Kreacijski paterni

1. *Singleton*

   Uloga ovog kreacijskog paterna je da osigura da se klasa samo jednom instancira i da ima globalni pristup prema njenoj instanci. Gledajući naš sistem, možemo primijetiti da na samom početku naš sistem zahtjeva upotrebu logiranja. Budući da se upravo ovaj patern koristi kada se u sistemu dešava logiranje, možemo dodati novu klasu „LogTracker“. Ta klasa bi nam pomogla tako što bi u cijelom sistemu vršilo logiranje na najefikasniji mogući način. LogTracker mora imati privatni staticki atribut, privatni konstruktor i getter.

2. *Prototype*

   Uloga ovog kreacijskog paterna je da klonira objekte koji zauzimaju mnogo resursa. U našem sistemu ovaj patern možemo iskoristiti za klasu “Artikl”. Mogli bismo napraviti interfejs IArtikl koji bi nam omogućio da kloniramo instance, te bi nam omogućio brzo kreiranje većeg broja sličnih artikala.

3. *Factory Method*

   Uloga ovog kreacijskog paterna je da omogući kreiranje objekata na način da se odluči koja se podklasa instancira. Naš sistem bi se nekada u budućnosti mogao proširiti tako da korisnik u početku može izabrati da li kupuje u domovini (Bosni i Hercegovini) ili u inostranstvu. U tom slučaju bi nam ovaj patern mogao pomoći. Bilo bi potrebno napraviti dvije nove klase npr. KorpaUBiH i KorpaUInostranstvu. Pored ovih klasa moramo napraviti i klasu Kreator kako je definisano u samoj definiciji paterna. Ona bi morala imati metodu Factory koja bi instancirala odgovarajuću podklasu u sistemu.

4. *Abstract Factory*

   Ovaj kreacijski patern omogućava da se kreiraju familije povezanih objekata. Trenutno kako je dizajniran naš sistem on bi trebao da ispuni većini korisnika sve potrebe i zahtjeve. Ali svjesni smo da postoji određeni broj ljudi koji vole dizajnirati sami svoju odjeću (npr. promijeniti boju, dodati neku animaciju, kombinovati više boja i slično). Tu dolazi Abstract Factory patern koji bi nam omogućio da napravimo dvije familije produkata (odjeća koja je na tržištu i odjeća koju sami kreiramo). Morali bismo kreirati dva interfejsa IPonudaProdavnice i IDizajnKupca. Zatim bi mogli kreirati posebne klase za ova dva interfejsa.

5. *Builder*

Budući da ovaj patern pomaže kada imamo neku klasu koja je sklona nadograđivanju i dodavanju parametara u konstruktor, kod samim time postaje složeniji i nečitljiviji. Jedan klasični primjer korištenja ovog paterna je upotreba za klasu "Osoba". Sistem na početku traži minimalnu količinu informacija o korisniku, dok prilikom plačanja, od korisnika bi se zahtjevalo dosta više informacija. Builder će nam omogućiti da u oba slučaja klasu "Osoba" iskoristimo na najefikasniji način.