KREACIJSKI PATERNI – SOFTWAIR

1. **Singleton patern**

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

U nasem slucaju, klasu *ListaLetova* koja nam sluzi da pokupimo letove sa baze mozemo napraviti kao singleton klasu.Ona bi nam sluzila kao globalni resurs, gdje druge klase mogu koristiti njen atribut u zavisnosti od potreba.

2. **Prototype patern**

Uloga **Prototype paterna** 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 objekata. Prototype dizajn patern dozvoljava da se kreiraju prilagođeni objekti bez poznavanja njihove klase ili detalja kako je objekat kreiran.

U nasem slucaju, mozemo koristiti ovaj patern nad klasom *Karta*.U mnogim situacijama kada bi nam zatrebao identican objekat klase *Karta* (dodavanje u *KupovinuKarte*, dodavanje u historiju kupljenih karata korisnika i sl.), mozemo jednostavno pozvati metodu clone().

3. **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 karte (objekat klase *KupovinaKarte*) metoda se izvrsava sa uracunatim popustom, dok sa druge strane, metoda se izvrsava na drugi nacin (nema takvu funkcionalnost) prilikom kupovine gift koda (objekat klase *KupovinaGiftKoda*) jer se popust iskljucivo uracunava samo prilikom kupovine karte (Ne postoji mogucnost popusta za kupovinu gift koda).

4**. Abstract factory patern**

Kupovina je omogućena registrovanim i neregistrovanim korisnicima. Međutim, samo registrovani korisnici mogu izvršiti i kupovinu gift koda. Pretpostavimo da registrovani korisnici mogu kupiti samo gift kod, ali ne i kartu (a neregistrovani samo kartu). 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 karte, a u drugom Kupovine gift koda (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.

5. **Builder patern**

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.

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.