

KREACIJSKI PATERNI – SOFTWARE

1. Singleton patern

Uloga **Singleton patern** je da osigura da se klasa može instancirati samo jednom i da osigura globalni pristup kreiranoj instanci klase.

U našem slučaju, singleton paternom možemo implementirati sve klase koje komuniciraju sa serverom, tj. bazom podataka (npr. kada bi iz baze preuzimali listu letova).

2. Prototype patern

Uloga **Prototype patern** je da kreira nove objekte klonirajući jednu od postojećih prototip instanci (postojeći objekat). Ako je trošak kreiranja novog objekta velik i kreiranje objekta je resursno zahtjevno, tada se vrši kloniranje već postojećeg objekta. Prototype dizajn patern dozvoljava da se kreiraju prilagođeni objekti bez poznavanja njihove klase ili detalja kako je objekat kreiran.

U našem slučaju, možemo koristiti ovaj patern nad klasom *Karta*. U mnogim situacijama kada bi nam zatrebao identičan objekat klase *Karta* (dodavanje u *KupovinuKarte*, dodavanje u historiju kupljenih karata korisnika i sl.), možemo jednostavno pozvati metodu `clone()`.

3. Factory Method patern

Uloga **Factory Method patern** 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 našem slučaju, možemo iskoristiti **Factory Method** prilikom kupovine, gdje ukoliko vrsimo kupovinu karte (objekat klase *KupovinaKarte*) metoda se izvršava sa uracunatim popustom, dok sa druge strane, metoda se izvršava na drugi način (nema takvu funkcionalnost) prilikom kupovine gift koda (objekat klase *KupovinaGiftKoda*) jer se popust isključivo uracunava samo prilikom kupovine karte (Ne postoji mogućnost popusta za kupovinu gift koda).

4. Abstract factory patern

Nema prostora za primjenu ovog patterna jer nasa aplikacija nigdje ne radi sa vecim brojem srodnih objekata / klasa.

5. Builder patern

Builder pattern se koristi u slucaju kreiranja kompleksnih objekata gdje nerijetko dolazi do pojave velikog broja parametara (konstruktor pretežno) od kojih vecina moze biti neiskoristena. Takodjer, moze biti iskoristen u situacijama kada se razliciti objekti kreiraju na slicne nacine.

Ovaj pattern bi se mogao primijeniti na kreiranje objekata tipa *KupovinaKarte* i *Let*.

Ove dvije klase sadrze odredjene atribute koji se odnose na istu informaciju (*KupovinaKarte* sadrzi opis leta kakvog korisnik trazi, a klasa *Let* sadrzi isti takav opis pomocu kojeg nalazimo odgovarajuci let).

Nigdje drugo nema prostora za primjenu ovog patterna. Sve klase su prilicno jednostavne i ni za jednu nije potrebna step by step procedura kod kreiranja objekata tog tipa.