Univerzitet u Sarajevu
Elektrotehnički fakultet
Sarajevo

Deveti projektni zadatak

Objektno orijentisana analiza i dizajn

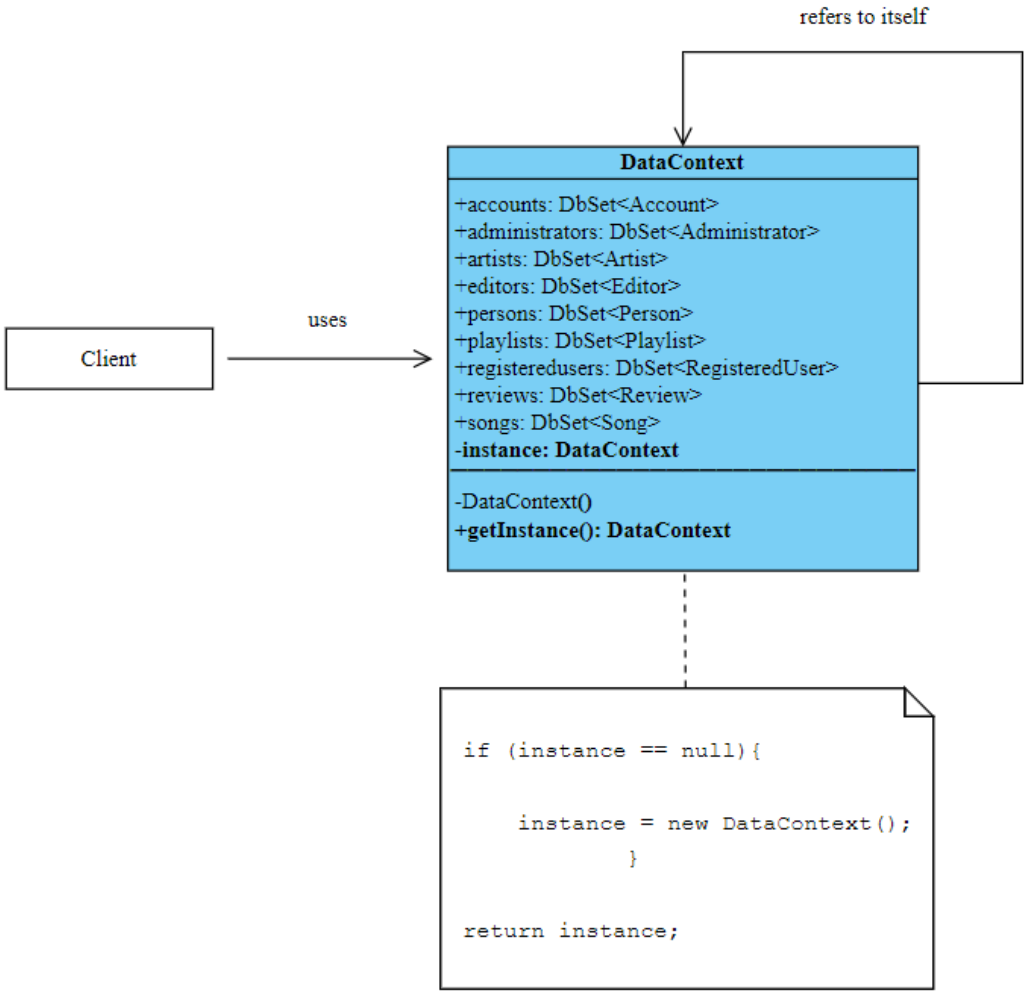
Naziv grupe: Spotifive
Članovi grupe: Nadina Miralem 18937
Amina Hromić 19084
Nerma Kadrić 19030
Amila Kukić 19065
Una Hodžić 19044

Kreacijski paterni

U nastavku slijedi opis kreacijskih paterna koji će biti primijenjeni u sistemu.

Singleton

Singleton patern ograničava instanciranje klase. Singleton klase mogu imati samo jedan objekat u bilo kojem trenutku. S obzirom na to da naša aplikacija zahtijeva pristup bazi podataka iz različitih dijelova koda, možemo implementirati Singleton klasu koja sadrži logiku za uspostavljanje veze s bazom podataka i izvršavanje upita. Na taj način možemo osigurati da postoji samo jedna instanca veze s bazom podataka u cijeloj aplikaciji.



U nastavku možemo vidjeti isječak koda kojim je ovaj patern implementiran u našem sistemu. Klasa DataContext na dijagramu ustvari predstavlja klasu ApplicationDbContext.

