STRUKTURALNI PATERNI

Adapter pattern

Adapter pattern koristimo kada je potreban drugačiji interfejs već postojeće klase, a ne želimo mijenjati postojeću klasu. Tim postupkom se dobija željena funkcionalnost bez izmjena na originalnoj klasi.

Adapter pattern u našem sistemu

Kao primjer primjene Adapter patterna u našem sistemu uradili smo sljedeću situaciju. U našem sistemu u klasi Doktor postoji metoda prikaziSimptome(), međutim dolazi do potrebe da doktor, radi svoje evidencije i znanja, želi imati evidenciju u najčešće simptome kod osoba zaraženih sa Covid-19. Da ne bismo mijenjali čitavu postojeću klasu doktor, dolazi do primjene adapter patterna. Uvesti ćemo novu metodu prikaziNajcesceSimptome().

Realizaciju adapter patterna možemo podijeliti u nekoliko koraka:

- Definiranje interfejsa ISimptomi koji sadrži metodu prikaziNajcesceSimptome()
- Adaptee klasa je klasa Doktor
- Definišemo klasu DoktorAdapter koja implementira interfejs ISimptomi. U metodi prikaziNajcesceSimptome() prolazimo kroz sve pacijente i pozivamo metodu prikaziSimptome, tri simptoma koja se najviše puta ponavljaju među pacijentima su najčešći simptomi.

Na ovaj način zapravo smo uspjeli unaprijediti klasu Doktor bez da vršimo direktne izmjene nad njom.

Decorator pattern

Osnovna namjena Decorator paterna je da omogući dinamičko dodavanje novih elemenata i ponašanja (funkcionalnosti) postojećim objektima. Objekat pri tome ne zna da je urađena dekoracija što je veoma korisno za ponovnu upotrebu komponenti softverskog sistema. Da ne bismo imali veliki broj izvedenih klasa dovoljno je da primjenimo decorator pattern i time omogućimo dekorisanje određenih objekata.

Decorator pattern u našem sistemu

Neka je zahtjev idući: „Doktor vrši izdavanje uputnica. Moguće je izdavanje dvije vrste uputnica, uputnica za pulmologa i uputnica za vađenje krvi.“

Prvo ćemo kreirati interfejs IUputnica koji sadrži metodu izdaj(). Sama klasa Uputnica treba da implementira interfejs IUputnica. Što se tiče konkretnih dekoratera, svaki od njih treba sadržavati kao atribut Uputnica. Konkretni dekorateri u našem slučaju su izdajUputnicuZaPulmogloga() i izadjUputnicuZaKrv().

Facade pattern

Osnovna namjena Facade patterna je da osigura više pogleda visokog nivoa na podsisteme (implementacija podsistema skrivena od korisnika). Operacije koje su potrebne određenoj korisničkoj perspektivi mogu biti sastavljene od različitih dijelova podsistema.

Facade pattern u našem sistemu

U našem sistemu postoje klase Vakcina i Vakcinacija. Primjena Facade patterna mogla bi se ogledati u pravljenju nove klase koju bismo nazvali npr. Vakcinisanje. Takva nova klasa sadržavala bi sve što sadržavaju klase Vakcina i Vakcinacija, čime bi se smanjila komunikacija između podsistema i sve bi se nalazilo u jednoj klasi. Također sve ostale klase koje su spojene sa klasama Vakcina i Vakcinacija bile bi spojene sa novom klasom Vakcinisanje.

Bridge pattern

Osnovna namjena Bridge patterna je da omogući odvajanje apstrakcije i implementacije neke klase tako da ta klasa može posjedovati više različitih apstrakcija i više različitih implementacija za pojedine apstrakcije.

