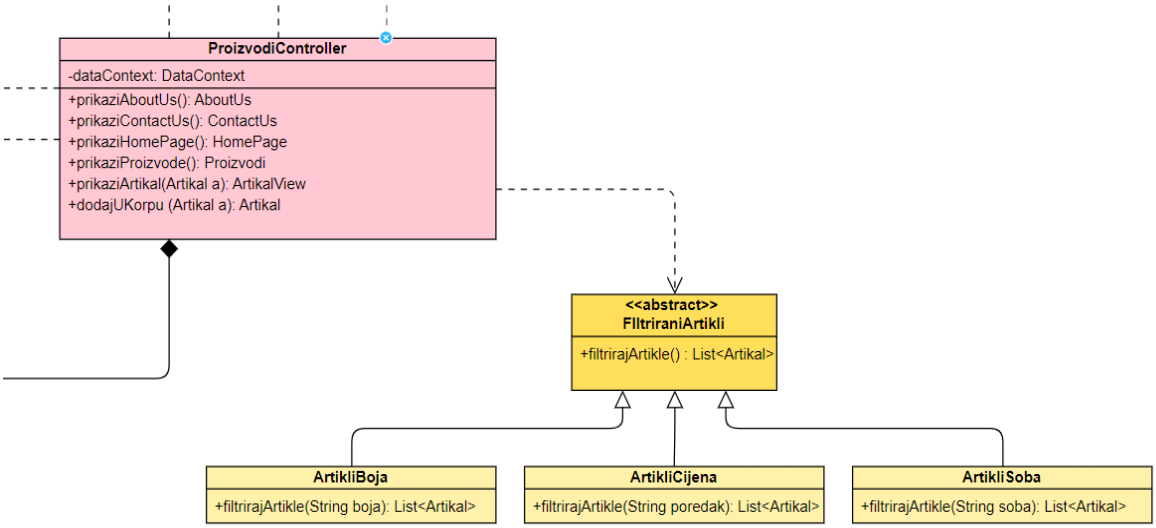
Paterni ponašanja

1. ***Strategy patern*** – koristi se kada postoje različiti primjenjivi algoritmi, odnosno strategije za neki problem. Ovaj patern bismo mogli iskoristiti kod odabira načina plaćanja narudžbe. Korisnik može odabrati kartično plaćanje ili plaćanje pouzećem. Kreiramo PlacanjeStrategy interfejs i definiramo metode *izvrsiPlacanje()* i *izvrsiPovrat()* za različite načine plaćanja. Imali bismo konkretne implementacije ovog interfejsa, i to *KarticaStrategy* i *GotovinaStrategy*, te bismo unutar klase Narudzba definisali metodu *postaviMedoduPlacanja()*, koja će postaviti željenu metodu plaćanja. Također bismo unutar klase *Narudzba* definisali i metodu *izvrsiPlacanje()*, koja bi na osnovu odabranog načina plaćanja pozvala odgovarajuću metodu.

2. ***State patern*** – objektu omogućava promjenu ponašanja na osnovu trenutnog stanja. U kontekstu našeg sistema za prodaju namještaja, state patern možemo primijeniti za upravljanje različitim stanjima narudžbe, npr. "naručeno", "u obradi", "isporučeno" i "otkazano". Ovo bismo postigli definiranjem *State* interfejsa unutar kojeg bismo definirali metode za svako moguće stanje narudžbe, nakon čega bismo imali konkretnu implementaciju interfejsa za svako stanje, npr. narudzbaNarucena, narudzbaUObradi, narudzbaIsporucena, narudzbaOtkazana. Interfejs *State* bi povezali sa kontrolerima koji rade sa narudžbama.

