STRUKTURALNI PATERNI

Strukturalne paterne čine: Facade, Decorator, Adapter, Bridge, Composite i Proxy.

Strukturalni paterni se bave kompozicijom i predstavljaju načine za definisanje odnosa među objektima. Strukturalni paterni daju mogućnost da kada je potrebna promjena u jednom dijelu sistema da se ne mora ostatak sistema mijenjati.

**Adapter pattern**

Osnovna namjena Adapter pattern-a je da omogući širu upotrebu već postojećih klasa. Kada je potreban drugačiji interfejs postojeće klase, a ne želimo da mijenjamo postojeću klasu koristimo Adapter pattern. Tada se kreira nova Adapter klasa koja se koristi kao posrednik između originalne klase i interfejsa.

Primjena u sistemu:

Ako korisnik želi da koristi pretraživanje na osnovu jedne ključne riječi, ponuđene nekretnine će se filtrirati po zahtjevu korisnika. Adapter pattern bi mogli iskoristiti na način da napravimo novi interfejs za pretragu i napraviti klasu Adapter koje će implemetirati taj interfejs i pozivati metodu za pretraživanje po ključnoj riječi.

**Decorator pattern**

Osnovna namjena Decorator pattern-a je da omogući dinamičko dodavanje novih elemenata i funkcionalnosti postojećim objektima. Objekat pri tome ne zna da je urađena dekoracija što je veoma korisno za ponovnu upotrebu komponenti softverskog sistema.

**Proxy pattern**

Proxy patern služi za dodatno osiguravanje objekata od pogrešne ili zlonamjerne upotrebe. Primjenom ovog paterna omogućava se kontrola pristupa objektima, te se onemogućava manipulacija objektima ukoliko neki uslov nije ispunjen, odnosno ukoliko korisnik nema prava pristupa traženom objektu.

Primjena u sistemu:

Proxy pattern u aplikaciji možemo iskoristi npr. kod u unošenja password-a prilikom login-a korisnika. Tada ćemo imati autentifikacijski proxy koji će omogućiti ili onemogućiti korisniku pristup mogućnostima registrovanog korisnika.

Proxy pattern možemo još iskoristiti i na način da kreiramo Proxy klasu koja će izvršiti kreiranje instance klase Admin, tj. klasa Admin treba biti privatna. Metoda za autentifikaciju će se pozvati samo kada su uneseni ispravni korisnički podaci.

**Composite pattern**

Composite pattern ima namjenu da omogući formiranje strukture stabla pomoću klasa u kojoj se individualni objekti i kompozicije individualni objekata jednako tretiraju. Koristi se za kreiranje hijerahije objekata, tj. Kada svi objeti imaju različite implementacije nekih metoda kojima je potrebno pristupati na isti način.

Primjena u sistemu:

Da bi implemetirali Composite pattern cijeli sistem bi trebali predstaviti kao stablo. Composite pattern bi mogli iskoristiti na način da napravimo IComposite interfejs koji će imati metodu azuriraj() koja bi služila za ažuriranje podataka u našim klasama.

**Facade pattern**

Facade pattern služi kako bi se klijentima pojednostavilo korištenje kompleksnih sistema. Klijenti vide samo krajnji izgled objekta, dok je njegova unutrašnja struktrura skrivena. Na ovaj način se smanjuje mogućnost pojavljivanja grešaka jer klijenti ne moraju dobro poznavati sistem da ga koriste.

**Flyweight pattern**

Flyweight patern se koristi kako bi se onemogućilo bespotrebno stvaranje velikog broja instanci objekata koji svi u suštini predstavljaju jedan objekat. Samo ukoliko postoji potreba za kreiranjem specifičnog objekta sa jedinstvenim karakteristikama, vrši se njegova instantacija, dok se u svim ostalim slučajevima koristi postojeća opća instance objekta.