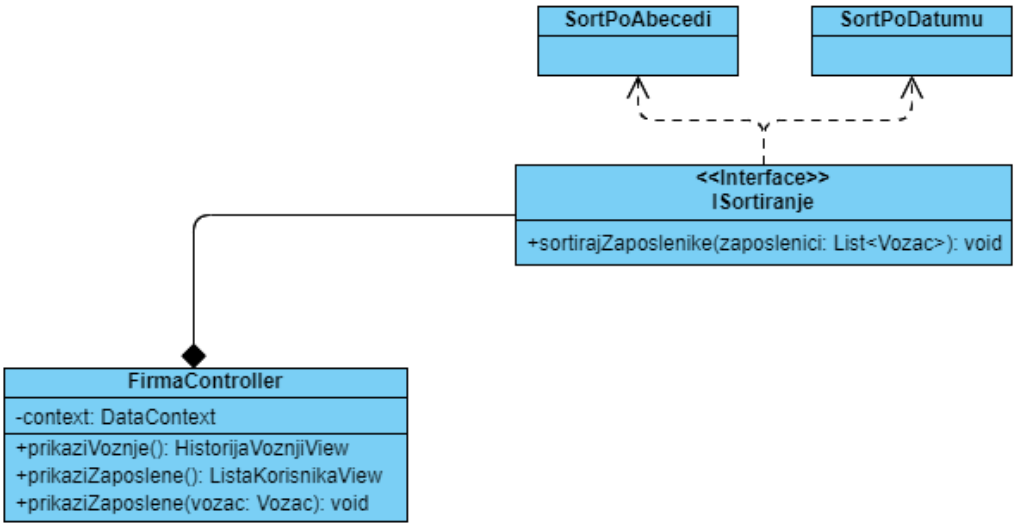
DIZAJN PATERNI PONAŠANJA

1. STRATEGY PATERN

Strategy patern izdvaja algoritam iz matične klase i uključuje ga u posebne klase.Pogodan je kada postoje različiti primjenjivi algoritmi (strategije) za neki problem.

Ovaj patern možemo iskoristiti prilikom sortiranja liste zaposlenika kojima mogu pristupiti vlasnici firma. Recimo možemo sortirati zaposleniko abecedeno po imenu, zatim možemo sortirati zaposlenike po datumu kada su počeli da rade u firmi.



2. STATE PATERN

State patern omogućava objektu da mijenja način ponašanja na osnovu trenutnog stanja. Postiže se promjenom podklase unutar hijerarhije klasa.

Ovaj patern bi mogli iskoristiti recimo da imamo dvije vrste korisnika VipKlijent i StandardKlijent. Recimo da nakon 50 vožnji kroz naš sistem omogućava se svakom klijentu da dobije jedinstveni kod koji nakon toga može ikoristiti da bilježi svaku sljedeću vožnju gdje bi se recimo nakon dodatnih 10 vožnji uračunao odredjeni popust. VipKlijentima omogućimo da dobiju ovaj kod čim postanu vip. U klasi Vožnja bi imali metodu koja bi davala korisnicima ovaj kod u zavisnosti od toga da li je klijent vip ili je standardni a broj vožnji kroz sistem mu je 50.

3. TEMPLATE METHOD PATERN

Omogućava izdvajanje određenih koraka algoritma u odvojene podklase. Ovaj patern definira kostur algoritma u superklasi, ali dozvoljava podklasama da nadjačaju određene korake algoritma bez promjene njegove strukture.

Ovaj patern bismo mogli iskoristiti na prilikom kreiranja korisničkog naloga u sistemu. Recimo prilikom prve prijave korisnika tj. registracije mogli bismo klijentu poslati mail dobrodošlice ili prikazati neku poruku dobrodošlice na ekranu.

4. OBSERVER PATERN

Uloga Observer paterna je da uspostavi relaciju između objekata tako kada jedan objekat promijeni stanje drugi zainteresirani objekti se obavještavaju.

Ovaj patern bismo mogli upotrijebiti za slanje obavijesti recimo od strane admina korisnicima. Recimo da admin kada izvrši neku promjenu ili ažuriranje sistema pošalje notifikaciju svim korisnicima da je to ažuriranje izvršeno ili npr. da admin pošalje obavijest korisnicima uz odredjeno upozorenje o načinu korištenja aplikacije. IObavijest bi bio handler preko kojeg bi admin mogao poslati te obavijesti.

5. ITERATOR PATERN

Iterator patern omogućava pristup elementima kolekcije bez poznavanja kako je kolekcija strukturirana.

Iterator patern bismo mogli implementirati kada bi uveli opciju prolaska kroz vožnje kolekcije na osnovu abecednog poretka ili recimo na osnovu cijene. Za ovo bi nam bile potrebne dvije klase AbecedaIterator, CijenaIterator koji bi implementirale metodu predjiNaSljedecuVoznju(): Voznja. Imali bi jednu klasu Kolekcija koja bi bila struktura preko koje bi pristupali vožnjama. Ona nasljeđuje interfejs IKreator koji sadrži metodu za kreiranje odgovarajućeg iteratora.

6. MEDIATOR PATERN

Mediator patern koji nam omogućava da smanjimo zavisnosti između objekata. Mediator ograničava komunikaciju između objekata i prisiljava ih da sarađuju samo preko posredničkog objekta.

Ovaj patern bi mogli iskoristiti pri obradi narudžbe. Mediator patern može poslužiti kao posrednik koji će koordinirati obradu narudžbe između različitih komponenti ili podsistema u našem sistemu. Recimo posrednik može obavijestiti sve vozače o novoj narudžbi, zatim kada vozač prihvati narudžbu, posrednik može obavijestiti firmu u kojoj je vozač zaposlen o statusu te narudžbe, zatim može obračunati plaćanje kada se vožnja završi.

7. CHAIN OF RESPONSIBILITY PATERN

Chain of Responsibility patern je patern koji nam omogućava da proslijedimo zahtjeve duž lanca handlera. Nakon što primi zahtjev, svaki handler odlučuje ili da obradi zahjtev ili da ga proslijedi sljedećem handleru u lancu.
Prilikom registracije kao vozač dolazi do provjere na više nivoa, prvo se vrši provjera podataka koje unosi vozač, kao što su ime, prezime, mail adresa itd, a zatim nakon toga se vrši do verifikacije profila gdje vozač mora skenirati vozačku dozvolu.