STRUKTURALNI PATERNI

1. ADAPTER PATERN

U klasi RegistrovaniKorisnik imamo metodu getListaPrijatelja() kojom korisnik može pregledati sve svoje prijatelje. Mogli bismo napraviti adapter kojim ćemo vraćati sortiranu listu prijatelja po broju zajedničkih događaja, tj. prvo mjesto da bude prijatelj koji je prisustvovao najviše događaja kao i korisnik. Patern bismo implementirali tako što ćemo naslijediti adapter iz klase RegistrovaniKorisnik i implementirati interfejs IPrijatelji.

2. FACADE PATERN

Facade paternom ćemo objediniti metode getRezervisaniDogađaji() iz klase RegistrovaniKorisnik i getPrijavljeneKorisnike() iz klase Događaj. Nova metoda u facade klasi će omogućiti korisniku da ima uvid u sve prijavljene korisnike na odabrani događaj bez obzira jesu li mu prijatelji ili ne.

3. DECORATOR PATERN

Ovaj patern bi se mogao primijeniti ako bismo krenuli u pravcu proširivanja broja tipova događaja, ako bi se dodavale mogućnosti da događaji mogu biti VIP ili ne, događaji na otvorenom ili zatvorenom, sa promocijama i bez itd. Dekorator patern bi tu poslužio jer bi se tada mogle kombinovati sve te kategorije pri kreiranju događaja. To bi se moglo implementovati na način da se kreira interfejs IDogađaj koji sadrži sve metode klase Događaj i onda se kreiraju dekorator klase koje implementiraju taj interfejs (klase VIPDogađajDekorator, PromocijaDogađajDekorator…). Bilo bi moguće napraviti kombinacije raznih tipova događaja pozivima new VIPDogađajDekorator(new PromocijaDogađajDekorator(... new Događaj() ...)).

4. BRIDGE PATERN

Ovaj patern bi se mogao primjeniti na tipove različitog slanja notifikacija. Korisnik može primiti notifikaciju putem aplikacije vezano o događajima, prijateljima i slično, a preko e-maila će dobiti notifikaciju o potvrdi registracije.

5. PROXY PATERN

Dobra upotreba ovog paterna bi bila u metodi recenzirajDogađaj(događaj: Događaj, ocjena: Integer) u klasi RegistrovaniKorisnik gdje bi se onemogućilo recenziranje događaja od strane korisnika koji nije prisustvovao istom. Da bismo to postigli dodat ćemo interfejs IRecenziraj i klasu ProxyRecenziraj koja kroz metodu autentifikacija(događaj: Događaj, korisnik: RegistrovaniKorisnik) provjerava da li je dati korisnik prisustvovao događaju.

6. COMPOSITE PATERN

Pošto postoje različiti načini prijave na sistem za korisnika i vlasnika objekta, Composite patern bismo iskoristili tako što bismo napravilii interfejs IPrijavi, te metodu prijava(korisničkoIme:String, lozinka:String) koja bi se različito implementirala u klasi RegistrovaniKorisnik i VlasnikObjekta.

7. FLYWEIGHT PATERN

- Pošto korisnik ima mogućnost postavljanja profilne slike, u slučaju da je ne želi postaviti, koristit ćemo unaprijed zadanu sliku sa našeg sistema, tj. avatar. Na taj način će se koristiti ista slika za sve korisnike koji nisu postavili vlastitu, čime smanjujemo korištenje memorijskih resursa. Ovo ćemo postići kreiranjem interfejsa ISlika koji vraća slike svih korisnika. Postavljena slika korisnika će biti u klasi Slika, a avatar u klasi Avatar. Klase Slika i Avatar implementiraju interfejs ISlika.
- Ovaj patern može pomoći u slučaju kreiranja događaja koji ima iste ili slične osobine kao ranije kreiran događaj, npr. ako novi događaj ima isto dobno ograničenje, napomene, opis i tip, a drugačiji termin, možemo iskoristiti flyweight i na taj način smanjiti potrebu za memorijom.