## **Kreacijski paterni**

1. **Singleton pattern**

Svrha singleton patterna je da omogućimo globalni pristup instanci neke klase, odnosno da se klasa može instancirati samo jednom.

U našem sistemu, ovaj patern se može primijeniti pri registraciji korisnika. Napravit ćemo jednu Logger klasu gdje će se nalaziti svi upiti, odakle ćemo dobavljati podatke, odnosno klasa u kojoj se nalazi konekcija na bazu. Na taj način će postojati samo jedna konekcija na bazu, što je vrlo bitno da ne bi došlo do greške zbog pokušavanja otvaranja više konekcija ka jednoj bazi istovremeno.

1. **Prototype**

Svrha prototype patterna je da klonira objekte koji zauzimaju previše resursa. U našem sistemu, najbolji primjer za to su artikli, zato što ćemo imati jako velik broj približno sličnih objekata. Stoga bi bilo dobro napraviti interfejs koji će klonirati instance artikala, što bi olakšalo rad sa istim. Taj interfejs bi sadržavao samo metodu clone(), koja bi stvorila novi objekat klase prenoseći sve vrijednosti atributa starog objekta u novi.

1. **Factory method**

Svrha factory method patterna je da omogući kreiranje objekta na način da se odluči koja se podklasa instancira. Ako bismo u budućnosti napravili suradnju sa nekim kompanijama/agencijama/sponzorima koji bi omogućili upotrebu raznih kupona ili povlastica (npr. Aplikacija za štednju preko koje se skupljaju bodovi kupujući na našoj aplikaciji te tako ostvaruju određene pogodnosti), onda bi nam u budućnosti trebale 2 različite košare, npr. Košara za običnog registrovanog usera, te košara za korisnika koji dolazi od partnerske organizacije.

1. **Abstract factory**

Svrha abstract factory patterna je da omogući kreiranje familija povezanih objekata. Recimo ako bismo budućnosti planirali proširenje naše online trgovine uvodeći i Prom edition kolekciju, tada bismo imali dvije familije produkata (limited edition za mature i redovna ponuda). Mogli bismo kreirati dva interfejsa npr. IProm i ICasual, a zatim kreirati klase za ova dva interfejsa.

1. **Builder**

Svrha abstract patterna je da smanji komplikovanje klasa koje su sklone nadograđivanju. U našem sistemu, nadograđivanje se najviše očekuje u korisničkim klasama, gdje će se pri kupovini tražiti još podataka o korisniku (npr. adresa, broj telefona, broj kartice i sl.). Umjesto da dodajemo sve više i više atributa u klasu i konstruktore, puno je praktičnije uvesti klasu Builder koja će sadržavati sve atribute kao i korisničke klase. Zatim ćemo izbrisati konstruktor u korisničkim klasama, pa ga zamijeniti tako da samo builder može kreirati instancu.