STRUKTURALNI PATTERNI

- Adapter pattern – svrha mu je da omoguci siru upotrebu postojecih klasa. Recimo da nam je potreban drugaciji interfejs postojece klase, ukoliko ne zelimo mijenjati njen kod koristimo adapter pattern.

Klasa Notifikacija može komunicirati s eksternim servisom (npr. Email). Ako je interfejs eksternog servisa drugačiji, koristi se adapter (NotifikacijaAdapter) koji implementira interfejs sa odredjenim metodama.

- Decorator pattern – omogucava dinamicko prosirivanje funkcionalnosti objekata bez izmjene izvornog koda. Koristenjem dekoratorskih klasa koje sadrze dodatne funkcionalnosti smanjujemo potrebu za kreiranjem velikog broja podklasa.

Ako zelimo omogućiti dodatne informacije uz Odgovor (npr. status "označen kao tačan", "ocijenjen"), koristili bismo dekoratore: OdgovorSaStatusom, OdgovorSaOcjenom, itd., koji proširuju osnovni prikaz odgovora.

- Façade pattern – koristi se kako bi se korisnicima olaksalo koristenje slozenijih Sistema. Recimo ukoliko ne zelimo da korisnik poznaje unutrasnju logiku Sistema, tvorimo jednostavniji interfejs preko kojeg korisnik moze komunicirati sa sistemom.

U ovom slucaju façade pattern mozemo iskoristiti kod situacija gdje izvrsavanje zeljene akcije zahtjeva vise pojedinacnih metoda. Za taj skup metoda mozemo napraviti jedinstvene metode koje sve te korake odjednom obavljaju.

- Bridge pattern – on sluzi za odvajanje apstrakcije od njene implementacije, I tako klasa mozemo istovremeno podrzati vise implementacija I apstrakcija.

Prikaz korisnika (student, profesor, admin) može imati različite UI elemente i funkcije. Definiše se apstrakcija IKorisnikView, a različite implementacije (npr. StudentView, AdminView) prikazuju iste podatke na različite načine.

- Proxy pattern – on ustvari upravlja pristupom stvarnim objektima, tj. on omogucava kontrolu pristupa. Preko proxy patterna definisemo pravila preko kojih se objekti aktiviraju.

Trenutno Administrator i Korisnik direktno koriste metode Controller klasa bez ikakve kontrole pristupa. Svaki korisnik teoretski može direktno pozvati funkcije koje nisu namijenjene njemu.

Rješenje:
Uvođenje interfejsa IAkcija sa metodom izvrsi(). Kreira se RealnaAkcija klasa koja sadrži konkretnu logiku (npr. dodajPitanje, obrisiOdgovor). ProxyAkcija implementira isti interfejs, ali prije prosljeđivanja poziva RealnaAkcija, provjerava da li korisnik ima pravo pristupa (npr. if korisnik.isAdmin()).

Prednosti:

- **Sigurnost**: Neovlaštenim korisnicima je onemogućeno izvršavanje nedozvoljenih operacija(Profesor ne moze postaviti pitanja I sl.).
- **Transparentnost**: Klijentski kod koristi isti interfejs bez znanja o stvarnoj implementaciji.
- **Ekspanzibilnost**: U budućnosti se mogu lako dodati logovi, keširanje ili praćenje aktivnosti.

- Composite pattern – koristimo ga za kreiranje hijearhijskih struktura u obliku stable. Pattern omogucava da razlicite klase sa drugacijiom implementacijom istih metoda mogu biti objedinjene kroz zajednicki interfejs.

Problem:
Komentari i podkomentari (npr. odgovor na komentar) se tretiraju kao odvojeni entiteti bez hijerarhijske veze. Trenutno ne postoji elegantan način za prikazovanje komentara ugniježđeno.

Rješenje:
Kreira se interfejs IKomentarElement sa metodama poput prikazi() i dodajKomentar(). Klasa Komentar implementira ovaj interfejs. Nova klasa GlavniKomentar takođe implementira IKomentarElement, ali sadrži listu IKomentarElement objekata – čime omogućava hijerarhijski prikaz komentara i podkomentara.

Prednosti:

- Mogućnost rekurzivnog prikaza i obrade komentara
- Olakšano dodavanje ugniježđenih odgovora
- Pojednostavljen kod View-a