajStolicu()
<<interface>>
Stolica
KancelarijskaStolica
BarskaStolica