Kreacijski paterni

Singleton patern

U našoj aplikaciji je već primijenjen singleton patern u vidu klase Termini, ali ovaj patern se može realizovati i u klasama Adresar i Cjenovnik. Obje klase će biti korištene globalno i neće biti potrebe za njihovim instanciranjem u većem broju. Pošto će ove klase koristiti samo korisnici aplikacije (klasa Cjenovnik će biti ista i za Doktora i za Pacijenta, dok će klasu Adresar koristiti samo Doktor za mogućnost pregleda i editovanja), idealni su za implementaciju singleton paterna.

Prototype patern

Ovaj patern je fokusiran na uštedi resursa pri obavljanju rada aplikacije pa shodno tome se može primijeniti za klasu Termin. U slučaju da jedan pacijent ima više termina koji će se trebati dodati u Raspored ili Planner, to bi zahtjevalo kreiranje zasebne instance za svaki od potrebnih termina. To možemo zaobići kloniranjem (shallow) samo jednog termina instanciranog za određenog pacijenta u više navrata i time uštedjeti resurse koji bi se potrošili u suprotnom. Također ovaj patern bi mogao poslužiti za sortiranje protokola u klasi Knjiga protokola. Pristup atributima svake stomatološke usluge mora biti omogućen da bi po određenom kriteriju vršili sortiranje. Shodno tome će se morati kreirati, pri vraćanju iz baze, nove instance Stomatološke usluge. Pozivanje sortiranja će se vršiti na raznim mjestima, pa nam kloniranje samo jedne instance može uštedjeti veliku količinu resursa (broj stomatoloških usluga za jednu ordinaciju s vremenom raste).

Factory Method patern

Naša aplikacija je otvorena za nadogradnju pa samim tim možemo pretpostaviti da će se u budućnosti dodavati korisnici u skladu sa real-world dešavanjima stomatološke ordinacije. U slučaju da se zaposli novi uposlenik koji možda nema pristup svim funckionalnostima aplikacije ili pak novi stomatolog koji je specijalizovan za određeno područje, morat ćemo osigurati da se kreira i određena klasa za tu osobu pri loginu odnosno pristupu aplikaciji. Zbog toga bi dodali novi interfejs (Creator), koji bi kontrolisao u zavisnosti od unesenih podataka pri loginu, koja će se instanca klase kreirati (Doktor, Stomatološki tehničar, specijalizovani doktor, itd...).

Abstract Factory patern

Ukoliko bi se mijenjao sistem pohranjivanja stomatoloških usluga, ovaj patern bi nam mogao biti od pomoći. Trenutno stomatološke usluge nisu grupisane, nego ih samo čuvamo u kolekciji. Kada bi htjeli bolje organizovati i poboljšati preglednost koda, pohrane podataka i samim time i aplikacije, realizovali bi abstract factory patern kojim bi dijelili stomatološke usluge na određene grupe kao npr. plombe, liječenje zuba, aparatići itd. Te bi onda grupe sadržavale određene usluge (metalni aparatić, gumeni

aparatić, providni aparatić). Ova grupacija bi nam optimizovala i sortiranje u slučaju ogromnog broja usluga. Interfejsi bi bili kreirani u skladu sa postojećim grupama kao i njihove respektivne klase.

Builder patern

Naša aplikacija će koristiti baze podataka za backup. Da to nije bio slučaj, te da nemamo neki način da sačuvamo stanje koje je prethodno bilo, mogli bismo imati neku klasu OrdinacijaBuilder, unutar koje bismo imali metode koje bi nam pravile kopije svega onoga što već imamo - napraviKopijuKartona, napraviKopijuPacijenata, napraviKopijuStomatološkihUsluga, ... te bismo tako pravili kopije svega što trenutno imamo da bismo mogli exportovati iz naše aplikacije kao neki fajl. Da bismo to sve sastavili, mogli bismo ili dodati metodu getResult pa da nam vrati tako kreiranu kopiju ordinacije, koja zapravo sadrži sve potrebne podatke pa da u nekoj klasi kao što je npr. Export, da sve podatke spakujemo i zapišemo ili u dokument ili na neki način sačuvamo tako da bismo mogli rekreirati stanje prije zatvaranja aplikacije. Cijela ordinacija je jako kompleksan objekat, te bi bilo previše komplikovano kreirati je cijelu odjednom - konstruktori bi imali jako mnogo parametara, imali bismo mnogo ugnježdenih objekata i atributa. Korištenjem builder paterna izbjegavamo te probleme te rješavamo na najlakši način mogući.