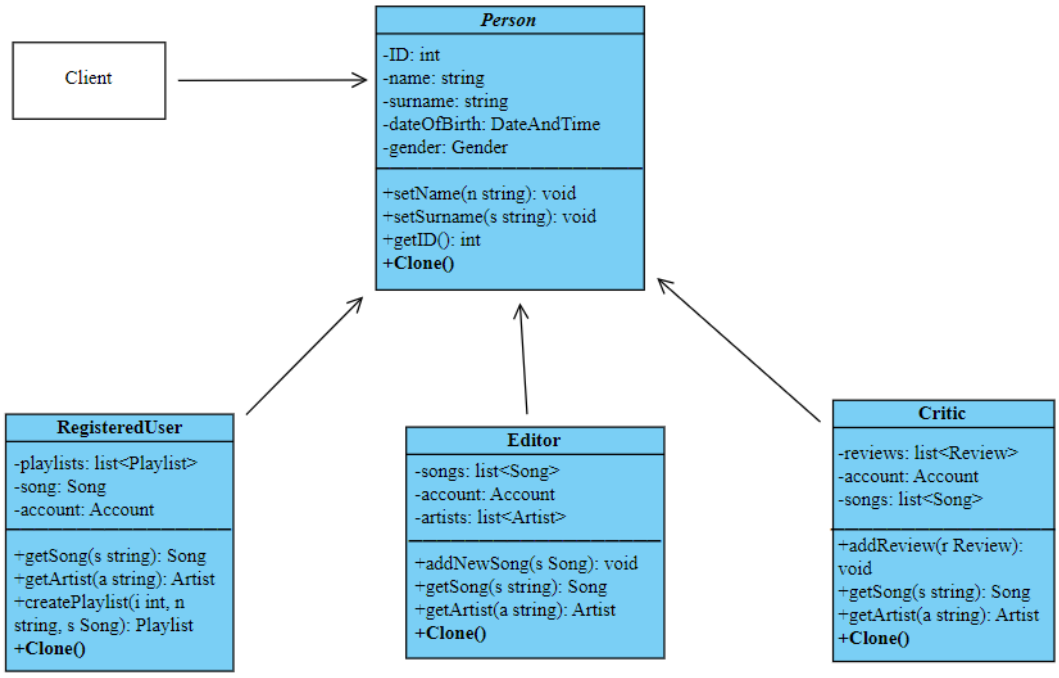
```
private static ApplicationDbContext instance;

        private ApplicationDbContext()
        {
        }
        public static ApplicationDbContext getInstance()
        {
            if (instance == null)
            {
                instance = new ApplicationDbContext();
            }
            return instance;
        }

        public Connection getConnection()
        {
            return null;
        }
```

Iz priloženog vidimo da je za implementaciju Singleton paterna potreban privatni konstruktor (kako bismo onemogućili druge klase da pomoću operatora new stvaraju nove instance klase ApplicationDbContext), zatim nam je potrebna statička metoda za kreiranje instance koja se ponaša kao konstruktor. Ova metoda zapravo poziva konstruktor kako bi kreirala objekat i sprema ga u statičko polje. Svi sljedeći pozivi ove metode vraćaju keširani objekat.

Prototype

Ovaj patern odnosi se na kloniranje objekata. Drugim riječima, omogućava da klase budu kopirane ili klonirane iz neke instance koja predstavlja prototip, umjesto da se kreiraju nove instance. Prototype patern ima mnoge prednosti, između ostalog ubrzava instanciranje veoma velikih klasa koje se dinamički učitavaju.

Ukoliko bismo Prototype patern koristili u našem sistemu, nad klasom Person, te ukoliko bi se pojavilo više korisnika sa istim imenom i prezimenom (ili bilo kojom drugom osobinom koja je opisana atributom te klase), mogli bismo iskoristiti prvu instancu, klonirati je a zatim samo promijeniti podatke koji se razlikuju. Zbog toga dodajemo interface IPrototype koji implementira klasa Person, a koji ima metodu Clone.




```
public abstract class Person{
        [Key] public int ID { get; set; }
        public string Name { get; set; }
        public string Surname { get; set; }
        public DateTime DateOfBirth { get; set; }
        public Gender Gender { get; set; }
        public Account Account { get; set; }
        public Person() { }

        public virtual Person Clone();
}
```

Kao što možemo vidjeti, gore navedena klasa ima nekoliko atributa koji su zajednički za podklase RegisteredUser, Critic, Editor i Administrator, te apstraktnu metodu koja će biti implementirana od strane navedenih podklasa.

Kao primjer, u klasi RegisteredUser možemo vidjeti implementaciju spomenute apstraktne metode. Metoda Clone stvara površinsku kopiju trenutnog objekta.

```
public class RegisteredUser: Person{

public RegisteredUser() { }

[ForeignKey("Song")] public int SongID { get; set; }

public Song Song { get; set; }


public override Person Clone(){

RegisteredUser rUser = (RegisteredUser) this.MemberwiseClone();

rUser.Name = (string) this.Name.Clone();

rUser.Surname = (string) this.Surname.Clone();

rUser.DateOfBirth = (DateTime) this.DateOfBirth;

rUser.Gender = (Gender) ((int)this.Gender);

return rUser;

        }

}
```

U nastavku slijedi opis ostalih kreacijskih paterna, te načina na koje bi se oni mogli implementirati u našem sistemu.

Builder

Kako ovaj patern odvaja specifikaciju kompleksnih objekata od njihove stvarne konstrukcije, te kako u našem sistemu nema nekih kompleksnih objekata koji bi zahtijevali upotrebu Builder paterna, njega nismo implementirali.

Factory method

Ovaj patern bi u našem sistemu mogao biti iskorišten prilikom pristupa žanrovima pjesama. Dodali bismo interface ISong, te klase Pop, Rock, Jazz, Folk i slične, koje implementiraju taj interface. Tako bismo izbjegli korištenje različitih metoda za kreiranje različitih žanrova.

Abstract Factory

Kod nas ovaj patern možemo primijeniti kod filtriranja pjesama tokom pretrage. Na osnovu nekog filtera, kreirala bi se fabrika produkata različitih pjesama i tako bi se njihov prikaz učinio mnogo efikasnijim jer ovaj patern upravlja familijama objekata i čuva sve informacije o njima.