Univerzitet u Sarajevu
Elektrotehnički fakultet u Sarajevu
Odsjek za računarstvo i informatiku

ReadTheWorld

Kreacijski patterni

Objektno orijentisana analiza i dizajn

Iris Pjanić
Adna Husičić
Mirza Kadrić

Kreacijski patterni

Singleton pattern

Singleton pattern se koristi u situacijama kada trebamo osigurati da se klasa može samo jednom instancirati te da bi se obezbjedio globalni pristup kreiranoj instanci klase.

Ovaj dizajn patern bi bio pogodan za neku kontejnersku klasu ActiveTakmičenja koja bi sadržavala informacije o svim trenutno aktivnim takmičenjima. Upotrebom singletona bismo zaobišli potencijalni konflikt koji bi se mogao desiti kad je više korisnika u interakciji sa istim takmičenjem.

Prototype pattern

Uloga ovog patterna je da kreira nove objekte klonirajući jednu od postojećih prototip instanci (postojeći objekat) u slučajevima kada je trošak kreiranja novog objekta velik i kada je kreiranje objekta zahtjevno po pitanju resursa.

Kada pogledamo naš sistem, primjećujemo da su klase Rad i Korisnik najkompleksnije te bi one bile najpogodnije za primjenu ovog patterna. Kako se najveći dio sistema bazira na manipulaciji sa radovima i feedbacku korisnika, višestruko korištenje podataka iz baze će biti potrebno. Kopiranje podataka iz baze i enkapsulacija u objekat može biti nešto duži proces i u tim slučajevima bolja opcija bi bila kloniranje već postojećeg objekta koji se trenutno analizira (obrađuje).

Factory method pattern

Factory Method patern omogućava kreiranje objekata na način da podklase odluče koju klasu instancirati. Različite podklase mogu na različite načine implementirati interfejs. Factory Method instancira odgovarajuću podklasu preko posebne metode na osnovu informacije od strane korisnika ili na osnovu tekućeg stanja.

U našem sistemu, Factory Method mozemo iskoristiti prilikom prijavljivanja (korisnika ili rada) adminu, kada admin odluči da kazni korisnika brisanjem rada ili njegovog

korisničkog računa. Recimo da uvedemo interfejs ISubjekt kojeg će implmentirati klase Rad i Korisnik i koji će imati metodu za brisanje rada ili korisnika u zavisnosti šta je od to dvoje prijavljeno adminu. Admin bi tada preko Factory metode klase Creator mogao pokrenuti brisanje subjekta svejedno da li je riječ o radu ili korisniku.

Navedimo drugi primjer. Recimo da je sistem dizajniran tako da rad nema žanrova ni kategorija, samo da može pripadati poeziji ili prozi. Tada bi interfejs IVrsta (koji definira metodu za ispis teksta djela) implementirale klase Proza i Poezija. Opet na isti način preko Factory metode bi mogli omogućiti različit prikaz teksta djela u odnosu da li je u pitanju poezija ili proza.

Abstract factory pattern

Abstract Factory patern omogućava da se kreiraju familije povezanih objekata/produkata. Na osnovu apstraktne familije produkata kreiraju se konkretne fabrike (factories) produkata različitih tipova i različitih kombinacija. Patern odvaja definiciju (klase) produkata od klijenta.

Trenutno u sistemu nema naročito sličnih objekata da bi mogli primjeniti ovaj patern. Kada bi npr. aplikacija bila proširena na objavljivanje likovnih radova pored literarnih, mogli bismo sva ta djela posmatrati kao slične objekte (iako različite) te pomoću apstraktne familije produkata i fabrika kreirati, mijenjati ili ažurirati objekte bez kaskadnog narušavanja strukture sistema.

Builder pattern

Builder pattern odvaja specifikacije kompleksnih objekata (objekata koji imaju mnogo parametara) od njihove stvarne konstrukcije. Dakle, konstrukcija se izvršava step by step za šta je zadužena Builder klasa čime se povećava održivost i čitljivost koda. Isti konstrukcijski proces može kreirati različite objekte.

Gledajući dijagram klasa za naš sistem, primjećujemo da nema glomaznih i komplikovanih klasa koje bi direktno mogle biti primjenjive na Builder pattern. Ako bismo htjeli uvesti

funkcionalnost da korisnik ima pregled aktivnosti na sistemu (objavio rad, dao recenziju, dao komentar, pročitao rad, posjetio profil drugog korisnika itd.) morali bismo nadograđivati Korisnik klasu gdje bi nam Builder patern bio od velike pomoći čime bismo težinu implementacije same klase prebacili na Builder klasu.