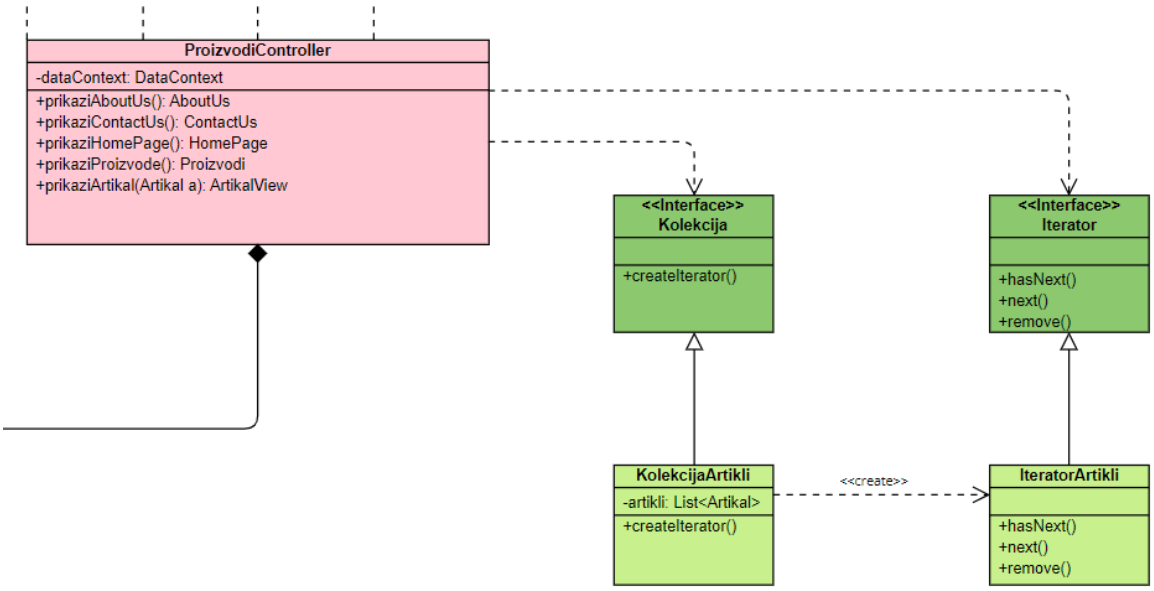
3. ***TemplateMethod patern*** - omogućava izdvajanje određenih koraka algoritma u odvojene podklase. U našem sistemu, ovaj patern bismo mogli primijeniti u svrhu prikaza artikala filtriranih na osnovu boje, cijene ili sobe u kojoj se primarno koristi. Definišemo apstraktnu klasu *FiltriraniArtikli* koja će sadržavati metodu *filtrirajArtikle()*. Biti će povezan sa *ProizvodiCotroller-om*, budući da je on zadužen za prikaz liste proizvoda. Konretno smo izdvojili dio u kojem se vršilo filtriranje iz funkcije prikaziProizvode. Zatim implementiramo konkretne klase koje proširuju ovu klasu i implementiraju metodu prema svojim potrebama. Ova ideja je prikazana na sljedećoj slici.



4. ***Chain of Responsibility patern*** - koristi se kada je potrebno da se neke akcije izvršavaju u lancu, u smislu da ako imamo tri radnje, izvršava se prva, pa druga, i naposlijetku treća. Dakle, nije moguće nakon izvršenja prve radnje preći na treću, već se mora izvršiti druga nakon prve. Ovaj patern bismo mogli primijeniti na proces dodavanja artikala u korpu i finaliziranja narudžbe. Korisnik prvo bira artikle koje dodaje u korpu, zatim potvrđuje artikle u korpi ili eventualno uređuje količinu tih artikala u korpi, nakon čega unosi podatke o dostavi i plaćanju. Ove radnje se izvršavaju u lancu, odnosno nije moguće unijeti podatke o dostavi i plaćanju ako prije toga uopšte nije bilo artikala u korpi.

5. ***Command patern*** – koristi se za enkapsuliranje svih informacija potrebnih za odgođeno izvođenje akcije ili pokretanje događaja. U našem sistemu, ovaj patern bismo mogli primijeniti na slučaj dodavanja artikala u korpu. Prvenstveno bismo implementirali interfejs *Command* koji sadrži samo jednu metodu *execute*, a zatim bismo definisali konkretnu implementaciju ovog interfejsa *DodajUKorpuCommand*. Ova klasa bi primala referencu na *Korpa* objekt, te informaciju o artiklu koji se dodaje. Metoda *execute* bi pozivala metodu *dodajUKorpu(artikal)* nad *Korpa* objektom. *Reciever* objekat (u našem slučaju *Korpa)* izvršava komandu za dodavanje namještaja u korpu.

