Paterni ponašanja

1) **Observer pattern**
   Ovaj patern se koristi kada više objekata treba automatski da bude obaviješteno o promijenama stanja nekog drugog objekta. Najčešće se koristi kod neke pretplate.

   U našem sistemu dobar način da se ovaj patern iskoristi je da se korisniku koji je član fitnesa i koji je rezervisao određeni termin ta trening javi ako zbog tehničkih ili nekog drugog razloga mora doći do promijene termina. Također možda ako željeni trener sazna da nije dostupan nakon prvobitnog rezervisanja termina.

2) **Strategy patern**
   Ovaj patern koristimo kada imamo više različitih alogoritama za realizaciju nekih zadataka i te algoritme možemo zamijeniti jedan sa drugim u toku izvršavanja programa.

   Jedna od mogućih primjena u našem sistemu je na primjer algoritam za provjeru dostupnosti termina, da ga ponekad po potrebi zamjenimo sa npr. algoritmom koji bolje radi sa manjim brojem datume ili algoritmom koji radi bolje sa većim brojem datuma.

3) **State patern**
   State patern se koristi kada objekat treba da mijenja ponašanje zavisno od trenutnog stanja.

   U našem slučaju naš korisnik ima različite uloge: Član, Trener, Administrator i svaki od njih ima različita prava. Napravili bismo apstraktnu klasu

UlogaKorisnik sa metodom obradiNarudzbu(), a konkretne klase AdminUloga, ČlanUloga, TrenerUloga implementiraj različita ponašanja.

4) **Iterator patern**

Iterator patern omogućava pristup elementima kolekcije bez izlaganja unutrašnje strukture.

U našem fitness centru može se koristiti za prikaz dostupnih odjevnih predmeta u fan shopu.

5) **Template method patern**

Ovaj patern se koristi kada imamo algoritam koji sadrži fiksne korake ali neki od njih treba da budu redefinisani u podklasama.

U našem slučaju to bi se moglo koristiti za izračun cijene narudžbe iz fan shopa gdje bi se određeni dijelovi razlikovali zavisno koji je tip korisnika.

6) **Chain of responsibility patern**

Ovaj patern omogućava lančano obrađivanje zahtjeva – svaki objekat u lancu može obraditi zahtjev ili ga proslijediti dalje.

U fitness centru primjena može biti za rezervaciju treninga. Tokom rezervacije korisnik unosi trenera i datum koji mu odgovara te željeni cilj. Svaka od ovih provjera može biti dio lanca odgovornosti.
Kreiramo apstraktnu klasu ValidacijaRezervacije sa metodom Validiraj(rezervacija n) gdje svaka konkretna klasa: ValidacijaTermina, ValidacijaTrenera… vrši svoj dio provjere i ako je sve uredu šalje zahtjev dalje.

7) **Mediator patern**

Patern se koristi za centralizaciju komunikacije među objektima.

U fitness centru bismo to koristili za upravljanje interakcijama između modula Korisnik, Termin, Trening…

U implementaciju napravimo klasu Imediator klasu ProcesRezervacijeMediator koja koordinira interakcije, npr. nakon što član odabere željeni termin i trenera vrši se provjera dostupnosti.

8) **Visitor patern**

Ovaj patern omogućava definisanje novih operacija nad objektima bez mjenjanja njihovih stanja.

Ovaj patern bismo mogli iskoristiti koji od termina do kraja mjeseca su nam ostali slobodni, koliko je artikala iz fan shopa ostalo na raspolaganju, koliko je novih članova sa trenutni mjesec itd.

9) **Interpreter patern**

Ovaj patern se koristi kada imamo jezik za specifičnu domenu I mi želimo da interpretiramo i izrvšavamo instrukcije u tom jeziku.

U našem slučaju to se može izvesti ako omogućimo naprednu pretragu pomoću vlastitog mini jezika. Npr. član želi da vidi slobodne termine u narednih mjesec dana te unese npr <10.7 i algoritam počevši od današnjeg datuma pregleda na dostupnost sve datume manje od zadanog.

10) Momento patern

Ovaj patern omogućava spremanje I kasnije vraćanje prethodnog stanja objekta bez narušavanja enkapsulacije.

U našem slučaju ovaj patern možemo koristiti za funkcionanost “poništi promjenu”. Npr ako član želi da promijeni svoj email ili šifru onda će sistem sačuvati stare podatke za logiranje.

11) Command patern

Command patern omogućava da svaki zahtjev tretiramo kao poseban objekat. Na taj način zahtjeve možemo lahko proslijediti, čuvati, staviti na red čekanja…

U našem sistemu bismo to mogli implementirati tako što npr kada se trener loguje svaki njegov novi trening stavimo u zamišljenu korpu. I svaki taj trening tretiramo kao poseban objekat komande (DodajTreningCommand, UkloniTreningCommand). Kreiramo interfejs Icommand sa metodama dodaj() i briši(). Svaka konkretna komanda implementira ove metode.