STRUKTURALNI PATERNI

Strukturalni paterni se bave kompozicijom klasa i objekata, omogućavajući kreiranje većih struktura kroz kombinovanje postojećih komponenti. Oni rješavaju probleme vezane za načine na koje se objekti kombinuju i kako se formiraju složenije cjeline, pri čemu se čuva fleksibilnost i efikasnost sistema.

---

1. ADAPTER PATTERN

Definicija: Adapter pattern je strukturalni pattern koji omogućava saradnju između klasa sa nekompatibilnim interfejsima. On "prilagođava" interfejs jedne klase tako da odgovara interfejsu koji klijent očekuje, bez mijenjanja originalnog koda.

Ključne karakteristike:

- **Omogućava postojećim klasama da rade zajedno bez modifikacije njihovog koda**
- **Rješava probleme nekompatibilnosti interfejsa**
- **Može se implementirati kroz nasljeđivanje (class adapter) ili kompoziciju (object adapter)**
- **Često se koristi za integraciju sa third-party bibliotekama ili legacy sistemima**

Struktura:

- **Target – interfejs koji klijent očekuje**
- **Adaptee – postojeća klasa sa nekompatibilnim interfejsom**
- **Adapter – klasa koja implementira Target interfejs i koristi Adaptee**

Svrha mu je da omogući širu upotrebu postojećih klasa.
Primjer: **Klasa Notifikacija može komunicirati s eksternim servisom (npr. Email). Ako je interfejs eksternog servisa drugačiji, koristi se adapter (NotifikacijaAdapter) koji implementira interfejs sa određenim metodama.**

---

2. DECORATOR PATTERN

Definicija: Decorator pattern omogućava dinamičko dodavanje novih funkcionalnosti objektima bez mijenjanja njihove strukture. Pattern kreira wrapper klase koje "ukrašavaju" originalne objekte dodatnim ponašanjem.

Ključne karakteristike:

- **Alternativa nasljeđivanju za dodavanje funkcionalnosti**
- **Kompozicija umjesto nasljeđivanja**
- **Mogućnost kombinovanja više dekoratora**
- **Čuva original interfejs objekta**
- **Podržava principe Open/Closed i Single Responsibility**

Struktura:

- **Component – osnovni interfejs**
- **ConcreteComponent – klasa koja implementira osnovnu funkcionalnost**
- **BaseDecorator – wrapper klasa koja sadrži referencu na Component**
- **ConcreteDecorator – konkretni dekoratori koji dodaju specifične funkcionalnosti**

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

---

3. FACADE PATTERN

Definicija: Facade pattern pruža pojednostavljeni interfejs za složen podsistem. On "sakriva" složenost sistema iza jednostavnog interfejsa, omogućavajući klijentima lakše korišćenje bez potrebe za poznavanjem unutrašnje logike.

Ključne karakteristike:

- **Pojednostavljuje upotrebu složenih sistema**
- **Smanjuje broj zavisnosti između klijenta i podsistema**
- **Ne sakriva funkcionalnost – klijenti i dalje mogu direktno pristupiti podsistemu**
- **Čini kod čitljivijim i lakšim za održavanje**
- **Često se koristi u arhitekturnim slojevima**

Struktura:

- **Facade – glavna klasa koja pruža jednostavan interfejs**

- **Subsystem classes – klase koje implementiraju složenu logiku**
- **Client – koristi Facade umjesto direktnog pristupa podsistemu**

Primjer: **Ako izvršavanje akcije zahtijeva više pojedinačnih metoda, možemo napraviti jednu metodu u Facade klasi koja sve te korake obavlja zajedno.**

---

4. BRIDGE PATTERN

Definicija: Bridge pattern odvaja apstrakciju od njene implementacije tako da se mogu nezavisno mijenjati. Pattern "povezuje" apstrakciju sa implementacijom kroz kompoziciju umjesto nasljeđivanja.

