# DIZAJN PATERNI PONAŠANJA

***Strategy***

Strategy dizajn patern izdvaja algoritam iz matične klase i uključuje ga u posebne klase, te omogućava klijentu izbor jednog od algoritama iz familije algoritama. U pogledu našeg sistema, funkcionalnost koja podrazumijeva pronalaženje odgovarajuće korisničke statistike može biti implementirana na više načina.

To možemo implementirati:

* deklaracijom *Strategy* interfejsa;
* izdvajanjem algoritama u njihove klase;
* dodavanjem reference na objekat strategije i setera za izmjenu tog polja.

Korištenje:

Strategy se može koristiti za odabir algoritma čiji je cilj izdvajanje statistike za korisnika u ovisnosti od njegove uloge.

***State***

Kod State dizajn paterna, objekat mijenja način ponašanja na osnovu trenutnog stanja. Implementacija ovog paterna bi, uzimajući u obzir kompleksnost našeg sistema, donijela više komplikacija nego koristi. Međutim, možemo posmatrati prisustvo i odsustvo učenika času kao dva stanja u okviru kojih učenik može, odnosno ne može biti ocijenjen.

***Template Method***

Template Method omogućava izdvajanje određenih koraka algoritma u odvojene podklase. Na primjer, definisanje osnovnog algoritma za generisanje izvještaja o ocjenama ili izostancima, uz mogućnost da podklase implementiraju specifične dijelove algoritma. Izvještaj se može sastojati od: ukupnog prosjeka ocjena jednog učenika, prosjeka po predmetima, prijedloga ocjena po predmetima, ukupnog broja izostanaka, broja opravdanih i neopravdanih izostanaka, te broja izostanaka po predmetima.

To možemo implementirati:

* kreiranjem apstraktne bazne klase;
* implementiranjem metode koja piše izvještaj na osnovu izdvojenih koraka;
* definiranjem skupa apstraktnih metoda koje predstavljaju korake algoritma.

Korištenje:

Na primjer, za potrebe održavanja roditeljskog sastanka, razrednik jednostavnim zahtjevom može pokrenuti generisanje izvještaja za svakog učenika u okviru razreda za koji je zadužen. Izvještaje može predati roditeljima u printanoj formi ili poslati putem mail-a.

***Observer***

Uloga Observer dizajn paterna je da uspostavi relaciju između objekata, tako da kada jedan objekat promijeni stanje, drugi zainteresovani objekti se obavještavaju. U okviru našeg sistema, akcija ocjenjivanja učenika može dovesti do potrebe za obavještavanjem razrednika. U ovisnosti od dinamike rada razrednika, on može odlučiti da li želi primati notifikacije ovog tipa, što Observer i omogućava. Ipak, kako ova mogućnost ne spada među primarne funkcionalnosti našeg sistema, njena implementacija bi mogla dovesti do bespotrebnih komplikacija. Međutim, u slučaju proširivanja sistema, ovakav pristup bi mogao biti veoma zanimljiv.

***Iterator***

Iterator dizajn patern omogućava pristup elementima kolekcije bez poznavanja kako je kolekcija strukturirana. U kontekstu našeg sistema, možemo posmatrati više grupacija objekata, na primjer, učenika, nastavnika ili razrednika, kao i kolekcija kao što su ocjene, izostanci ili održani časovi. U zavisnosti od uloge prijavljenog korisnika, potrebno je prikazati odgovarajuće podatke – dakle, prolaziti kroz dostupne kolekcije.

To možemo implementirati:

* deklaracijom *Iterator* interfejsa;
* deklaracijom *Collection* interfejsa;
* implementacijom konkretne klase iteratora za željenu kolekciju;
* implementacijom metode za dohvaćanje iteratora u konkretnoj kolekciji.

Korištenje:

Unutar sistema možemo koristiti Iterator kako bi prolazili kroz kolekciju ocjena tačno određenog učenika ili kolekciju njegovih izostanaka, kao i za prikaz učenika tačno određenog razreda.

***Command***

Command dizajn patern se koristi za enkapsuliranje svih informacija potrebnih za odgođeno izvođenje akcije ili pokretanje događaja. Ovaj dizajn patern, iako koristan, u naš sistem potencijalno donosi više pometnje nego koristi. Ipak, možemo posmatrati akciju dodavanja učenika od strane administratora, koja se može relizirati u više etapa. *Command* interfejs definira zajednički interfejs za sve komande i sadrži metodu *execute()*. *AddStudentCommand* (konkretni *Command*) implementira navedeni interfejs i enkapsulira podatke o učeniku. *Receiver* klasa obavlja stvarni posao dodavanja učenika u sistem, te ima metodu *addStudent()*. *Invoker* poziva komande, tj. inicira akcije. Za kraj, klijent kreira konkretne komande i prosljeđuje ih invokeru za izvršavanje.