1. SINGLETON

Singleton kreacijski pattern osigurava instataciju jedne globalne klase kojoj svako može pristupiti.

U većini programa postoji potreba za nekim objektima koje je potrebno samo jednom instancirati i nad kojim je potrebna jedinstvena kontrola pristupa.

Ovo se postiže sa privatnim konstruktorom, te upotrebom statičkog atributa tipa same klase. Također je potrebno napraviti javnu statičku kreacionu metodu. Kako bi osigurali da uvijek može postojati najviše jedna instanca singleton klase, ova metoda će koristiti if statement koji provjerava da li istanca uopšte postoji, te ako postoji, vraća reference na njega. U suprotnom, kreira novi singleton objekat i vraća reference na njega.

Mi smo iskoristili Singleton pattern da bi realizirali funkcionalnost preporuka. Svaki korisnik može na početnoj stranici vidjeti preporučeni koncert, baziran na ocjenama izvodača. Ovaj koncert se bira svake sedmice.

2. BUILDER

Uloga ovog paterna je odvajanje konstrukcije objekta od njegove reprezentacije, kreiranjem builder klase koja sadrži iste parametre kao i objekat koji treba kreirati.

Ovaj pattern bi mogli iskoristiti za kreiranje klasa Dvorana. Konstruktor ove klase nema previše parametara, mada, u budućnosti, mogu biti dodani dodatni atributi ovoj klasi, pa ćemo u tom slučaju radi preglednosti i jednostavnosti, kreirati Builder klasu koja ima metode koje će postavljati atribute na odgovarajuće vrijednosti. Ovo omogućava kreiranje koncerta na više načina.

3. PROTOTYPE

Prototype patern kreira nove objekte putem klonova već postojećih. Ovo primjenjujemo ukoliko je kreiranje novih objekata resursno zahtjevno, a pogotovo ako ti objekti sadrže dijelove koji ostaju isti.

Konkretno u našem projektu, ovaj patern bi mogli primjeniti za kloniranje instanci ObičniKorisnik, Izvođač ili Iznajmljivač, ukoliko bi one sadržavale više kompleksnih atributa. Prema našem

dijagramu klasa, navedene klase su još uvijek dovoljno jednostavne da primjena ovog paterna nije previše potrebna.

4. FACTORY METHOD

Factory metod pattern koristimo ukoliko nismo sigurni tačno sa kakvim tipom objekata će naš kod raditi. Ovaj pattern proširuje konstrukcijski kod nekog objekta nezavisno od ostatka aplikacije. U budućnosti, ovo omogućava uvođenje novih objekata bez kvarenja dijelova postojećeg koda. Ovaj metod također centralizira kreiranje objekata na jedno mjesto u aplikaciji.

U našem sistemu, imamo više vrsta koncerata (ovisno od žanra), pa nalazimo primjenu za ovaj pattern. Kreiranjem nekog interfejsa IVrstaKoncerta, zajedno sa podklasama koje odgovaraju vrstama koncerata,

5. ABSTRACT FACTORY

Ovaj pattern koristimo kada naš program treba da radi sa različitim familijama povezanih objekata. Ovaj pattern zapravo predstavlja nastavak dizajnova koji počnu kao Factory Method a kasnije evoluiraju u Abstract Factory. Ovaj pattern prati open-closed i single-responsibility principe. Slično kao i prethodni pattern, ovaj metod također centralizira kreiranje objekata na jedno mjesto u aplikaciji.

U našem sistemu, ovaj pattern bi smo mogli realizirati ako malo proširimo kompleksnost samog sistema. Na primjer, dodali bi smo novi tip korisnika, koji, s obzirom na neki koncert, ima veće mogućnosti od običnog korisnika, ali manje od npr. Izvođača ili vlasnika dvorane.