Paterni ponašanja

Bave se organizacijom algoritama, dodjeljivanja odgovornosti i komunikaciju između objekata.

Strategy patern

Izdvaja algoritam iz matične klase i uključuje ga u posebne klase.

Ovaj patern bismo u našem sistemu mogli iskoristiti ukoliko bismo imali mogućnost izbora različitog prevoznog sredstva. Tada bismo dodali klasu PrevoznoSredstvo preko koje bi odlučivali koju 'strategiju' ćemo koristiti, tj na koje prevozno sredstvo je odabrano. Implementirali bismo interfejs IprevoznoSredstvo koji bi imao metodu izračunajCijenu i njega bi naslijedile klase Autobus i Avion koje bi na različit načine računale cijenu aranžmana, tj. na osnovnu cijenu bi dodavale različite iznose. Bilo bi moguće promijeniti 'strategiju', tj. prevozno sredstvo pri čemu bi se cijenja mijenjala shodno algoritmu implementiranom u određenoj klasi. Klasa Rezervacija bi sadržala instancu klase PrevoznoSredstvo, jer bi vrsta prevoznog sredstva ovisila od toga šta bira klijent.

Na ovaj način bismo smanjili kompleksnost metoda time što ne bismo koristi mnogo if ili switch-case izraza.

State patern

Mijenja način ponašanja objekata na osnovu trenutnog stanja

U našem primjeru ovaj patern bismo mogli implementirati tako što klasa RezervacijaController šalje obavijesti klasi Context do kojeg stepena rezervacije putovanja je korisnik stigao. Početno stanje bi bilo 'nemaRezervacije'. Na akciju klika na dugme Rezervisi, klasa Context mijenja svoje stanje na 'rezervisano', dok bi klasa Rezervacija izvršila rezervaciju putovanja za tog korisnika. Ukoliko bi klijent izabrao elektronski način plačanja, tada bi sljedeće akcija bila uplata amortizacije, pa bi se stanje promijenilo na 'uplacenaAmortizacija', dok bi klasa Amortizacija vršila isplatu vrijednosti amortizacije sa računa klijenta na račun agencije. Posljednje stanje bi bilo 'uplacenCijeliIznos', a da bi se moglo doći do tog stanja, bilo bi potrebno uvesti opciju 'Plati cijeli iznos' u naš sistem, i tada bi klasa CijeliIznos vršila isplatu ostatka cijene putovanja sa računa klijenta na račun agencije.

TemplateMethod patern

Omogućava algoritmima da izdvoje pojedine korake u podklase.

Ovaj patern nismo primijenili, ali bismo mogli ukoliko dodamo da klijent može biti obični i VIP i ukoliko bismo imali bolje razrađenu akciju plaćanja. U apstraktnoj klasi Klijent bismo

