STRUKTURALNI PATERNI

Adapter patern:

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

Bridge patern:

U našem slučaju imamo apstraktnu klasu Osoba. U trenutnom rješenju nemamo mogućnost promjene username-a i password-a. Ukoliko želimo da dodamo mogućnost promjene username-a i password-a onda možemo koristiti bridge patern tako da postoji neki interfejs, npr. „Promjena“ unutar kojeg imamo dvije mogućnosti.
Jedna od mogućnosti bi bila promjena šifre, a druga mogućnost promjena username.
'Promjena' služi kao most te tako možemo mijenjati oba tipa klasa bez da ostavljaju uticaj jedna na drugu.

Facade patern:

U našoj aplikaciji, korisnik koji je prijavljen kao doktor treba imati pristup i kartonima, terminima, nalazima. 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 3 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.

Proxy patern:

Pošto je namjena Proxy paterna da omogući pristup i kontrolu pristupa stvarnim objektima, mi to možemo iskoristiti kod apstraktne klase Osoba. 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. Proxy klasa ima iste metode kao i klasa Osoba.

Composite patern:

Ukoliko zamislimo da imamo neke različitosti između kartona za djecu i kartona za odrasle, imali

bismo dva tipa kartona. Klasa Karton bi u tom slučaju bila apstraktna, dok bi klase Dječiji karton i Karton za odrasle bile naslijeđene iz apstraktne klase Karton.
Ukoliko bismo htjeli da imamo metodu za printanje svih informacija koje se nalaze u kartonu ta metoda bi trebala imati različitu implementaciju u ovisnosti o kojem tipu kartona se radi. Razlog tome je što bismo npr. u kartonima za djecu imali neke informacije koje ne bismo imali unutar kartona za odrasle. Tada bismo listu kartona mogli obrađivati jednom metodom a ta naša metoda bi imala različite implementacije u zavisnosti od tipa kartona.

Decorator patern:

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
Kada bismo željeli da skeniranje QR koda može da se radi i preko slike, tada bismo dodali da se ta slika može i urediti (npr. Da korisnik može izrezati sliku). Metode (npr postaviSliku(), izrežiSliku()) bi implentirali unutar interfejsa ISlikaQR.

Flyweight patern:

Postoje situacije u kojima je potrebno da se omogući razlikovanje dijela klase koji je uvijek isti za sve određene objekte te klase (tzv. glavno stanje (engl. intrinsic state)) od dijela klase koji nije uvijek isti za sve određene objekte te klase (tzv. sporedno stanje (engl. extrinsic state)).
Osnovna namjena Flyweight paterna je upravo da se omogući da više različitih objekata dijele isto glavno stanje, a imaju različito sporedno stanje.
Mi bismo ovaj patern mogli iskoristit u slučaju printanja nalaza gdje bismo imali pristup glavnom stanju dokumenta (koji bi bio podaci koji se ne mijenjaju) i u njega bismo samo dodavali podatke specifične za taj nalaz.