**Paterni ponašanja**

1. Strategy patern (primjenjeno)

Strategy patern izdvaja algoritam iz matične klase i uključuje ga u posebne klase, te kako bi se omogućila brza i jednostavna promjena implementacije koja se želi koristiti u bilo kojem trenutku. Na ovaj način omogućava se jednostavno brisanje ili dodavanje novog algoritma koji se može koristiti po želji. Podržava O princip. Na ovaj način vrši se refaktoring postojećeg dijagrama klase kako bi se dobio novi dijagram u kojem je ispoštovan strategy patern.

Sada je za dodavanje novog algoritma (sortiranja) dovoljno dodati novu klasu koja će naslijediti interfejs ISortiranje, a u postojećim klasama ne treba praviti nikakve izmjene.

U našem sistemu, možemo primjeniti ovaj paten u klasi Prodavač. Ova klasa sadrži atribut ObjavljeniArtikli koji je tipa lista Artikala. Mi želimo da omogućimo prodavaču sortiranje ovih artikala po različitim kriterijima (po imenu, po datumu objavljivanja, po cijeni, po stanju na zalihama itd.) kako bi mu se prvo prikazali oni artikli, koji su mu u tom momentu najpotrebniji.

Ovo ćemo uraditi na sljedeći način: Prvo ćemo kreirati interfejs ISortiranje, koji će sadržavati metodu sortiraj, koja kao parametar prima listu artikala. Nakon toga ćemo kreirati klase NazivSort, DatumuObjaveSort, CijenaSort, itd. Svaka od ovih klasa će implementirati interfejs ISortiranje, odnosno njegovu metodu sortiraj (), na odgovarajući način. Zatim ćemo u klasi Prodavač kreirati atribut strategija, koji je tipa ISortiranje. Nakon toga u klasu Prodavač ćemo dodati i metodu sortirajArtikle() koja će samo pozivati metodu sortiraj () atributa strategija i na taj način sortirati listu artikala. Trebamo još da implementiramo metodu promijeniStrategiju() koja prima ISortiranje objekat i mijenja atribut strategija. Na ovaj način smo postigli da prodavač ima opciju da promijeni redosljed artikala koji mu se prikazuju na način koji mu u tom momentu najviše odgovara. Takoder, ako nekada bude potrebno, veoma lako bi bilo dodati novu strategiju sortiranja.

1. State patern

State patern omogućava objektu da mijenja svoja stanja, od kojih zavisi njegovo ponašanje. Sa promjenom stanja objekat se počinje ponašati kao da je promijenio klasu. Stanja se ne mijenjaju po želji klijenta, već automatski, kada se za to steknu uslovi. State Pattern je dinamička verzija Strategy paterna. Postiže se promjenom podklase unutar hijerarhije klasa.Podržava O princip.

Ovaj patern u našem sistemu možemo primijeniti na sljedeći način: Ukoliko bismo izbrisali enum Status, kada kupac završi kupovinu, pošiljka će se nalaziti u četiri moguća stanja: Kupljeno, Spremno, Poslano i Isporučeno. U našem sistemu ćemo dodati interfejs IStanjePošiljke te klase KupljeniArtikal, SpremanArtikal, PoslanArtikal, IsporučenArtikal koje implementiraju ovaj interfejs. U klasi Notifikacija ćemo zamijeniti atribut StatusPošiljke (tipa Status) sa atributom stanjePošiljke (tipa IStanjePošiljke). Ukoliko je artikal kupljen ili spreman mogli bismo uvesti novu mogućnost da kupac bira koja brza pošta će izvršiti isporuku. Ukoliko je poslan artikal kupac ne bi imao tu mogućnost, a ukoliko je isporučen tek tada bi imao mogućnost reviewa i npr.ocjenjivanja kurirske službe.

1. Template Method patern

Template method patern služi za omogućavanje izmjene ponašanja u jednom ili više dijelova. Najčešće se primjenjuje kada se za neki kompleksni algoritam uvijek trebaju izvršiti isti koraci, ali pojedinačne korake moguće je izvršiti na različite načine.

U našem sistemu bismo ovo mogli realizirati na način da npr. Klasu Kupac proširimo na posebne vrste kupaca („obični“ i „privilegovani“). U tom slučaju bi kupac prilikom plaćanja narudžbe imao određeni popust i imao dodatne pogodnosti prilikom dopune ETF kredita.

1. Observer patern (primjenjeno)

Uloga Observer paterna je da uspostavi relaciju između objekata tako da kada jedan objekat promijeni stanje, drugi zainteresirani objekti se obavještavaju.

