Univerzitet u Sarajevu
Elektrotehnički fakultet u Sarajevu
Odsjek za računarstvo i informatiku

Strukturalni paterni
Imunizacija '21

Objektno orijentisana analiza i dizajn

Naziv grupe: Schpritzer
Članovi: Muhamed Borovac
Eldar Čivgin
Dženan Nuhić
Benjamin Pašić

1. Adapter patern

Svrha **Adapter patern**-a je da omogući širu upotrebu već postojećih klasa. U situacijama kada je potreban drugačiji interfejs već postojeće klase, a ne želimo mijenjati postojeću klasu koristi se Adapter patern. Adapter patern kreira novu adapter klasu koja služi kao posrednik između originalne klase i željenog interfejsa.
Adapter patern možemo koristiti u zastarjelim sistemima koji funkcionišu dobro, čije metode možemo idalje koristiti, ali želimo nadograditi sistem nekim novim funckionalnostima.
U našem sistemu, u budućnosti, ako dođe do promjena u imunizaciji (npr. dođe do promjene u vakcinaciji u smislu da se prima više doza, ili da je potrebno primati vakcinu na godišnjem nivou), mi možemo kreirati klasu *VakcinacijaAdapte*r kao i interfejs *IPrimioDozu*.
VakcinacijaAdapter bi u sebi sadržavala potrebne dodatne mogućnosti da bi se bez promjene klase Vakcinacija mogao funkcionalno koristiti naš sistem uz pomenute izmjene.

2. Facade patern (primijenjen)

Svrha **Facade patern**-a je što pruža jedinstven interfejs na skup interfejsa u podsistemu. Facade definira interfejs na višem nivou koji olakšava upotrebu podsistema.
Motivacija Facade paterna je upravo to što organiziranje sistema u podsisteme pomaže u smanjenju kompleksnosti. Uobičajeni cilj dizajna je da se minimizira komunikacija i ovisnosti između podsistema.
U našem sistemu postoje klase *ZahtjevZaTestiranje* i *ZahtjevZaVakcinaciju* sa određenim metodama. Mi smo taj sistem proširili time što smo ubacili klasu *ZahtjevFacade* koja objedinjuje sve metode koje se nalaze u dvije navedene klase.
Kako su *ZahtjevZaTestiranje* i *ZahtjevZaVakcinaciju* povezane i sa klasom *StrucnaOsoba* i *Korisnik*, mi smo našu klasu *ZahjtevFacade* povezali sa njihovom apstraktnom klasom *Osoba* iz koje su one naslijeđenje.
Time *Osoba*, bilo *StrucnaOsoba* ili *Korisnik*, može pristupati objema klasama *ZahtjevZaTestiranje* i *ZahtjevZaVakcinaciju* preko klase *ZahtjevFacade* čime je ovaj proces olakšan.

3. Decorator patern

Svrha **Decorator patern**-a je 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.
U našem sistemu postoji klasa *CovidKarton* koja služi za prikaz nekih osnovnih informacija o *Korisnik*u, specifično za stvari vezane za trenutnu pandemiju (*CovidTest* i *Vakcinacija*). U budućnosti se može pojaviti potreba za širenjem ove klase te dodavanjem novih atributa i metoda. Kako ne bi mijenjali *CovidKarton* klasu i time naštetili funkcionisanju cijelog sistema, mi ovdje možemo primjeniti Decorator patern, tako što ćemo dodati interfejs *ICovidKarton* i klasu *Decorator* koja bi sadržavala dodatne metode i atribute koje želimo implementirati.

