Paterni ponašanja

Projekat “Pixel Vrtić” obuhvata dinamičke procese kao što su registracija prisustva, slanja obavještenja roditeljima, analiza aktivnosti i upravljanje korisničkim profilima. U takvom okruženju, upotreba paterna ponašanja omogućava modularizaciju funkcionalnosti, poboljšava čitljivost i održavanje koda, olakšava testiranje, te omogućava jednostavnu nadogradnju sistema u budućnosti.

Strategy pattern

Izdvaja algoritme iz matične klase i uključuje ga u posebne klase, omogućava kreiranje porodice algoritama i zamjenu algoritama bez potrebe za izmjenom klijentskog koda. Za naš projekat moguće je izdvojiti algoritam za izračunavanje naknade koji se trenutno nalazi unutar *AdminController* u posebnu klasu *AlgoritamZaNaknadu*, te u ovisnosti od kakve usluge koristi korisnik upotrijebiti različitu vrstu algoritma ili različitu cijenu naknadu za obračun.

State pattern

Mijenja način ponašanja objekta na osnovu trenutnog stanja, odnosno ponaša se kao da mijenja klasu tokom izvršavanja programa. Na primjer, klasa *Dijete* može da bude u više različitih stanja na osnovu toga da li je *Prisutan, Odsutan* ili *Bolestan*. Na osnovu tog stanja, vrši se obrada prisustva ili se po mogućnosti mogu slati neke specijalne poruke i obavještenja roditeljima.

Template Method pattern

Definiše kostur algoritma u nadklasi, dok konkretni koraci mogu biti redefinisani u podklasama. Iz *AlgoritamZaNaknadu* moguće je razdvojiti algoritam gdje se koraci dobavljanja podataka i ispisa podataka stavljaju u podklase. Proces dobavljanja podataka, odnosno dobavljanja prisustva za određeni *roditeljId* se može staviti u posebnu klasu *DobaviPrisustva* a dio algoritma koji kreira klasu *FinancijskaEvidencija* za praćanje plaćanja staviti u klasu *KreirajFinancijskuEvidenciju*. Na ovaj način olakšava se dodavanje novih scenarija bez dupliciranja osnovne logike.

Chain of Responsibility pattern

Predstavlja listu objekata, ukoliko objekat ne može da odgovori proslijeđuje se narednom u nizu sve dok se zahtijev ne može uspješno izvršiti. Na primjer, ako *Roditelj* šalje upit o promjeni termina neke aktivnosti, kreirala bi se lista klase *Vaspitač,* zahtjev se šalje prvo vaspitaču koji je odgovoran za taj termin, ako on nije u mogućnosti da odgovori, zahtjev se šalje narednom vaspitaču i proces se nastavlja sve dok neko ne može da odgovori ili dok se ne dođe do kraja niza.

Command pattern

Omogućava enkapsulaciju zahtjeva kao objekta čime se ti zahtjevi mogu dinamički slati, stavljati u red čekanja, poništiti (undo), ponoviti (redo) ili zabilježiti za kasniju obradu, odnosno zahtjev se pretvara u samostalni objekat. Na taj način se mogu jednostavno proslijediti, staviti u red čekanja ili zabilježiti zahtjev i podržati operacije poput undo/redo. Kao što smo za chain of responsibility slali zahtjev i čekali odgovor za promjenu termina, ako niko u datom trenutku nije u mogućnosti da odgovori, zahtjev se sačuva. Kreirali bi novu klasu *ZahtjevPromjeneTermina* koja bi čuvala željenje podatke, roditelj onda može da promjeni sadržaj početnog zahtjeva i neko od vaspitača koji bude u mogućnosti može da odogovori na datu poruku. Ova klasa sadrži sve potrebne informacije (termin, razlog, roditeljId, itd.) i može se kasnije ponovno proslijediti ili izmijeniti prije slanja.

Iterator pattern

Omogućava pristup elementima kolekcije sekvencijalno bez poznavanja interne strukture kolekcije. U našem slučaju imamo dosta kolekcija podataka, kao što su liste klasa *Dijete, Roditelj, Vaspitača, Aktivnosi* ili *Obavještenja*. Možemo kreirati paterne za bilo koju od ovih lista podataka ako je potrebno. Na primjer za pregled sve djece u određenoj grupi ili za pregled svih aktivnosti koje su zadane od strane nekog vaspitača.

