STRUKTURALNI PATERNI

Adapter patern:

Sortiranje i pretraživanje adresara su dvije veoma korisne funkcionalnosti koje bi svakako olakšale korištenje adresara a s tim u vezi i cijele aplikacije. Za slučaj kada želimo da dodamo te dvije funkcionalnosti u klasu Adresar koristimo Adapter patern. Potrebno je da koristimo Adapter patern jer ne želimo da mijenjamo originalnu klasu Adresar i time ne ugrožavamo integritet cijele aplikacije.

Facade patern:

U našoj aplikaciji, korisnik koji je prijavljen kao doktor treba imati pristup i stomatološkim kartonima, stomatološkim uslugama, knjizi protokola i adresaru. Svaku od tih klasa možemo posmatrati kao poseban podsistem onoga što doktor može da radi. Tako bismo u klasi Facade mogli imati svaki od tih 4 podsistema, a svaki od tih podsistema ima svoje vlastite operacije. Sve one operacije koje doktor može da izvodi možemo da stavimo u klasu Facade jer u njoj možemo držati operacije sastavljene od različitih dijelova podsistema. Na taj način sakrivamo kompleksnost sistema i pružamo korisniku (doktoru) interfejs kojim on može da pristupa svakom od tih podsistema.

Decorator patern:

Ukoliko bismo željeli da naznačimo da je neki karton zatvoren, dodali bismo datum zatvaranja kao atribut u klasi stomatološki karton te bismo mogli da stavimo neki datum umjesto vrijednosti null. U slučaju da je vrijednost atributa različita od null onda je taj karton zatvoren, te razlog zatvaranja kartona može biti odlazak pacijenta u drugu ordinaciju i prestanak dolaska u trenutnu, u slučaju da je pacijent koji se liječio u toj ordinaciji preminuo ili u slučaju forenzičkih istraživanja kada zubni karton pacijenta igra veliku ulogu, a promjena istog bi mogla dovesti do gubitka dokaza. Tada bismo dinamički mogli dodati funkcionalnosti koje bi mogli vraćati usluge neovisno od obavljenog termina kao i sve podatke o tom pacijentu. U tom slučaju bismo imali metodu kojom bi se mogli printati svi podaci koji se nalaze u kartonu, te bismo također mogli dodati i vraćanje svih slika vilica pacijenta po određenim terminima. Za dinamičko dodavanje ovih funkcionalnosti koristimo Decorator patern.

Bridge patern:

U našem slučaju imamo apstraktnu klasu Korisnik. U trenutnom rješenju nemamo mogućnost promjene e-maila. Ukoliko želimo da dodamo mogućnost promjenu e-maila onda možemo koristiti bridge patern tako da postoji neki interfejs, npr. 'Promjena' unutar kojeg imamo dvije mogućnosti. Jedna od mogućnosti bi bila ona koju već posjedujemo (promjena šifre), a druga mogućnost koju bismo dodali - u vidu klasa sa istoimenim funkcionalnostima. 'Promjena' služi kao most te tako možemo mijenjati oba tipa klasa bez da ostavljaju uticaj jedna na drugu.

Proxy patern:

Pošto je namjena Proxy paterna da omogući pristup i kontrolu pristupa stvarnim objektima, mi to možemo iskoristiti kod apstraktne klase Korisnik. To je obično mali javni surogat objekat koji predstavlja kompleksni objekat čija aktivizacija se postiže na osnovu postavljenih pravila. Pravilo koje bismo mi uradili je tačno unesena šifra. Ukoliko šifra nije tačna, tada neko ko je pokušao ući ne bi trebao da to može postići, te tako kontrolišemo pristup objektima, tačnije instancama klase Korisnik (kod nas su to klase koje su naslijeđene iz klase Korisnik – Pacijent i Doktor). Subject klasa ima iste metode kao i klasa Korisnik. Također i Proxy ima iste metode kao i Korisnik.

Composite patern:

Ovaj patern koristimo kada trebamo da grupu objekata tretiramo isto kao i pojedinačne objekte tog tipa. Ukoliko bismo htjeli da kartone možemo brisati iz aplikacije, grupu kartona bismo također trebali moći obrisati. Time smo uvidjeli da nam je potreban composite patern, jer želimo da i instancu klase Stomatološki karton a i kolekcija objekata tipa Stomatološki karton možemo brisati. Ovo bismo mogli implementirati tako što bi unutar Stomatološki karton imali operaciju za brisanje, a u Composite bismo imali listu objekata tipa Stomatološki karton, te bismo također imali i operaciju za brisanje. IComposite bi nam bilo zajedničko ponašanje za oba tipa (i za jedan objekat i za kolekciju), što je u našem slučaju operacija za brisanje.

Flyweight patern:

Veoma je bitno da iskoristimo RAM memoriju što je moguće efikasnije. Jedan od načina da to postignemo je da stavimo što je više moguće objekata u raspoloživu RAM memoriju tako što ćemo dijeliti stanja između više objekata umjesto da čuvamo sve podatke o svim objektima. S tim u vezi, imamo dva stanja – glavno i sporedno. Glavno stanje je uvijek isto za sve određene objekte, dok sporedno nije uvijek isto za sve određene objekte te klase. Ukoliko uzmemo klasu LoyalKartica, ona posjeduje neke zajedničke atribute za sve objekte te klase, te bismo mogli da ih sačuvamo i da zapravo to čini dijelove koje su iste za sve klase – glavno stanje (trajanje kartice je isto za sve korisnike, iznos popusta je isti za sve korisnike, radilo se to o fiksnom popustu ili popustu na broj usluga). Za sporedno stanje bismo imali dijelove koji zapravo čine ono što se može razlikovati odnosno nisu isti za sve objekte te klase (id, datum kreiranja, da li je aktivna i datum isteka kartice).