Ključne karakteristike:

- **Omogućava nezavisno mijenjanje apstrakcije i implementacije**
- **Smanjuje broj klasa u hijerarhiji nasljeđivanja**
- **Podržava runtime zamjenu implementacije**
- **Sakriva detalje implementacije od klijenata**
- **Poboljšava proširivost sistema**

Struktura:

- **Abstraction – definiše interfejs apstrakcije**
- **RefinedAbstraction – proširuje apstrakciju**
- **Implementor – interfejs za implementacione klase**
- **ConcreteImplementor – konkretne implementacije**

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

---

5. PROXY PATTERN

Definicija: Proxy pattern pruža zamjenu ili predstavnika za drugi objekat kako bi kontrolisao pristup tom objektu. Proxy objekat ima isti interfejs kao originalni objekat, ali može dodati dodatnu logiku prije ili poslije prosleđivanja zahtjeva.

Ključne karakteristike:

- **Kontrola pristupa originalnom objektu**
- **Lazy initialization – kreiranje objekta tek kada je potreban**
- **Keširanje rezultata**
- **Logovanje i monitoring**
- **Bezbjednosne provjere**

Tipovi Proxy pattern-a:

- **Virtual Proxy – odlaže kreiranje "skupih" objekata**
- **Protection Proxy – kontroliše pristup na osnovu dozvola**
- **Remote Proxy – predstavlja objekat u drugom adresnom prostoru**
- **Smart Proxy – dodaje dodatnu funkcionalnost (reference counting, locking)**

Struktura:

- **Subject – zajednički interfejs za RealSubject i Proxy**
- **RealSubject – stvarni objekat koji Proxy predstavlja**
- **Proxy – sadrži referencu na RealSubject i kontroliše pristup**

Primjer: **Trenutno Administrator i Korisnik direktno koriste metode Controller klasa bez ikakve kontrole pristupa. Svaki korisnik teoretski može direktno pozvati funkcije koje nisu namijenjene njemu.**
Rešenje: Uvođenje interfejsa IAkcija sa metodom izvrši(). Kreira se RealnaAkcija koja sadrži konkretnu logiku (npr. dodajPitanje, obrišiOdgovor). ProxyAkcija implementira isti interfejs, ali prije poziva RealnaAkcija provjerava da li korisnik ima pravo pristupa (npr. if korisnik.isAdmin()).

Prednosti:

- **Sigurnost: Neovlašćenim korisnicima je onemogućeno izvršavanje nedozvoljenih operacija (npr. Profesor ne može postavljati pitanja)**
- **Transparentnost: Klijentski kod koristi isti interfejs bez znanja o stvarnoj implementaciji**
- **Ekspanzibilnost: Moguće dodati logovanje, keširanje, praćenje aktivnosti**

---

6. COMPOSITE PATTERN

Definicija: Composite pattern omogućava tretiranje pojedinačnih objekata i kompozicija objekata na uniforman način. Pattern omogućava kreiranje hijerarhijskih struktura u obliku stabla, gdje se objekti organizuju u tree-like strukturu.

Ključne karakteristike:

- **Uniformno tretiranje leaf i composite objekata**
- **Rekurzivna struktura koja omogućava hijerarhije proizvoljne dubine**
- **Pojednostavljuje klijentski kod – isti interfejs za sve objekte**
- **Olakšava dodavanje novih tipova komponenti**
- **Implementira part-whole hijerarhije**

Struktura:

- **Component – zajednički interfejs za sve objekte u kompoziciji**
- **Leaf – predstavlja krajnje objekte u hijerarhiji (nema djece)**
- **Composite – definiše ponašanje komponenti koje imaju djecu i čuva child komponente**

Primjer: **Komentari i podkomentari (npr. odgovor na komentar) se trenutno tretiraju kao odvojeni entiteti bez hijerarhijske veze.**
Rešenje: Kreira se interfejs IKomentarElement sa metodama poput prikazi() i dodajKomentar(). Klasa Komentar implementira ovaj interfejs. Nova klasa GlavniKomentar također implementira IKomentarElement, ali sadrži listu IKomentarElement objekata – omogućavajući hijerarhijski prikaz komentara i podkomentara.

Prednosti:

- **Mogućnost rekurzivnog prikaza i obrade komentara**
- **Olakšano dodavanje ugniježdenih odgovora**

- **Pojednostavljen kod View-a**

ZAKLJUČAK

Strukturalni paterni predstavljaju važan dio dizajna softvera jer omogućavaju kreiranje fleksibilnih i skalabilnih sistema kroz elegantno komponovanje objekata i klasa. Svaki od ovih patterna rješava specifične probleme kompozicije i omogućava lakše održavanje i proširivanje koda bez narušavanja postojeće funkcionalnosti.