Kreacijski paterni

1. Singleton patern

Uloga Singleton paterna je da osigura da se klasa može instancirati samo jednom i da osigura globalni pristup kreiranoj instanci klase. Postoji više objekata koje je potrebno samo jednom instancirati i nad kojim je potrebna jedinstvena kontrola pristupa. Singleton patern nam osigurava da neka klasa ima samo jednu instancu, a dozvoljava globalni pristup (sve klase mogu pristupiti) ovoj instanci.

U našem sistemu, bismo ovaj patern mogli iskoristiti kada bismo napravili klasu statistika koja bi prikazivala koliko je objavljenih, a koliko prodanih artikala. Ovu klasu bismo koristili kao ”globalni resurs” aplikacije, te ostale klase mogu koristiti njene atribute i ažurirati ih.

U klasi Statistika dodali bismo privatan konstruktor - ovim bismo zabranili drugim klasama da stvaraju nove instance ove klase. Zatim bismo dodali statičku metodu getInstance - ova metoda ”glumi” konstruktor, ona uvijek vraća isti objekat, a ne pravi novi (osim kada se prvi put pravi instanca ove klase). Ovim smo osigurali da se ne pravi nova instanca kada neka druga klasa želi pristupiti klasi Statistika.

1. Prototype patern (primjenjeno)

Prototip patern nam omogućava da kopiramo postojeće objekte bez da ovisimo o njihovim klasama. Npr, ukoliko želimo kreirati identičnu kopiju nekog objekta. Prvo bismo morali kreirati novi objekat iste klase. Zatim moramo preći kroz sve njegove atribute i kopirati njihove vrijednosti u novi objekat.

Medutim, tu može postojati problem - ne mogu se svi objekti kopirati na ovaj način, jer neki atributi su privatni te nisu vidljivi izvan tog samog objekta. Još jedan mogući problem je što kada trebamo kreirati duplikat nekog objekta, moramo znati klasu tog objekta, te naš kod postaje ovisan o toj klasi.

Iz ova dva razloga koristimo Prototip patern. Koristeći ovaj patern, pravimo jedan interfejs za sve objekte koji podržavaju kloniranje. Ovaj interfejs se obično sadrži od metode clone(). Implementacija ove metode je slična u svim klasama - kreira se objekat trenutne klase i prenesu se vrijednosti svih atributa u novi objekat. Na ovaj način, kada nam treba isti objekat kao objekat koji smo konfigurisali, pozovemo metodu clone() umjesto da iz početka stvaramo novi objekat.

U našem sistemu, primjena ovog patern-a bi se mogla primijeniti nad klasom Artikal. Ako imamo više artikala iz iste kategorije lakše je klonirati jedan artikal, a zatim mijenjati neke podatke (kao što su naziv, cijena, opis i sl.).

1. Factory Method pattern

Uloga Factory Method paterna je da omogući kreiranje objekata na način da podklase odluče koju klasu instancirati. Sistem je moguće proširiti korisničkim zahtjevom da razlikujemo artikle koje nudimo prema kategorijama(memorija, procesor, periferija). Kako bi instanciranje različitih klasa artikala bilo pojednostavljeno kreirali bi Factory Method pattern.

Trebali bismo kreirati više klasa, koje bi implementirale interfejs IArtikal. Također potrebna je klasa Kreator koja ima metodu Fabrika koja sadrži logiku koja determinira koja će se klasa instancirati. Konkretna usluga bi mogla biti u sklopu Kreator klase, pristup proizvođaču pojedinacnih artikala, dajProizvođača() koja vraća string.

1. Abstract Factory pattern

Abstract Factory patern omogućava da se kreiraju familije povezanih objekata/produkata. Na osnovu apstraktne familije produkata kreiraju se konkretne fabrike (factories) produkata različitih tipova i različitih kombinacija. Patern odvaja definiciju (klase) produkata od klijenta.

U naš sistem moguće je implementirati ovaj patern tako što u klasi Artikal zamijenimo tip komentara, sa vlastitim tipovima, odnosno klasama. Ove klase će biti apstraktne i zvat će se PozitivniKomentari, NegativniKomentari, NeutralniKomentari. Nakon toga, ukoliko imamo da pored ostavljanja komentara kupci ostavljaju ocjene 1-5, iz apstraktne klase PozitivniKomentari naslijedimo klase PozitivniKomentariDobar, PozitivniKomentariVrlodobar. Na isti način naslijedimo klase NegativniKomentariLoše, NegativniKomentariJakoLoše iz klase NegativniKomentari. Sada ćemo napraviti interfejs IKomentariFactory koji će sadržavati metode getPozitivniKomentari i getNegativniKomentari koje vraćaju instance klasa PozitivniKomentari I NegativniKomentari respektivno. Sljedeći korak je da kreiramo klase PozitivniKomentariFactory, NegativniKomentariFactory, koje će implementirati interfejs IKomentariFactory i njegove metode. Na kraju klasa Artikal, koja sada sadrži dva atributa (jedan tipa PozitivniKomentari i jedan tipa NegativniKomentari ), će u konstruktoru primati instancu odgovarajuće factory klase, te na taj način može da instancira svoje privatne atribute pozitivnih I negativnih komentara. Ovim ćemo osigurati da neće doći do miješanja nekompatibilnih tipova pozitivnih I negativnih komentara. Takoder, pored ovoga, možemo naravno i imati neke metode koje postavljaju ili izmjenjuju komentare, a pri tome koriste factory klasu.

1. Builder pattern (primjenjeno)

Uloga Builder paterna je odvajanje specifikacije kompleksnih objekata od njihove stvarne konstrukcije. Isti konstrukcijski proces može kreirati različite reprezentacije.

U našem sistemu ovaj patern smo realizirali tako što smo dodali interfejs Builder sa metodama dajArtikalUOpsegu I dajArtikalUKategoriji. Ove metode vraćaju liste artikala, te uzimaju u obzir faktore kao što su cijena I kategorija.