PATERNI PONAŠANJA

Dijagram klasa sa implementiranim paternima ponašanja se nalazi u folderu “Paterni” i u folderu sa dijagramima klasa pod nazivom “DijagramKlasaSaSvimPaternima”. Napominjem da su neki paterni stavljeni na dijagram zbog pokazivanja kako izgledaju na dijagramu, ali ne znači da će svi biti implementirani (konkretno, postoje preklapanja paterna ponašanja i kreacijskih paterna na nekim mjestima, što znači da će se za implementaciju razmotriti najbolje rješenje).

Elementi paterna ponašanja su na dijagramu obojeni plavom bojom, kao i u tabeli.

NAZIV PATERNA	OPIS
STRATEGY PATERN	Strategy patern služi kako bi se različite implementacije istog algoritma izdvojile u različite klase, te kako bi se omogućila brza i jednostavna promjena implementacije koja se želi koristiti u bilo kojem trenutku. Na ovaj način omogućava se i jednostavno brisanje ili dodavanje novog algoritma koji se može koristiti po želji. S obzirom da ovaj patern rješava problem razbacanih if-else struktura u programu, u našem slučaju bi se ovaj patern mogao iskoristiti kod odabira načina plaćanja, odabira vrste fizičkog lica i odabira vrste karte.
STATE PATERN	State patern omogućava objektu da mijenja svoja stanja, od kojih zavisi njegovo ponašanje. Sa promjenom stanja objekt se počinje ponašati kao da je promijenio klasu. Stanja se ne mijenjaju po želji klijenta, već automatski, kada se za to steknu uslovi. U našem slučaju, klasa obavijesti se sastoji od puno atributa tipa bool i puno metoda koje ovise o ovim atributima. Ovakvo stanje klase bi u implementaciji izazvalo puno if-else struktura. Zbog toga, bilo bi praktičnije dodati interfejs IObavijestMode koji će sadržavati metode čija je implementacija različita za različite vrste obavijesti koje se šalju korisnicima. Dodati ovaj interfejs kao atribut klase Obavijest i sa njim ćemo zamijeniti sve ove bool atribute. Dalje, treba povezati ovaj interfejs sa postojećim klasama za vrste obavještenja koja postoje u aplikaciji.
TEMPLATE METHOD PATERN	Template method patern služi za omogućavanje izmjene ponašanja u jednom ili više dijelova. Najčešće se primjenjuje kada se za neki kompleksni algoritam uvijek trebaju izvršiti isti koraci, no pojedinačne korake moguće je izvršiti na različite načine. U biti,ovaj patern je već

	primijenjen kod nas,jer kao “template” za glave aktere u sistemu je klasa Korisnik i svi korisnici koji postoje i koji se izvode iz ove klase, imaju zajedničke osobine, ali i dosta dijelova u kojima je ponašanje izmijenjeno i koje se izvršava na različite načine.
OBSERVER PATERN	Observer patern služi kako bi se na jednostavan način kreirao mehanizam pretplaćivanja. Pretplatnici dobivaju obavještenja o sadržajima na koje su pretplaćeni, a za slanje obavještenja zadužena je nadležna klasa. Na ovaj način uspostavlja se relacija između klasa kako bi se mogle prilagoditi međusobnim promjenama. Ovaj patern se već nalazi u našem sistemu kod klase Obavijest, sa tim da ako se pogleda dijagram klasa, ova klasa je zadužena da se svakako šalju obavijesti, ali bismo mogli ubaciti mogućnost da se korisnik pretplati na obavještenja iz neke ustanove. Dodat ćemo interfejs IPretplatnik koji će sadržavati metodu za pregled kompletnog događaja u aplikaciji i metodu koja će omogućavati rezervaciju nakon primljene obavijesti. Trebamo u klasu Ustanova dodati atribut lista pretplatnika i pretplatnici će nasljeđivati interfejs IPretplatnik. Interfejs će se trebati povezati sa klasama za rezervaciju i za događaj. U biti, ovaj patern je kod klase Obavijest već na neki način obrađen.
ITERATOR PATERN	Iterator patern namijenjen je kako bi se omogućio prolazak kroz listu elemenata bez da je neophodno poznavati implementacijske detalje strukture u kojoj se čuvaju elementi liste. Izvedba liste može biti u obliku drveta, jednostruke liste, niza i sl., no klijentu se omogućava da na jednostavan način dolazi do željenih elemenata. Osim toga, ovaj patern preporučljivo je iskoristiti kada se za iteriranje koristi kompleksna logika koja ovisi o više kriterija. Ovaj patern možemo primijeniti kod situacije da korisnik želi da filtrira i pretražuje događaje po određenom kriterijumu. Možemo dodati interfejs ISljedeciDogadjaj, a ovaj interfejs će da naslijedi iteratore: nasumičniIterator, abecedniIterator, ustanovaIterator. Dalje, dodaćemo interfejs IKreatorIteratora, koji će biti povezan sa klasom EventManager i omogućiti će kretanje kroz listu

	događaja koji se već nalaze u bazi podataka aplikacije.
CHAIN OF RESPONSIBILITY PATERN	Chain of responsibility patern namijenjen je kako bi se jedan kompleksni proces obrade razdvojio na način da više objekata na različite načine procesiraju primljene podatke. Kreiranjem lanca obrade, pri čemu nakon vršenja parcijalne obrade svaki objekat prosljeđuje podatke narednom objektu u nizu, postiže se smanjenje kompleksnosti pojedinačnih procesa obrade i jednostavnije razumijevanje cijelog procesa. Osim toga, svakom objektu dolaze samo oni podaci koje trebaju obraditi te se na taj način izbjegava dupliciranje sadržaj. U našem slučaju postoji mogućnost da fizičko lice podnese zahtjev da objavi događaj u ime neke ustanove. Da bi se taj događaj na kraju našao u bazi podataka i u aplikaciji, taj događaj moraju da potvrde i klasa EventManager,tj.da on ispunjava zahtjeve aplikacije šta jedan zahtjev i događaj treba da ima od ključnih stvari, klasa Ustanova za koju se objavljuje događaj i klasa Admin. Zato nam trebaju klasa Zahtjev sa osnovnim atributima koje jedan zahtjev treba da ima, klasa ZahtjevPotvrda, interfejs IHandler, te klase za obradu koje će naslijediti klasu ZahtjevPotvrda, one će obrađivati zahtjev na osnovu datih informacija i one će implementirati vrituelnu metodu obradi na način da vrše pozivanje onih dijelova za koje su zaduženi.
MEDIATOR PATERN	Mediator patern namijenjen je za smanjenje broja veza između objekata. Umjesto direktnog međusobnog povezivanja velikog broja objekata, objekti se povezuju sa međuobjektom medijatorom, koji je zadužen za njihovu komunikaciju. Kada neki objekt želi poslati poruku drugom objektu, on šalje poruku medijatoru, a medijator prosljeđuje tu poruku namijenjenom objektu ukoliko je isto dozvoljeno. Na ovaj način stvara se centralizovano mjesto kontrole objekata te onemogućavanje komunikacije između objekata u slučajevima kada je to potrebno. Ovaj patern bismo iskoristili ukoliko bi dodali mogućnost stvaranja chara za korisnike aplikacije, i da onda razgraničimo poruke za ustanovu i za fizičko lice, recimo za ustanovu bi se vršila određena filtracija, kao i za maloljetne korisnike sistema.