KREACIJSKI PATTERNI ZA ULAZNICE.COM

1) SINGLETON PATTERN

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 kupe kartu, tj. kako korisnik vidi obavijest, nije potrebno praviti zasebnu instancu za svakog korisnika.

2) PROTOTYPE PATTERN

Ovaj pattern se koristi kada želimo da izbjegnemo bespotrebno gomilanje klasa i objekata koji su međusobno slični. U našem sistemu, ovaj patern bi se mogao zadovoljiti u klasi Karta. Ukoliko želimo da dodamo novu ulaznicu, mnogo nam je lakše da kloniramo prethodnu ulaznicu i mijenjamo informacije o njoj (npr karta za nogometnu utakmicu, ako imamo drugu utakmicu promijenimo vrijeme, timove, stadion i sl.).

3) FACTORY METHOD PATTERN

U objektno-zasnovanom programiranju, Factory Method Pattern je kreacijski pattern koji koristi factory metode za rješavanje problema kreiranja objekata, bez potrebe da se specificira tačna klasa objekta nad kojom će biti kreiran. Factory je funkcija ili metoda koja vraća objekte promjenjivog prototipa ili klase iz nekog poziva metode, za koji se pretpostavlja da je "novonastala". Ovaj pattern bi iskoristili na način da prilikom nagrađivanja najlojalnijeg kupca, kojeg biramo iz liste sortiranje po broju ostvarenih kupovina u BAM, pazimo da jedan te isti korisnik ne bude svaki mjesec nagrađivan. Kreirali bi izvještaj koji bi sačinjavao podatke npr. za koji tip manifestacije najlojalniji kupci najčešće kupuju ulaznice i koliku sumu novca izdvajaju za njih. Iznimku bi učinili jedino ako bi jedan te isti kupac iz mjeseca u mjesec ostvarivao kupovinu koja je za npr za 100-200 KM veća od ukupnog broja kupovina prethodni mjesec, što bi bilo javno objavljeno.

4) ABSTRACT FACTORY PATTERN

Abstract Factory patern nam omogućava da se kreiraju familije povezanih objekata. Na osnovu apstraktne familije produkata kreiraju se konkretne fabrike produkata različitih tipova i različitih kombinacija, a pošto patern odvaja definiciju klase produkata od klijenta, familije produkata je moguće jednostavno prikazivati, mijenjati i ažurirati. Abstract Factory patern možemo iskoristiti kada korisnik bude vršio filtriranje manifestacija po različitim aspektima tokom pretrage (vrsta manifestacije, vrijeme, da li je manifestacija na otvorenom/zatvorenom, itd). Na osnovu

korisničkih filtera kreiraće se i fabrika produkata različitih tipova artikala kao i različitih kombinacija. Ovim bi se prikaz artikala učinio dosta efikasnijim s obzriom da Abstract Factory patern upravlja familijama produkata i čuva njihove detalje neovisnim od klijenata.

5) BUILDER PATTERN

Uloga Builder paterna je da pruži fleksibilno rješenje za različite probleme kreiranja objekata u objektno orijentisanom programiranju. Namjena Builder patterna je da odvoji konstrukciju složenog objekta od njegovog predstavljanja. U našem projektu, Builder patern bi se mogao iskoristiti kod postavljanja korisničkih podataka. Korisnik bi sam mogao mijenjati svoju korisničku šifru, dodavao nove Bankovne račune preko kojih će ići plaćanje itd, a sve to bez odobrenja administratora.