4. Bridge patern

Svrha **Bridge patern**-a je omogućavanje da se iste operacije primjenjuju nad različitim podklasama. Ovim izbjegavamo nepotrebno dupliciranje koda, odnosno izbjegavamo kreiranje novih metoda za već postojeće funkcionalnosti.
Bridge patern nam omogućava da razdvojimo velike klase u dvije razdvojene hijerarhije - abstrakciju i implementaciju, koje mogu biti razvijane neovisno jedna od druge.
Ovaj patern je pogodan kada se implementira nova verzija softvera a postojeća mora ostati u funkciji.
U našem sistemu ovaj patern bismo mogli primijeniti ukoliko odlučimo da omogućimo plaćanje testiranja i vakcinisanja putem naše aplikacije. Kako bi se smanjila potreba za fizičkim kontaktom, svi korisnici bi dobijali malu količinu popusta ako se odluče za online plaćanje. Ako bismo smo u klasama *CovidTest* i *Vakcina* dodali metode *getCijena()*, onda bi ove klase mogle implementirati interfejs *Bridge* koji bi u sebi istoimenu metodu *getCijena()*, koji bi na osnovu vrste testa/vakcine računao cijenu.

5. Composite patern

Osnovna namjena **Composite patern**-a je da omogući formiranje strukture stabla pomoću klasa, u kojoj se individualni objekti (listovi stabla) i kompozicije individualnih objekata (korijeni stabla) jednako tretiraju. Ovo zapravo znači da je moguće pozivati metodu koja je zajednička na nivou svih tih klasa.
U sklopu našeg sistema najbolja primjena ovog patern-a bi bila sa klasom *Vakcina*. Za primjenu ovog patern-a bismo ovdje mogli proširiti ovu klasu tako što bismo dodali više vrsta firmi kao klase, koje su proizvođači određene vrste vakcina. Primjera radi, uzet ćemo samo dvije klase proizvođača, i to *Moderna*

i *Sinovac*. Na ovaj način bi se stvorila zahtijevana struktura stabla, gdje bi korijen stabla bila klasa *Vakcina*, a listovi stabla klase *Moderna* i *Sinovac*.
Za finalizaciju ovog patern-a potreban nam je neki interfejs. U našem slučaju implementirali bismo interfejs *IOpisVakcine* koji sadrži metodu *dajOpis()*. Ova metoda bi vratila detaljan opis za vakcinu određene kompanije, čiji bi nam opis mogao eventualno olakšati odabir željene vakcine. Ova metoda će se naravno moći pozivati iz klase *Vakcina* preko neke novo dodane metode *dajOpis()* koja bi vraćala opis vakcine koju proizvodi svaka od tih kompanija, pri čemu se svaki od tih opisa razlikuju.

6. Proxy patern (primijenjen)

Svrha **Proxy patern**-a je da omogući pristup i kontrolu pristupa stvarnim objektima. Proxy je obično mali javni surogat objekat koji predstavlja kompleksni objekat čija aktivizacija se postiže na osnovu postavljenih pravila.
U našem sistemu korisnik može imati zakazanu drugu dozu bez da je primio prvu dozu. Kako bi osigurali da ne dođe do toga, mi možemo implementirati Proxy patern.
Proxy patern možemo primjeniti u klasi *Vakcinacija* tako što ćemo dodati interfejs *IDrugaDozaProxy* kao i klasu *DrugaDozaProxy* sa metodom *odobrenaDrugaDoza()* koja će osigurati da Korisnik ne može primiti drugu dozu vakcine bez da je već primio prvu dozu.

7. Flyweight patern

Flyweight patern se može primjeniti da bi se napravio objekat koji bi minimizirao utrošak memorije tako što on dijeli što više podataka sa njemu sličnim objektima. Njegova glavna funkcija je da omogući da više zasebnih objekata dijele isto glavno stanje, a da im sporedna stanja budu različita.
U našem sistemu svaki korisnik ima Covid karton, te kada se korisničkom računu pristupi prvi put, Covid karton ima neko default-no stanje, bez testova i vakcinacija. Flyweight patern bi mogli iskoristiti tako što ujedinimo svaki default-ni Covid Karton u jedan resurs, te od njega odvojimo personalizovane Covid Kartone.
To možemo uraditi tako što ćemo implementirati interface *ICovidKarton* sa metodom *dajCovidKarton()*. Taj interfejs će naslijediti klase *CovidKarton* i *CovidKartonDefault*, koja će u sebi sadržavati osnovni oblik Covid kartona.