Paterni ponašanja

1. Strategy patern

Strategy algoritam izdvaja algoritme iz matične klase i uključuje ga u posebne klase. U našem sistemu ovaj patern ćemo primjeniti tako što ćemo Administratoru dati mogućnost da sortira Usluge po cijeni usluge, nazivu usluge ili po korisniku usluge. Da bi ovo realizirali napravit ćemo interfejs ISortiraj koji će imati jednu metodu sortiraj(). Ovaj interfejs implementirat će tri nove klase SortirajCijena, SortirajUsluga i SortirajKorisnik. U klasi Administrator dodat ćemo atribut tipa klase ISortiraj pod nazivom sortiraj i metodu sortirajPo(String uslov) u kojoj će se na osnovu stringa koji se pošalje odlučivati koju vrstu sortiranja ćemo izvršavati. Ovaj patern nam pomaže da smanjimo dužinu koda funkcije, ali i omogućava nam da uz minimalne prepravke dodamo neki novi način sortiranja ukoliko se to bude tražilo od nas.

2. State patern

State patern je dinamička verzija strategy paterna koji omogućava objektu da mijenja način ponašanja na osnovu trenutnog stanja. Želim da osiguramo da naš korisnik može biti označen kao dužnik i kao redovni platiša i da se ta stanja mogu mijenjati obzirom na to da li korisnik ima dugova ili nema. Da bi ovo osigurali na elegantan način koristit ćemo state patern. Kreirat ćemo interfejs IDugovi koji će imati metodu postaviStanje. Ovaj interfejs implementirat će klase Duzan i Platisa. U klasi Korisnik dodat ćemo atribut stanje tipa IDugovi i metodu provjeriDugovanja na osnovu koje će se koristiti interfejs i klase koje smo kreirali da bismo dobili kakvo je stanje našeg korisnika. Tačnije ako je dug korisnika veći od nula atribut stanje će biti tipa Duzan, a ako su dug jednak nula atribut stanje će biti tipa Platisa. Ovi atributi će mijenjati svoj tip u ovisnosti od stanja dug što znači da ćemo dinamički vršit promjenu stanja objekta.

3. TemplateMethod patern

TemplateMethod patern omogućava izdvajanje određenih koraka algoritma u odvojene podklase tako da se struktura algoritma ne mijenja. Ovaj patern nismo iskoristili u našem sistemu pa ćemo ga hipotetički analizirati. Želimo omogućiti da na osnovu spola osobe odredimo način sahrane. Napravit ćemo interfejs ISahrana koja će imati metodu sahrani() tu metodu će implementirati dvije klase MuskaSahrana i ZenskaSahrana gdje će ova metoda biti realizirana na različit način. U klasu Sahrana dodajemo metodu SahraniOsobu(ISahrana s) u kojoj ćemo izvršiti pozivanje metode sahrani() one klase čija nam je instanca poslana kao parametar. Ovim smo postigli da smo izdvojili akciju koja bi se inače morala obavljati u jednoj klasi i bila komplikovana te imala puno grananja u posebne klase koje će se ovisno o parametru koji se proslijedi koristiti i olakšati i pojednostaviti rad.

4. Observer patern

Observer patern uspostavlja relaciju između objekata tako kada jedan objekat promijeni stanje drugi „zainteresirani“ objekti se obavještavaju. Ovaj patern nismo primjenili u našem sistemu pa ćemo ga pokušati hipotetički razmotriti. Želim omogućiti da se pri promjeni cijene usluge obavijeste korisnici. U klasu Usluge dodat ćemo metodu obavjestenje(List<Korisnik>) koja će primati listu korisnika kojima će se u slučaju promjene cijene usluge slati obavještenje. Dodatno ćemo kreirati i interfejs IKorisnik koji će imati jednu metodu update() koja će biti implementirana u klasi Korisnik i izvršavati promjenu cijena usluga koje korisnik ima nakon što se izvrši promjena cijene.

5. Iterator patern

Iterator patern omogućava sekvencijalni pristup elementima kolekcije bez poznavanja kako je kolekcija sruktuirana. Ovaj patern nismo primjenili u našem sistemu pa ćemo ga razmotriti hupotetički. Zamislimo da želimo proći kroz sve zaposlenike pogrebnog društva, ali da nemamo interfejs IZaposlenici i kolekciju tih instanci sačuvanih u listi u klasi VlasnikSingleton.

Napravili bismo klasu Zaposlenici što bi simuliralo klasu Collection i u njoj bi čuvali sve zaposlenike u pogrebnom društvu. Interfejs IZaposleni simulirao bi interfejs IEnumerable i u sebi bi imao metodu getZaposleni koju implementirala klasa Zaposleni, a ona bi simulirala rad metode getEnumerator() tj. pružala bi vrijednosti kolekcije u sekvenci. U klasi VlasnikSingletone čuvali bismo kolekciju npr. listu zaposlenih i u njoj bismo mogli kroz neku metodu ili funkciju unutar klase prolaziti foreach petljom i obavljati neku akciju koju želimo.