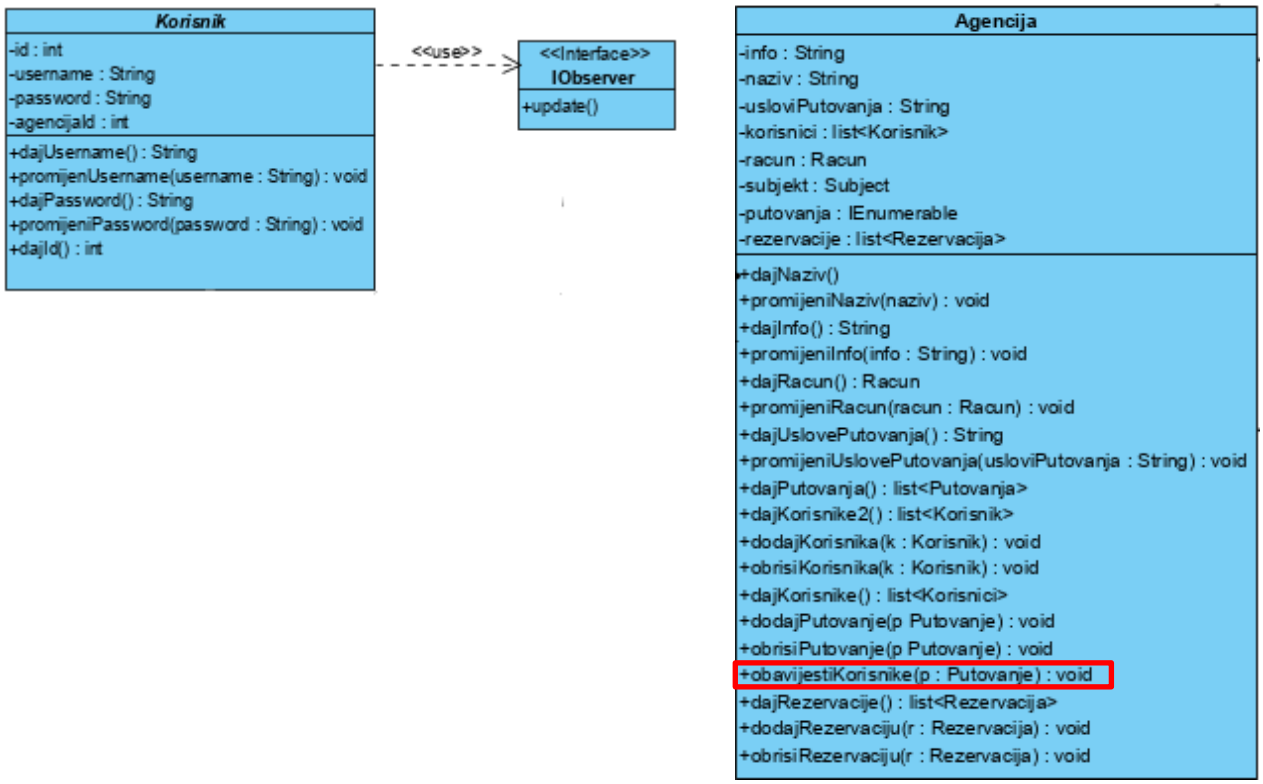
implementirale template metodu placanje(), koja poziva metode platiAkontaciju(), obracunajPopust() i platiPunuCijenu(). Ovu metodu izvedene klase ObičniKlijent i VIPKlijent neće mijenjati, ali ostale metode koje metoda placanje() poziva mogu biti različito implementirane u ovisnosti od klase. Metoda platiAkontaciju() za kalsu ObičniKlijent bi obračunavala akontaciju kao 40% pune cijene, dok bi za klasu VIPKlijent akontaciju obračunavala kao 20% pune cijene. Nad metodom izracunajPopust() nije potrebno izvršiti override, već je dovoljno nju implementirati u apstraktnoj klasi da računa popust od 10% na punu cijenu, i da se u apstraktnu klasu uvede atribut čija će vrijednost biti npr. *true* za VIP, a *false* za obične klijente, pa će se na osnovu tog atributa u metodi placanje() odlučivati da li će se pozivati metoda za računanje popusta ili ne. Metodu platiPunuCijenu() također nije potrebno mijenjati u izvedenim klasama, ali nju metoda placanje() poziva umjesto prethodne dvije metode ukoliko neki atribut inicira da je klijent odabrao odmah platiti punu cijenu.

Observer patern

Uspostavlja relaciju između objekata takvu da kad se stanje jednog objekta promijeni svi vezani objekti dobiju informaciju

Svim putnicima koji su rezervisali putovanje šalje se obavijest putem mail-a ukoliko se putovanje odgodi.

Metoda obrisiPutovanje klase Agencija poziva metodu obavijestiKorisnike preko koje se svim klijentima iz liste onih koji su rezervisali to putovanje šalje obavijest da je putovanje otkazano. Ta metoda prolazi kroz list rezervaciju i za svakog korisniku koji je rezervisao to putovanje, a koje se šalje kao parametar, poziva metodu update() koja vrši slanje obavijesti klijentu putem mail-a. Update metodu smo implementirali u inerfejsu IObserver kojeg implementira klasa Korisnik.



Iterator patern

Omogućava pristup elementima kolekcije sekvencijalno bez poznavanja interne strukture kolekcije

U našem primjeru u klasi Agencija imamo listu svih putovanja. Cilja nam je da omogućimo iteriranje foreach petljom kroz ovu lista na način da nam prvi element bude ono putovanje čiji je prihod najmanji, a svako iduće putovanje ono čiji je prihod najmanji od onih koji su veći od posljednjeg. U klasu Agencija smo dodali instancu klase Iterator koja će nam to omogućiti. Klasa Agencija također implementira interfejs IterableCollection koji ima metodu createIterator. Ova metoda poziva konstruktor klase Iterator kojoj šalje listu prihoda i broj indeksa od kojeg započinjemo iteraciju. Prihod će se računati preko liste putovanja i rezervacija, pri čemu će se tražiti one rezervacije koje sadrže određeno putovanje i sabirati će se cijene tih rezervacija. Ovo ćemo uraditi jer se cijena aranžmana može razlikovati ukoliko klijent iskoristi neki od kodova za popuste, što će biti uračunato u cijenu klase Rezervacija. Ukoliko se promijeni lista putovanja, tada se ponovo poziva metoda createIterator sa ažuriranom listom prihoda i željenom pozicijom indeksa.