Paterni ponašanja

Strategy patern

Strategy patern vrši izdvajanje algoritma iz matične klase u posebne klase čime omogućava klijentu izbor algoritma iz familije algoritama, te održava neovisnost algoritama od klijenata koji ih koriste.

Ukoliko bismo htjeli omogućiti više vrsta sortiranja recepata, tako da se odabere najefikasniji način u zavisnosti od broja recepata koje je potrebno sortirati, mogli bismo iskoristiti ovaj patern. Tada bismo dodali novi interface ISortiranje sa metodom sortiraj(). Dodali bismo i klase QuickSort, MergeSort i InsertionSort, te bi ove klase implementirale pomenuti interface. Kreirali bi i klasu KolekcijaRecepata u koju bi smjestili listu recepata koju treba sortirati, te bi dodatni atribut te klase bio i tipa ISortiranje i određivao bi upravo koja strategija sortiranja će biti upotrijebljena prilikom sortiranja ove liste recepata, odnosnokoja verzija metode sortiraj() će biti pozvana.

State patern

State patern je dinamička verzija Strategy paterna i omogućava promjenu načina ponašanja objekta u zavisnosti od stanja u kom se nalazi. Za razliku od Strategy paterna, objekat sam mijenja svoje stanje, ono nije izabrano od strane klijenta.

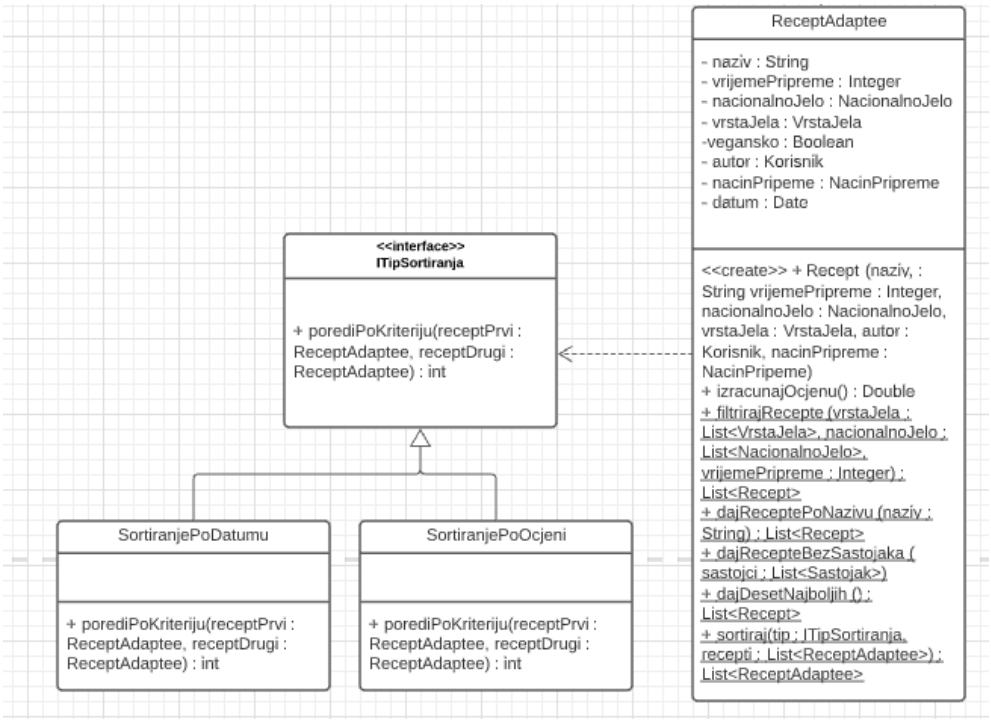
Recimo da želimo omogućiti korisnicima da žele svoj profil učiniti privatnim. Tada bi klasa Korisnik imala dva mode-a: privatni i javni, odnosno ovo bi bile klase koje implementiraju interface IMode. U zavisnosti od klase, odnosno mode-a bi omogućili drugim korisnicima prikaz profila tog korisnika, odnosno poziv metode prikaziProfil klase KorisnikController.

TemplateMethod patern

Ovaj patern omogućava izdvajanje pojedinih dijelova algoritama u podklase. Ovi izdvojeni dijelovi se mogu implementirati različito.

