Paterni ponašanja

Strategy pattern

Strategy patern izdvaja algoritam iz matične klase i uključuje ga u posebne klase. Pogodan je ukoliko postoje različiti primjenjivi algoritmi za neki problem. Potencijalna mjesta za primjenu ovog paterna vidimo pri sortiranju te pretraživanju podataka o pacijentima. Uposlenici klinike bi mogli odabrati neki algoritam za sortiranje medicinskih kartona preko klase Nalaz. Ukoliko razmatramo primjenu ovog paterna pri sortiranju podataka o pacijentima u sistemu,možemo to realizirati na sljedeći način. Možemo omogućiti dva načina sortiranja, po imenu i po prezimenu. Da bismo implementirali ovaj patern, potrebno je da imamo interfejs ISortiraj u kojem imamo metodu sortiraj() te konkretne implementacije ovog interfejsa, što su ovom slučaju klase SortiranjePoImenu i SortiranjePoPrezimenu. U ovim klasama se nalazi implementacije metode sortiraj().

Pozivanje ovih metoda može biti aktivirano pritiskom na dugme ili neku opciju na grafičkom okruženju aplikacije.

Observer patern

Glavna uloga ovog paterna je da uspostavi relaciju između objekata tako da kada jedan objekat promijeni stanje, drugi zainteresirani objekti se obavještavaju.

U našem projektu, ovaj patern smo iskoristiti kada korisnik odabere slobodan termin za pregled te taj zahtjev biva odobren korisnik dobije obavještenje. Također ukoliko se desi neka promjena korisnik biva obaviješten.

Unutar klase Termin nalazila bi se lista registrovanih korisnika kojima se treba poslati obavještenje. Unutar Temin klase bi se trebale dodati metode registrujNovogClanaNaListiZaObavjestenja(Pacijent) te metoda PosaljiObavijest(Pacijent) koja će slati obavještenja pacijentima o eventualnim promjenama, podsjetnicima, potvrdama o uspješno zakazanom terminu itd.

State pattern

State patern koristimo kada imamo objekt koji se ponaša različito ovisno o trenutnom stanju u kojem se taj objekat nalazi.Ovaj pater možemo realizirati tako što ćemo napraviti klase koje ce tačno opisivati trenutno stanje objekta.

Ovaj patern bi mogli iskoristiti kada bi na osnovu trenutnog broja zakazanih (i odobrenih) termina od strane korisnika formirali određena stanja. Svaki korisnik sa jednim aktivnim terminom ne ostvaruje nikakav popust/privilegije i nalazi se na stanju Pacijent1, dok za svaka dva termina, pacijent prelazi u stanje Pacijent2, Pacijent3... Ove izmjene bi se izvršavale automatski u zavisnosti od broja trenutno zakazanih termina.

Iterator pattern

Iterator patern omogućava sekvencijalni pristup elementima kolekcije bez poznavanja kako je kolekcija struktuirana. Struktura iterator paterna se sastoji iz nekoliko dijelova. Prvi je kolekcija objekata kroz koju se želi iterirati.
Ovaj patern možemo iskoristiti za prolazak kroz listu doktora klinike unutar klase UposlenikUsluga. Da bi se implementirao ovaj patern, potrebno je kreirati klasu Iterator koja će implementirati interfejs iIterator. Pomenuta klasa bi imala atribute listu tipa UposlenikUslugai indeks trenutnog elementa i metode dajSljedećeg() i dajTrenutnog().

TemplateMethod patern

Ovaj patern omogućava izdvajanje određenih koraka algoritma u odvojene klase. Sama struktura algoritma se ne mijenja, nego se mali dijelovi operacija izdvajaju i oni se mogu različito implementirati.
Ovaj patern bismo mogli iskoristiti u našem sistemu npr. za pretraživanje pacijenata i usluga. To bismo uradili na način da imamo klasu Pretrazivanje i u toj klasi metodu pretrazi(IPretrazivanje obj, int nacinPretazivanja). Zatim bismo dodali interfejs IPretrazivanje koji bi sadržavao definicije metoda pretražiPoImenu(), pretraziPoPrezimenu, pretraziPoNazivuUsluge(), itd. Pošto nam se lista korisnika i termina nalazi u klasama Osoba i Termin, mogli bismo staviti da klase Osoba i Termin implementira interfejs IPretrazivanje, i u njoj dodati implementacije metoda koje su navedene u interfejsu.