

## **Proxy – Patern Strukture**

### **Opis**

Proxy objekat radi kao surogat za stvarne objekte. Koristi se za omogućavanje pristupa i kontrolu pristupa stvarnim objektima. Manipulacija nad nekim objektom je dozvoljena ukoliko je neki uslov ispunjen ili ako korisnik npr. nema pravo pristupa traženom objektu.

### **Problemi koje rješava**

Kontrola pristupa: Proxy omogućava kontrolu pristupa stvarnim objektima, što znači da se određeni objekti mogu koristiti samo ako korisnik ispunjava određene uvjete ili ako ima odgovarajuće ovlasti.

### **Primjena**

Proxy uzorak koristi se u aplikaciji VitalFlow za kontrolu pristupa funkciji zakazivanja termina donacija, tj. pruža dodatnu logiku za upravljanje situacijama u kojima donor uzastopno propušta zakazane termine. Dakle ukoliko donor uzastopno propusti dva ili više zakazana termina, onda on u određenom vremenskom periodu neće biti u stanju da zakaže bilo kakav termin.

## **Facade – Patern Strukture**

### **Opis**

Facade je strukturni patern dizajna koji pruža jednostavan interfejs za kompleksan sistem klasa ili podsistema. Omogućuje klijentima da jednostavno koriste složene operacije ili funkcionalnosti bez potrebe za detaljnim poznavanjem unutaršnjih procesa.

### **Problemi koje rješava**

Jednostavno obavješćavanje donora na osnovu stanja zaliha: Donori primaju obavijesti o nedostatku krvi na jednostavan i učinkovit način.

### **Primjena**

U kontekstu aplikacije VitalFlow, Facade patern bi mogao biti primijenjen za pojednostavljivanje procesa obavješćavanja donora o nedostatku određene krvne grupe. Umjesto da svaki zaposlenik direktno pristupa logici provjere stanja zaliha i slanja notifikacija, Fasadu bi mogli koristiti kao jednostavnu tačku za pristup tim operacijama. Fasada bi se brinula o provjeri stanja zaliha za

određenu krvnu grupu i slanju notifikacije ako je potrebno, omogućavajući tako lakši i čistiji pristup ovim funkcionalnostima.

## **Decorator – Patern Strukture**

### **Opis**

Decorator patern je strukturni patern dizajna koji omogućuje dinamičko dodavanje dodatnih funkcionalnosti ili ponašanja postojećim objektima bez mijenjanja njihove osnovne strukture. Ovaj patern omogućuje fleksibilno proširenje funkcionalnosti sistema tako da se nove funkcionalnosti mogu dodati ili ukloniti neovisno o kodu.

### **Problemi koje rješava**

Decorator za označavanje hitnosti: Možemo dodati mogućnost označavanja hitnosti kliničkih zahtjeva, što pomaže u prioritetizaciji zahtjeva.

Decorator za slanje notifikacija: Možemo dodati funkcionalnost slanja notifikacija putem e-maila kada se forma "Klinički zahtjevi" submit-a, a također možemo proširiti slanje notifikacija donorima putem drugim aplikacija (npr. Facebook) kada nestane zaliha.

### **Primjena**

Decorator se koristi za proširivanje funkcijonalnosti obavješćavanja donora o nedostatku grupe krvi, proširivanje funkcijonalnosti obavješćavanja klinika kada one popune formu, te za dodavanje mogućnost označavanja hitnosti kliničkih zahtjeva, što pomaže u prioritetizaciji zahtjeva.

## **Adapter – Patern Strukture**

### **Opis**

Adapter patern se koristi kada treba omogućiti nekompatibilne interfejse da rade jedan sa drugim posredstvom Adapter klase koja impementira klijent interfejs i služi kao omotač za servis koji je potrebno adaptirati.

## **Problemi koje rješava**

Migracija podataka: Kada aplikacija treba migrirati s jedne baze podataka na drugu, npr. kada bi naša aplikacija trebala migrirati s SQL Servera/MySQL-a na MongoDB, suočili bismo se s različitim interfejsima i načinima komunikacije s bazom podataka.

## **Primjena**

Adapter patern rješava gore navedeni problem omogućujući prilagodbu operacija i zahtjeva aplikacije na interfejs koji podržava nova baza podataka, čime se olakšava migracija i održavanje aplikacije.

## **Bridge – Patern Strukture**

### **Opis**

Bridge pattern omogućuje odvajanje apstrakcije (klijentske logike) od implementacije (komunikacija s vanjskim sistemom klinike).

### **Problemi koje rješava**

Ako aplikacija komunicira s vanjskim sistemom klinike ili sličnim vanjskim servisom, važno je odvojiti klijentsku logiku (kako se aplikacija koristi) od detalja implementacije (kako se komunicira s vanjskim sistemom). Bridge pattern rješava ovaj problem omogućujući da se apstrakcija (Bridge) koristi za komunikaciju s vanjskim sistemom, dok konkretne implementacije bridge-a rade s detaljima komunikacije.

### **Primjena**

Bridge pattern u VitalFlow aplikaciji bi olakšavao prilagođavanje i proširenje komunikacije s vanjskim sistemom bez mijenjanja klijentske logike.

## **Composite – Patern Strukture**

### **Opis**

Composite pattern se često koristi kada treba raditi s objektima koji predstavljaju hijerarhijske strukture, poput stabla ili strukture odjeljenja u organizaciji.

## **Problemi koje rješava**

Jednostavno praćenje grupnih donacija: Kada bi omogućili grupne donacije, ukoliko želimo da lakše računamo koje sale su slobodne u kojim terminima, te koji donor pripada kojoj krvnoj grupi.

## **Primjena**

Ovaj pattern bi mogli implementirati ukoliko omogućimo grupne donacije, na ovaj način bi imali Composite klasu koja sadrži listu korisnika i leaf klasu kao pojedine korisnike različitih vrsta. Composite i leaf klase bi implementirale interfejs sa operacijama koje vršimo nad pojedinim objektima.