Obzirom da nam je potrebno sortiranje recepata po datumu objave, te po ocjeni, implementiraćemo ovaj patern. Naša algoritam klasa će u suštini biti klasa Recept jer ćemo unutar nje implementirati metodu sortiraj(ITipSortiranja tip, List<Recept> recepti) koja vrši sortiranje proslijeđene liste recepata, a interno poziva metodu interfejsa ITipSortiranja. ITipSortiranja je novi interface koji pruža metodu porediPoKriteriju(Recept r1, Recept r2) koja upravo predstavlja funkciju kriterija koja je potrebna za sistemsku metodu sort(). Nudiće se dva tipa sortiranja u vidu klasa SortiranjePoDatumu i SortiranjePoOcjeni koje će implementirati pomenuti interface, tako da prva klasa implementira metodu porediPoKriteriju(Recept r1, Recept r2) tako što poredi recepte po datumu objave, a druga po njihovoj ocjeni.

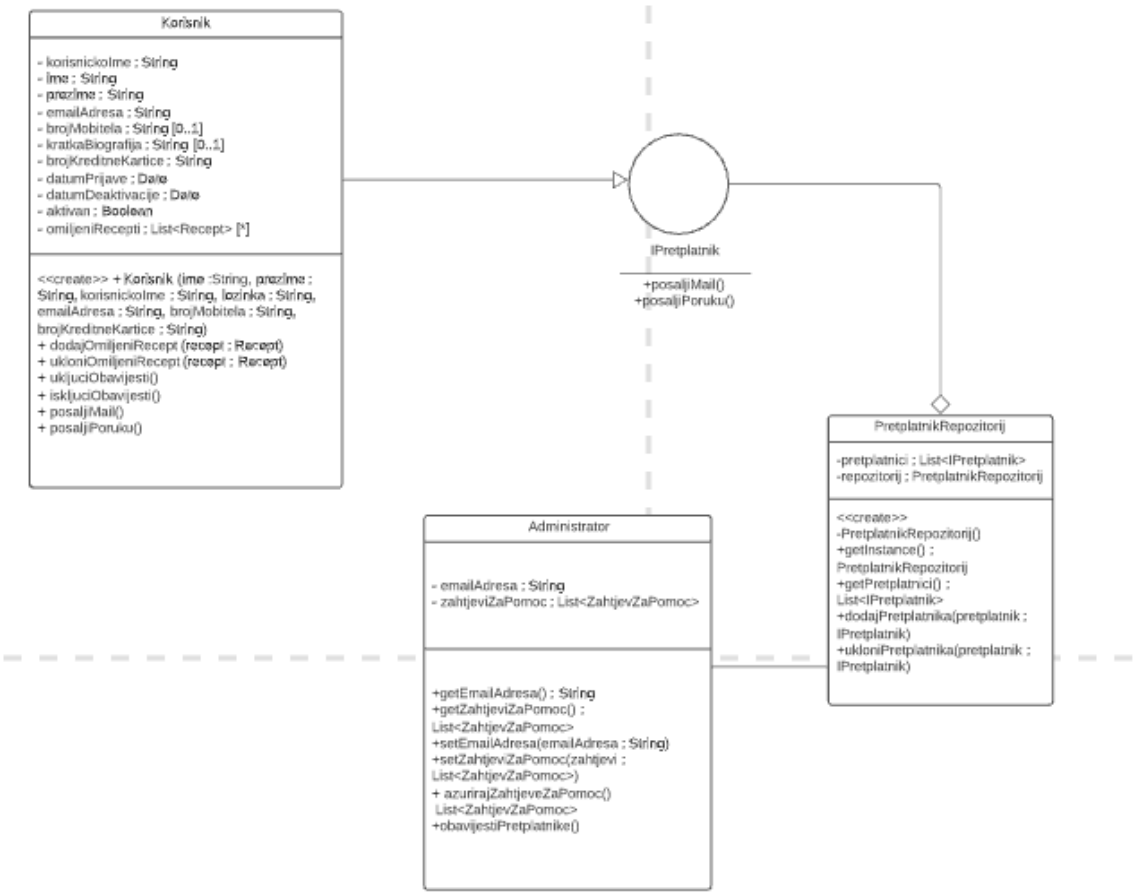
Napomena: Pod klasom Recept se misli na ReceptAdaptee, nakon primjene Adapter patterna.



Observer patern

Observer patern služi za uspostavljanje takve relacije između objekata, da promjene koje se dešavaju nad jednim objektom potiču slanje obavijesti drugim zainteresiranim objektima o tim promjenama.

