**Kreacijski paterni**

***Factory method patern***

Factory method patern se koristi kada se želi instancirati izvedena klasa preko posebne metode na osnovu informacija od strane klijenta ili na osnovu tekućeg stanja. Za našu aplikaciju, ovaj patern bi smo mogli iskoristiti pri kreiranju instanci izvedenih klasa iz klase *User.* Da bi neki user mogao biti *Editor*, *Administrator* ili *Kritičar,* treba dobti dozvolu od strane *Admina.* Potrebna nam je klasa *UserFactory*, koja sadrži metodu, koja na osnovu toga da li je Admin dao dozvolu, instancira konkretnu podklasu *User*-a.

***Builder patern***

Builder patern bi u našoj aplikaciju mogli iskoristiti za kreiranje klase *Movie.* Kako konstruktor za ovu klasu ne bi imao previše parametara, a pogotovo u budućnosti ako bismo dodali još atributa ovoj klasi, radi jednostavnosti i preglednosti, možemo kreirati Builder klasu koja bi imala metode koje bi postavljale atribute na odgovarajuće vrijednosti. Administrator i Editor bi pozivali klasu *MovieBuilder* da se izvrši konstrukcija objekta tipa *Movie*.

***Protutip***

Prilikom prikaza različitih stranica kao što je stranica koja prikazuju favourite filmove, koja prikazuje rezultate pretraživanja, početna stranica I tako dalje, možemo primjetiti da sve one sadrže neke djelove koji ostaju isti. Primjenom protutip paterna, prilikom prelaska s jedne od ovih stranica na drugu moguće je klonirati objekat stranice s koje se prelazi i nakon minornih izmjena dobiti objekat koji odgovara stranici na koju se prelazi.

***Abstract factory patern***

Abstract factory patern u našem sistemu se može iskoristiti ako malo proširimo kompleksnost samog Sistema. Ono što bismo morali uvest, pored već postojećih registrovanih korisnika sistema su takvi korisnici koji su na skali ‘moći’ iznad običnih korisnika, ali ispod I kritičara filmova. To bi bila jedna od factory klasa, nazvat ćemo je *Grader* (onaj koji ocjenjuje), dok bi druga factory klasa bila upravo već postojeća *Critic*. Uveli bismo apstraktnu klasu *Review*, iz koje ćemo naslijediti klase *Rating* I klase *Comment*. *Rating* klasa je predviđeba za *Grader*-a, dok je *Comment* klasa tu zbog *Critic*-a. U ovom slučaju kada stvorimo objekat *Grader*-a I *Critic*-a tačno znamo njegove mogućnosti.

|  |
| --- |
| public abstract class Review { }  public class Rating : Review { }  public class Comment : Review { }  public interface IFactory {  public Review getReview();  }  public class CriticFactory : IFactory {  Comment review = new Comment ();  public Review getReview() {  return review;  }  }  public class GraderFactory : IFactory {  Grade review = new Grade ();  public Review getReview() {  return review;  }  } |

***Singleton***

Klasa koja će raditi sa API-em će sadržavati GET i POST zahtjeve koji će biti statički dok će njihovi parametri biti dinamički. U toj klasi ćemo pomću argumenata funkcija podesavati potrebne parametre url-a. Kako se objekat ove klase neće mjenjati i kako bi imali neometan rad s serverom ovu klasu cemo pretvoriti u singleton

|  |
| --- |
| public sealed class APIManager {  private static APIManager instance = null;  private static APIManager Instance () {  get{  if(instance==null) instance = new APIManager();  return instance;  }  }  } |