KREACIJSKI PATERNI

1. Singleton pattern

Uloga Singleton paterna je da osigura da se klasa može instancirati samo jednom i da osigura globalni pristup kreiranoj instanci klase. U našem sistemu bismo mogli iskoristiti ovaj pattern pri pristupu bazi podataka, kako bi riješili problem mogućeg istovremenog pristupa od strane admina i medicinskog osoblja. Klasa koja bi nam omogućila pristup bazi bi bila singleton. Također, ako bismo željeli da imamo samo jednog admina, onda bi se mogao primijeniti singleton pattern na klasu Admin. Međutim kako želimo da imamo mogućnost kreiranja više admin računa, ovaj pattern nećemo primijeniti u našem sistemu. Isto vrijedi i za ostale klase- nije moguće imati singleton pattern jer svaka od klasa sadrži unikatne podatke koji razlikuju jednu instancu od druge.

2. Prototype pattern

Uloga Prototype paterna je da kreira nove objekte klonirajući jednu od postojećih prototip instanci (postojeći objekat). Ako je trošak kreiranja novog objekta velik i kreiranje objekta je resursno zahtjevno tada se vrši kloniranje već postojećeg objekata. U našem sistemu ovaj pattern možemo primijeniti pri rezervaciji termina testiranja. Klasa RezervacijaTestiranja ima mnogo atributa koji će u dosta slučajeva biti isti, kao i klasa Test koju koristimo pri rezervaciji. Kako bi se smanjio trošak, postojeći objekat bi mogli klonirati i promijeniti podatke koji se razlikuju. Također, ako bi imali različite vrste pacijenata mogli bi klonirati postojeću instancu pacijenta, kada bi prelazio na viši/niži nivo.

3. Factory Method pattern

Uloga Factory Method paterna je da omogući kreiranje objekata na način da podklase odluče koju klasu instancirati. Različite podklase mogu na različite načine implementirati interfejs. Factory Method instancira odgovarajuću podklasu(izvedenu klasu) preko posebne metode na osnovu informacije od strane klijenta ili na osnovu tekućeg stanja. U našem sistemu korisnici imaju pristup različitim izvještajima zavisno od njihove uloge. Stoga, ovaj pattern možemo primijeniti kod izvještaja (prikaz odgovarajućih u zavisnosti od uloge korisnika – admin, medicinska sestra, pacijent). Kreirali bi klasu sa metodom factoryMethod() koja na osnovu usera odlučuje koje će izvještaje prikazati.

4. Abstract Factory pattern

Abstract Factory patern omogućava da se kreiraju familije povezanih objekata/produkata. Ukoliko postoji više tipova istih objekata te različite klase koriste različite podtipove, te klase postaju fabrike za kreiranje objekata zadanog podtipa bez potrebe za specificiranjem pojedinačnih objekata. Na ovaj način se, korištenjem nasljeđivanja, ukida potreba za postojanjem if-else uslova jer određeni tip fabrike sadrži određene tipove objekata i zna se tačno koju podklasu će instancirati. U našem sistemu nemamo puno tipova klasa koje su naslijeđene iz neke apstraktne bazne klase (većinu stvari smo riješili preko enum klase npr. vrsta, namjena testa...). Izvještaje smo implementirali preko interfejsa, međutim ako bi umjesto iStatistika napravili apstraktnu klasu Izvjestaj, iz koje bi izveli klase IzvjestajAdmin, IzvjestajOsoblje, IzvjestajGost, te dodali interfejs u kojem bi imali metodu dajStatistiku, onda bi na osnovu toga koji od korisnika poziva metodu slali različite izvještaje.

5. Builder pattern

Uloga Builder paterna je odvajanje specifikacije kompleksnih objekata od njihove stvarne konstrukcije. Isti konstrukcijski proces može kreirati različite reprezentacije. U našem sistemu možemo ovaj pattern primijeniti kod RezervacijeTestiranja. Napravili bi „Direktor“ klasu koja bi imala sljedeće metode: napraviTest(), plati(), dajTermin() (koja daje termin na bilo kojoj lokaciji)...Također, trebali bi dodati builder interfejs koji bi sadržavao metode za dodavanje potrebnih informacija i builder klase koje mi implementirale ovaj interfejs.