U našoj aplikaciji zainteresirani korisnici će moći primati obavijesti o ažuriranju kategorije najboljih recepata putem e-maila i poruke na mobilni uređaj. Kako bi implementirali ovu funkcionalnost, poslužićemo se Observer paternom. Kreiraćemo interface IPretplatnik koji pruža metode posaljiMail() i posaljiSMS(), a ovaj interface implementiraće klasa Korisnik. Kada se korisnik pretplati na primanje obavijesti, on se dodaje u listu pretplatnika koja se čuva u Singleton klasi PretplatnikRepozitorij. Pristup ovoj klasi omogućen je klasi Administrator. U klasi Administrator imamo metodu obavijestiPretplatnike(), pa unutar te metode upravo možemo za svakog od pretplatnika koji se nalazi u listi u klasi PretplatnikRepozitorij pozvati metode posaljiMail() i posaljiSMS() da obavijestimo te korisnike o izvršenoj promjeni. U budućnosti možemo proširiti ovaj patern dodavanjem novih vrsta obavijesti, a korištenjem iste klase za pretplatnike.


Korisnik
- korisnickoIme : String
- ime : String
- prezime : String
- emailAdresa : String
- brojMobitela : String [0..1]
- kratkaBiografija : String [0..1]
- brojKreditneKartice : String
- datumPrijave : Date
- datumDeaktivacije : Date
- aktivan : Boolean
- omiljeniRecepti : List<Recept> [*]
<<create>> + Korisnik (ime :String, prezime : String, korisnickoIme : String, lozinka : String, emailAdresa : String, brojMobitela : String, brojKreditneKartice : String)
+ dodajOmiljeniRecept (recept : Recept)
+ ukloniOmiljeniRecept (recept : Recept)
+ ukljuciObavijesti()
+ iskljuciObavijesti()
+ posaljiMail()
+ posaljiPoruku()
IPretplatnik
+posaljiMail()
+posaljiPoruku()
PretplatnikRepozitorij
-pretplatnici : List<IPretplatnik>
-repozitorij : PretplatnikRepozitorij
<<create>>
-PretplatnikRepozitorij()
+getInstance() : PretplatnikRepozitorij
+getPretplatnici() : List<IPretplatnik>
+dodajPretplatnika(pretplatnik : IPretplatnik)
+ukloniPretplatnika(pretplatnik : IPretplatnik)
Administrator
- emailAdresa : String
- zahtjeviZaPomoc : List<ZahtjevZaPomoc>
+getEmailAdresa() : String
+getZahtjeviZaPomoc() : List<ZahtjevZaPomoc>
+setEmailAdresa(emailAdresa : String)
+setZahtjeviZaPomoc(zahtjevi : List<ZahtjevZaPomoc>)
+ azurirajZahtjeveZaPomoc() List<ZahtjevZaPomoc>
+obavijestiPretplatnike()

Iterator patern

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

Recimo da želimo da administrator prolazi kroz listu zahtjeva za pomoć tako da mu zahtjevi s većim prioritetom, odnosno oni na koje što prije treba da odgovori, dolaze prvi. Ta klasifikacija po prioritetu se može vršiti upravo na osnovu kategorije zahtjeva za pomoć (npr. da Neprimjereni sadržaj ima najveći prioritet (1), a Pitanja i sugestije najmanji (4)). Tada bismo dodali u klasu Administrator IIterator, odnosno interface koji implementira metodu dajNaredniZahtjevZaPomoc(). Ovaj interface bi implementirale klase IteriranjePoKriteriju i ObicniIterator, koje interno sadrže i listu zahtjeva za pomoć. U IteriranjePoKriteriju klasi bi metoda dajNaredniZahtjevZaPomoc() bila implementirana tako da vraća zahtjev za pomoć iz liste s najvišim prioritetom, dok bi u klasi ObicniIterator ona prosto vraćala sljedeći objekat iz te liste. Također bi i dodali IKreatorIteratora interface koji pruža metodu kreirajIterator(int vrsta, List<ZahtjevZaPomoc> zahtjevi) za kreiranje potrebnog iteratora, a koji implementira klasa Administrator.

Medijator patern

Medijator patern bi se mogao koristiti ako u našu aplikaciju dodamo opciju "chat" (između korisnika i administratora). Između ovih objekata postojao bi medijator koji bi kontrolisao razmjenu poruka. Na ovaj način bismo imali kontrolu na jednom mjestu.