KREACIJSKI PATERNI

Potreba za dodavanjem kreacijskih paterna u sistem se prvobitno ogleda u cilju povećanja nivoa principa enkapsuiliranja, kako bi se dovelo do što većeg odvajanja korištenja sistema i u ovom slučaju kreiranja klasa koje sistemu trebaju. Cilj ovog postupka je sličan kao i kod mnogih drugih paterna, a to jeste smanjenje mogućnosti greške, te ako se greška i desi, pokušati ćemo minimizirati njen utjecaj. Želimo da greška uvijek utiče na aspekte sistema, a ne na čitav sistem.

U našem slučaju kreacijski paterni koje ćemo primjeniti su:

- Singleton patern
- Builder patern

1.**Singleton pattern**

-ovaj patern osigurava da postoji samo jedna instanca određene klase koja je dostupna globalno. U softver inženjeringu, singleton pattern je dizajn pattern koji ograničava instanciranje klase na jednu jedinu instancu. Ovo je korisno kada je potreban tačno jedan objekt za koordinaciju akcija u cijelom sistemu, i koristi se tamo gdje je potrebna samo jedna instanca klase za kontrolu radnje tokom cijelog izvršenja. Singleton klasa ne bi trebala imati više instanci ni u kom slučaju i po svaku cijenu. Singleton patern možemo primjeniti na obavijesti koja se prosljedjuje svim korisnicima koji uspješno izvrše narudžbu, tj. kako korisnik vidi obavijest, nije potrebno praviti zasebnu instancu za svakog korisnika.

2. **Builder pattern**

-koristi se za konstruisanje složenih objekata korak po korak. **Builder pattern** se koristi u slucaju kreiranja kompleksnih objekata gdje nerijetko dolazi do pojave velikog broja parametara (konstruktor pretezno) od kojih vecina moze biti neiskoristena.Takodjer, moze biti iskoristen u situacijama kada se razliciti objekti kreiraju na slicne nacine. Koristan je kada objekat ima mnogo opcionalnih parametara ili složene konfiguracije.Primjer builder paterna u našoj aplikaciji mogao bi se ogledati u kreiranju kompleksnog objekta narudžbe koja može imati različite dodatke, napomene i posebne zahtjeve itd.

Ostali paterni bi mogli biti iskorišteni :

1. **PrototypePatern**:

• **Primjena**: Prototype patern koristi se za kloniranje postojećih objekata umjestokreiranja novih od nule. Ovaj patern može biti koristan kada imamo objekte sa složenom konfiguracijom koje često ponavljamo.

• **Primjer**: Kloniranje prethodnih narudžbi korisnika kako bi se ubrzalo kreiranje nove porudžbine sa sličnim stavkama.

• Ovaj patern nismo implementirali jer nam je zahtijevao viši nivo kompleksnosti od našeg ciljnog nivoa.

2.**Factory Method patern**

Uloga **Factory Method paterna** je da omogući kreiranje objekata na način da podklase odluče koju klasu instancirati. Različite podklase mogu na različite načine implementirati interfejs. Factory Method instancira odgovarajuću podklasu(izvedenu klasu) preko posebne metode na osnovu informacije od strane klijenta ili na osnovu tekućeg stanja.

U nasem slucaju, mozemo iskoristiti **Factory Method** prilikom kupovine, gdje ukoliko vrsimo kupovinu proizvoda, metoda se izvrsava sa uracunatim popustom, dok sa druge strane, metoda se izvrsava na drugi nacin (nema takvu funkcionalnost) prilikom kupovine bez popusta, jer niko nije u mogućnosti ostvariti popust bez prethodnih narudžbi.

3.

Abstract factory patern

Kupovina je omogućena registrovanim i neregistrovanim korisnicima. Međutim, samo registrovani korisnici mogu izvršiti i kupovinu uz popust .Možemo napraviti apstraktnu factory klasu iz koje ćemo naslijediti dvije factory klase, za registrovane i neregistrovane korisnike. Shodno tome kakav korisnik je u pitanju, overrideana metoda koja kreira Kupovinu bi u jednom slučaju vraćala instancu Kupovine sa popustom, a u drugom Kupovina bez popusta(Obe ove klase su naslijeđene iz Kupovina).

Ako bismo u budućnosti željeli omogućiti specifičan način kupovine nekoj trećoj kategoriji korisnika, to bi nam sada bilo znatno olakšano. Naravno, ako bismo još neki segment sem kupovine smatrali pogodnim za „personalizaciju“ po ovim kategorijama korisnika, to bismo mogli riješiti tako što dodamo još jedan sistem sličan već postojećem koji se tiče kupovine i još jednu apstraktnu metodu unutar factory klase koja bi kreirala instance takvih objekata.