Paterni ponašanja

- ## STRATEGY PATTERN

Uloga Strategy paterna je izdvajanje algoritma iz matične klase u posebne klase. Najpogodniji je kada za jedan isti problem možemo iskoristiti dva ili više različitih algoritama tj. strategija. Ovaj patern omogućava klijentima izbor jednog od primjenjivih algoritama.

Strategy patern bismo mogli iskoristiti za stvaranje različitih strategija pretraživanja registrovanih korisnika koji su popunili obrazac, kako bismo omogućili administratoru olakšano ispunjavanje dodatnih zahtjeva VIP korisnika to jeste pomoć pri nalaženju korisnika koji bi najbolje ispunili iste. Recimo ukoliko VIP korisnik kao dodatni zahtjev navodi da njegov/njen potencijalni partner mora željeti djecu, onda administrator po tom kriteriju može naći korisnike koji ispunjavaju taj zahtjev, te među njima ručno pronaći najbolje podudaranje. Druga situacija bi mogla biti ako VIP korisnik zahtjeva da njegov/njen partner ima završen određeni nivo školovanja i to je već druga strategija za pretraživanje korisnika.

Patern bi realizovali tako što bi se u klasi RegistrovaniKorisnik i Sistemu mogle birati određene imolementirane strategije u okviru istoimenog atributa “strategija”. Za ovo je neophodan I dodatni interfejs Istrategija koji implementira sve klase postojećih strategija.

- ## STATE PATTERN

State patern je zapravo dinamička verzija prethodnog paterna. On podrazumijeva da objekat mijenja način ponašanja na osnovu trenutnog stanja promjenom podklase unutar hijerarhije klasa.

U našem sistemu State pattern možemo iskoristiti za smanjivanje i eventualno ukidanje cijena registrovanim korisnicima na osnovu toga koliko je moralo vremena proći prije nego smo im uspjeli pronaći idealnog partnera. S obzirom na činjenicu da se plaćanje za izvršenu uslugu vrši tek nakon što su predloženi potencijalni partneri, ovo bi bilo moguće realizovati. Recimo ukoliko je korisnik na spajanje morao čekati duže od prosječnog vremena potrebnog za isto(npr. mjesec dana), ažurira mu se cijena, što se nastavi i svakih sljedećih mjesec dana do nekog perioda od recimo godine kada usluga postaje besplatna u cilju zadržavanja interesovanja klijenta jer je možda on upravo idealan partner za nekog ko trenutno popunjava obrazac.

Za Context klasu bismo mogli uzeti Plaćanje, te interfejs IPlaćanje iz kojeg bi se naslijedile plaćanjePrviMjesec, plaćanjeDrugiMjesec, … kao i plaćanjeNakonGodinu. Plaćanje bi se ažuriralo samo za svakog korisnika kako vrijeme prolazi.

- # TEMPLATE METHOD PATTERN

Uloga Template Method paterna je izdvajanje pojedinih koraka algoritma u odvojene podklase. Struktura algoritma ostaje nepromijenjena.

Ovaj patern bismo mogli iskoristiti za pomoć administratoru pri određivanju prioriteta odobravanja matcheva. Recimo ukoliko imamo dvije ženske osobe pri čemu prva ima 4 potencijalna matcha s jednakim ili gotovo jednakim procentom, a nije VIP korisnik, to jeste nema pravo birati između njih, a druga ima samo jedno poklapanje sa zadovoljivim procentom, tada bi bilo logično spojiti prvo koisnicu broj 2, te onda korisnicu broj jedan. Na ovaj način postigli bismo da su obje uparene. Isto tako, logično bi bilo da se administratoru prikazuju redom potencijalni match-evi onako kako su pronađeni, međutim, ukoliko bi ih bilo previše moglo bi doći do toga da dok odobrava ostale, jedan od korisnika upravo prelazi prvi mjesec čime gubimo novac radi jednog sata ili dana. Da bi postigli najveću efikasnost, administratoru treba omogućiti različite vrste prikaza uparenih korisnika recimo prikaži prvo one koji uskoro ulaze u novi mjesec, ili prikaži prvo one sa samo jednim poklapanjem.

Da bismo ovo postigli potreban nam je interfejs Iprikaz sa metdom sortiraj, zajedno sa različitim klasama za sortiranje koje bi nam predstavljale AnyCass i implementirale taj interfejs. Postojala bi super klasa za korisnika sa obicnom sortiraj metodom koja bi bila template i podklase za različite vrste korisnika po broju potencijalnih matcheva koji bi imali različito implementirane metode za za poređenje po uslovu.

- # OBSERVER PATTERN

Uloga Observer paterna je uspostavljanje relacija među ovisnim objektima tako da jedni o drugima primaju obavještenja.

Ovo bismo mogli iskoristiti za obavještavanje već uparenih korisnika o aktivnosti njihovih match-eva, tj. korisnik bi bio obaviješten svaki put kada bi se njegov potencijalni partner ulogovao u aplikaciju ili kada bi recimo promijenio sliku profila ili neku drugu informaaciju o sebi.

U ovu svrhu trebala bi nam nova Subject i Observer klasa gdje prva mijenja svoje stanje i o tome obavještava drugu. dodali bismo Iobserver sa jednom metodom – update koja bi se pozivala kada bi bilo koja instanca Subject klase bila promijenjena.

- ITERATOR PATTERN

Uloga Iterator paterna je omogućavanje sekvencijalnog pristupa elementima kolekcije bez poznavanja o strukturi kolekcije.

Ovaj pattern bismo mogli iskoristiti za listu mogućih odgovora na svako pitanje, pa bi kasnije jednostavno mogli prolaziti kroz te odgovore i ustanoviti recimo koji procenat korisnika je odgovorio na neko pitanje s nekim određenim odgovorom.

Za ovo bi nam bio potreban OdgovoriIterator sa privatnim atributom lista odgovora. Također bismo trebali interfejs koji implementira IEnumerable sa metodom getEnumeration.

- CHAIN OF RESPONSIBILITY PATTERN

Uloga Chain of responsibility paterna je da se jedan kompleksni proces obrade razdvoji na način da više objekata na različite načine procesira primljene podatke.

Ovo je idealno za ažuriranje bračnog statusa o braku sklopljenom putem aplikacije radi izbjegavanja nedosljedne statistike. Da bi izmjena statusa bila validna, potrebna je potvrda od partnera s kojim je korisnik spojen.

Jedan korinsik šalje zahtjev drugom korisniku, pa su nam potrebni PotvrdaZahtjeva, Ihandler, te klase koje nasljeđuju klasu PotvrdaZahtjeva koje će zahtjeve obrađivati.

- MEDIJATOR PATTERN

Uloga Medijator paterna je enkapsulacija protokola za komunikaciju između objekata dozvoljavajući im komunikaciju bez međusobnog poznavanja interne strukture objekta.

Ovaj patern bismo mogli iskoristiti za provjeru koji korisnici ocjenjuju i komentarišu aplikaciju s obzirom na činjenicu da ne bi imalo smisla da registrovani korisnik kojin nije uopće popunio i poslao obrazac daje ocjene i komentare o iskustvu.

Bilo bi nam potrebno da klasa RegistrovaniKorisnik ima atribut tipa IMedijatorRecenzija, kao i istoimeni interfejs. IMedijatorRecenzija bi imao metode koje bi vršile provjeru da li je korisnik kompetentan za ostavljanje recenzija, te sadržaj komentara gdje bi komentari koji sadrže neku vulgarnu riječ trebali biti neprihvaćeni.