Mediator pattern

Enkapsulira protokol za komunikaciju među objektima dozvoljavajući da objekti komuniciraju bez međusobnog poznavanje interne strukture objekta, odnosno da nema potrebe stvaranja direktnih veza među objektima. Obzirom na veliki broj komponenti našeg sistema i potrebom za komunikaciju između klasa *Roditelj* i *Vaspitač*, *Roditelja* i *Obavještenja* i drugih, možemo uvesti medijatore za komunikaciju između navedenih pojedinačnih klasa ili jedan medijator koji bi bio odgovoran za komunikaciju između svih klasa sistema, odnosno klasu *SistemMediator*. Ona omogućava, na primjer, da kada roditelj zatraži promjenu termina, *SistemMediator* zna kojoj komponenti treba poslati poruku, odnosno vaspitaču, administraciji ili oboje bez da roditelj direktno poznaje strukturu ostalih klasa. Na taj način poboljšavamo skalabilnost i smanjujemo međuovisnost modula.

Observer pattern

Uspostavlja relaciju između objekata takvu da kad se stanje jednog objekta promijeni svi vezani objekti dobiju informaciju. Moguće je koristiti veliki broj observera o našem sistemu. Ako klasa roditelj označi da dijete sutra neće biti prisutno, svi vaspitači zaduženi za tu grupu koje dijete pohađa će dobiti obavijest o tome ili ako se odgodi neka aktivnost svi roditelji čija djeca su trebala prisustvovati toj aktivnosti će znati da se odgodilo. Tako da sve promjene koje se dese za klasu *Prisustvo* vezane klase *Dijete, Vaspitač* i *Grupa* će dobiti tu informaciju.

Visitor pattern

Omogućava dodavanje novih operacija postojećim klasama bez izmjene njihove strukture, koristeći posebnu klasu posjetioca za implementaciju novih funkcionalnosti. Ima generalno široku upotrebu, na primjer možemo kreirati klasu *DijeteVisitor* koja omogućava izvođenje različitih obrada nad instancama klase *Dijete*, bez promjene same klase. Dodajemo operacije poput *PretvoriPodatkeUString*, *PretvoriPodatkeUJSON*, *GenerisiIzvjestajODjetetu*, itd. Time omogućavamo lako proširivanje funkcionalnosti kao i podršku za različite formate izvještaja (npr. PDF).

Interpreter pattern

Podržava interpretaciju instrukcija napisanih za određenu upotrebu. Ako bi naš sistem koristio neki specijalan način filtriranja djece ili filtriranje aktivnosti, onda bi koristili interpretere da bi mogli taj tekst ili pravila koristiti u algoritmu za filtriranje. Na primjer traženja više različitih imena iz klasa *Dijete* ili *Roditelj* možemo u search baru odvojiti zarezom ili traženje imena i prezimena sa razmakom. Kreiranjem gramatike za filtriranje (npr. pretraga po imenu, prezimenu, grupi, terminu) možemo omogućiti korisnicima da sami formulišu kompleksne upite, a sistem ih interpretira i pretvara u odgovarajuće SQL upite ili filterske operacije. Takav unos bi algoritam koristio na osnovu tih zadatah pravila.

Memento pattern

Omogućava spašavanje internog stanja nekog objekta van sistema i njegovo ponovno vraćanje, bez narušavanja enkapsulacije, i kasnije po potrebi vrati. Koristi se za implementaciju funkcionalnosti kao što su undo/redo ili verzionisanje podataka. Na primjer kada administrator (ili vaspitač) izvrši promjenu u podacima klase *Dijete* ili *Roditelj*, prethodno stanje se može sačuvati kao *Memento* objekat. Ako se kasnije utvrdi da je došlo do greške (npr. greškom prebrisani podaci), korisnik može vratiti podatke na prethodno stanje klikom na dugme „Poništi izmjene“. Ovaj pristup je posebno koristan za zaštitu integriteta podataka i sprječavanje neželjenog gubitka informacija tokom uređivanja.