Bridge pattern u našem sistemu

Ovdje uvodimo novi korisnički zahtjev. Korisnik želi da ima uvid koliko je novca do sada potrošeno na nabavku koje vakcina. Više je vrsta vakcina i njihove cijene su različite i ovise o količini mililitara u dozi te ovise o tome da li se cijena npr. 15$ odnosi na jednu dozu vakcine ili dvije doze(ukoliko vakcina zahtjeva revakcinaciju).

Za ovakav zahtjev nam je potreban Bridge u kojem se nalazi metoda dajPotroseniNovac() i atribut ITrosak koji sadrzi metodu izracunajTrosak(), takav trošak biti će računat po broju doza, broju mililitara u jednoj dozi vakcine i količini koja je došla, te po tome da li dolaze dvije vakcine u paketu ili jedna. Sve iznad nabrojano nalazilo bi se kao atributi u svakoj vrsti vakcine pojedinačno.

Proxy pattern

Namjena Proxy patterna je da omogući pristup i kontrolu pristupa objektima ili metodama. Proxy je obično mali javni surogat objekat koji predstavlja kompleksni objekat čija aktivizacija se postiže na osnovu postavljenih pravila.

Proxy pattern se koristi da bi postigli dodatnu sigurnost u sistemu.

Proxy pattern u našem sistemu

Primjena proxy patterna u našem sistemu se javlja pojavom prijave za termin revakcinacije odnosno terminDrugeDoze. Kako vakcina Sinopharm nema proces revakcinacije, prijavu za termin

revakcinacije je potrebno dodatno osigurati i omogućiti to samo ljudima koji su primili ostale vrste vakcina.

Da bi postigli to dodan je interfejs IProxyOcjenjivanje i klasa ProxyOcjenjivanje koja kroz metodu provjeriVakcinu(korisnik: Korisnik) vrši detekciju koju vakcinu je određeni korisnik primio, te ukoliko je riječ o vakcini koja nije Sinopharm, korisnik će moći pristupiti biranju termina za revakcinaciju. Na ovaj način bi se unaprijedila sigurnost sistema te svakako ne bi moglo doći do zabune da li osobe koje su primile Sinopharm vakcinu trebaju ići na revakcinaciju.

Composite pattern

Osnovna namjena Composite patterna (kompozitni patern) je da omogući formiranje strukture stabla pomoću klasa, u kojoj se individualni objekti (listovi stabla) i kompozicije individualnih objekata (korijeni stabla) jednako tretiraju

Omogućava uređenu hijerarhijsku strukturu koja zadržava uniformnost tako što je moguće istu metodu primjeniti nad različitim implementacijama.

Composite pattern u našem sistemu

U našem sistemu bismo mogli upotrijebiti composite pattern nad klasom Korisnik. Korisnike možemo podijeliti na korisnike sa kiseoničkom podrškom i korisnike na respiratoru, pri čemu bi se stvorilo stablo gdje bi klasa Korisnik bila korijen tog stabla. Klase kisikKorisnik i respiratorKorisnik bi imale metode koje vraćaju broj pacijenata na kisiku i broj pacijenata na respiratoru, dok bi u klasi Korisnik dodali metodu getBrojHospitaliziranih() pri čemu bi se nad svakim pacijentom provjeravalo da li je jedan od hospitalizovanih i ako jeste jedan od dva pomenuta tipa brojHospitalizovanih bi se povećavao za jedan.

Flyweight pattern

Osnovna namjena Flyweight patterna je upravo da se omogući da više različitih objekata dijele isto glavno stanje, a imaju različito sporedno stanje.

Flyweight pattern u našem sistemu

U našem sistemu postoji klasa Vakcinacija koja sadrži atribute terminPrveDoze i terminDrugeDoze. Na početku te vrijednosti su default-ne. Npr. terminPrveDoze dok se ne odabere ručno je današnji datum, a termin druge doze je 6 sedmica poslije. Dolazi do promjena terminDrugeDoze u zavisnosti od vrste vakcine zbog toga što ne zahtjevaju sve vakcine da prođe isto vremena od primanja prve doze.