6. ***Iterator patern*** - omogućava pristup elementima kolekcije bez poznavanja kako je kolekcija strukturirana. Iterator patern bismo mogli primijeniti za prolazak kroz listu namještaja. Definiramo interfejs *Iterator*, koji će sadržavati metodu za provjeru postojanja sljedećeg elementa, metodu za dohvaćanje sljedećeg elementa i metodu za uklanjanje elementa. Zatim definiramo interfejs *Kolekcija* koji predstavlja kolekciju namještaja, te ima metodu createIterator() koja će vraćati instancu Iteratora. Implementiramo konkretnu kolekciju namještaja ili narudžbi koja implementira interfejs *Kolekcija,* a nakon toga i konkretni iterator koji prolazi kroz elemente kolekcije. Interfejse povezujemo sa *ProizvodiController*-om, budući da je on "zadužen" za prikaz artikala.



7. ***Mediator patern*** - za komunikaciju među objektima bez direktnog povezivanja između njih. Umjesto toga, objekti komuniciraju putem posrednika ili posredničke komponente. Ovaj patern bismo mogli iskoristiti za komunikaciju između zaposlenih. Mediator objekt bi služio kao centralna tačka za razmjenu informacija između zaposlenih unutar sistema. Recimo, kada jedan zaposlenik završi za zahtjevom za naručivanje robe u inventaru, obavijest može poslati mediatoru, koji će tu informaciju dalje proslijediti ostalim uposlenim, kako ne bi došlo do naručivanja viška proizvoda.

8. ***Observer patern*** – uspostavlja relaciju između objekata tako kada jedan objekat promijeni stanje, drugi zainteresovani objekti se obavještavaju. Observer patern primijeniti kako bi pratili promjene u inventaru. Kreiramo klasu *Inventar* koja upravlja artiklima i posmatračima. Posmatrači su klase koje žele biti obaviještene o promjenama, kao što su korisnički interfejs, sistem za izvještaje i sistem za obavijesti. Kada se artikli dodaju ili uklone iz inventara, klasa Inventar obavještava sve registrovane posmatrače o promjenama. Posmatrači implementiraju interfejs posmatrača i definiraju konkretne akcije koje preduzimaju na obavijest. Recimo, korisnički interfejs može osvježiti prikaz, sistem za izvještaje može ažurirati podatke, a sistem za obavijesti može poslati notifikacije korisnicima.

9. ***Visitor patern*** - omogućuje dodavanje novih operacija ili funkcionalnosti objektima bez mijenjanja njihove strukture, tj. koristi se za oblikovanje i manipulaciju objektima u strukturama podataka, bez promjene samih objekata. Vezano za naš sistem, pogodno bi ga bilo primijeniti kada bismo imali opciju određenog popusta na artikle. To bi podrazumijevalo da definiramo *Visitor* interfejs koji bi sadržao metode za svaku vrstu namještaja koju želimo posjetiti (visitSto, visitStolica...), nakon čega bismo implementirali konkretne "visitore" koji implementiraju *Visitor* interfejs za primjenu popusta. Definiramo *Element* interfejs, koji predstavlja različite tipove najmeštaja, te konkretne tipove namještaja (Sto, Stolica, Sofa...) koji implementiraju ovaj interfejs, te sadrže metodu accept koja prima posjetitelja kao argument i poziva odgovarajuću metodu posjetitelja.

10. ***Interpreter patern*** – koristi se za interpretaciju definicija jezika, te omogućuje interpretaciju rečenica izraženih u tom jeziku. Ovaj patern bismo mogli iskoristiti za interpretaciju upita korisnika. Kada korisnik unese neki upit, npr. "Prikaži sve crvene stolice jeftinije od 200 KM", sistem bi koristio interpretatore za "crvene", "stolice" i "jeftinije od 200 KM" kako bi generirao odgovarajući upit prema bazi podataka i rezultate prikazao korisniku.

11. ***Memento patern*** - omogućuje snimanje trenutnog stanja objekta i vraćanje tog stanja u budućnosti, vez otkrivanja detalja njegove implementacije ili narušavanja njegove privatnosti. U slučaju našeg sistema, ovaj patern bismo mogli iskoristiti recimo za pamćenje stanja korpe prije nego što korisnik započne proces plaćanja. Na taj način, ako korisnik prekine proces plaćanja ili se vrati kasnije, može mu se omogućiti povratak u stanje u kojem je bio prije nego što je započeo proces plaćanja.