Ovaj patern ćemo implementirati u naš sistem. Kupci imaju mogućnost da spase bilo koji artikal. U klasi Artikal imamo atribut PotencijalniKupci koji je Lista Kupaca. Ukoliko dođe do toga da količina artikla bude manja od broja potencijalnih kupaca, svi oni će dobiti obavijest o tome da se približava istek zaliha.

1. Iterator patern

Iterator patern omogućava sekvencijalni pristup elementima kolekcije bez poznavanja kako je ta kolekcija struktuirana.

U sistemu imamo liste kupaca, artikala, komentara itd. i sve vrste elemenata navedenih listi su kompleksni objekti tj. nisu ugrađeni tipovi u C#. Napravit ćemo jednu vrstu iteratora koji će imati interfejs koji će definisati metode getNext(), hasMore(), count() (broj elemenata u listi), current() (vraća trenutni objekat na kojem smo pozicionirani), rewind() (vraća iterator na prvu poziciju) i svaki kontroler koji ima neku interakciju sa navedenim listama će po potrebi koristiti taj iterator. Također trebamo napraviti ConcreteIterator koji će u sebi sadržavati kolekciju koja će se inicijalizirati tokom kreiranja konkretnog iteratora. ArtikalController će koristiti iterator i ubaciti listu artikala, KupacController će inicijalizirati iterator i ubaciti u njega listu kupaca. Ako želimo iterator za artikle, napišemo npr. Iterator iterator = new Iterator(listaArtikala). Treba naglasiti da opisani iterator ima najviše smisla kada sve klase koje mogu biti elementi liste implementiraju isti interfejs tj. da kupac, artikal, komentar itd. imaju isti interfejs.

1. Chain of Responsibility pattern

Chain of responsibility patern namijenjen je kako bi se jedan kompleksni proces obrade razdvojio na način da više objekata na različite načine procesiraju primljene podatke.

Ukoliko se sistem proširi zahtjevom da se svakom kupcu prilikom izbora artikla omogući najoptimalniji izbor artikla po kriterijima (cijena, rejting). Kupcu će biti omogućen izbor prioriteta za ove kriterije. Za ovo je potrebno je pratiti Chain of responsibility pattern.

Kreirat ćemo interfejs IUpravljač s metodom proslijedi() koja će slati zahtjeve narednoj klasi po redu za obradu po lancu koji je definiran prioritetom kupca. Ovaj interfejs nasljeđuje klasu ArtikalFilter. Ona ima atribute artikli (lista potencijalnih artikala) i bazniFilter koji predstavlja trenutni objekat u lancu koji je zadužen za vršenje obrade nad objektom artikal. Dodati nove klase obrađivača FiltrirajCijenu, FiltrirajRejting i sl., koje će naslijediti klasu ArtikalFilter te koje će posjedovati vlastite metode za vršenje različitih filtriranja liste artikala na osnovu datih informacija. One će implementirati virtuelnu metodu obradi na način da vrše pozivanje onih dijelova obrade za koje su zaduženi.

1. Mediator pattern

Mediator pattern namijenjen je za smanjenje broja veza između objekata. Umjesto direktnog međusobnog povezivanja velikog broja objekata, objekti se povezuju sa međuobjektom medijatorom, koji je zadužen za njihovu komunikaciju. Kada neki objekt želi poslati poruku drugom objektu, on šalje poruku medijatoru, a medijator prosljeđuje tu poruku namijenjenom objektu ukoliko je isto dozvoljeno.

Mediator pattern možemo implementirati na način da ubacimo nove vrste korisnika. Korisnici će se sada dijeliti na: NeregistrovaniKorisnik, RegistrovaniKorisnik, PremiumKorisnik. Sve navedene vrste korisnika imaju različitu implementaciju. Dodat ćemo i funkcionalnost chat rooms gdje jedan od premium korisnika ili administratora može kreirati chat room u koji se svaka od vrsta korisnika mogu pridružiti i komunicirati. Sve poruke se šalju ChatRoom klasi koja će imati metodu showMessage, koja će primati pošiljaoca i poruku koju šalje. Administratori su zaduženi za brisanje chat soba. Chat room je u ovom primjeru Mediator klasa koja će imati svrhu komunikacijskog posrednika između dva ili više korisnika. Na ovaj način svaki do korisnika može slati svoje poruke I vidjeti poruke drugih korisnika , a uz to ne mora biti upoznat sa implementacijom drugih korisnika.