

# Dizajn paterni

## 1 KREACIJSKI DIZAJN PATERNI

---

### 1.1 SINGLETON

Ovaj patern bi se mogao koristiti za klasu koja pripada Data Access Layeru, FakultetDataSource. Prvenstveno iz razloga što služi za pristupanje bazi koja pripada određenom frameworku, te bi nadogradnja korištenjem nekog drugog paterna bila nezgodna. Drugi razlog je što je dovoljna jedna instanca klase koja komunicira s bazom (jer komunicira samo sa jednom bazom), te bi u slučaju višenitnog pristupa moglo doći do konflikta nad resursom. Implementacija je trivijalna, što se vidi na slici:

FakultetDataSource
-static instancaBaze : Singleton -const DbPath : String -Korisnici : DbSet<Korisnik> -Grupe : DbSet<Grupa> -Sale : DbSet<Sala> -Laboratoriji : DbSet<Laboratorij> -Zahtjevi : DbSet<Zahtjev> -Termini : DbSet<Termin>
-FakultetDataSource() +getInstance() +getKorisnici() : DbSet<Korisnik> +setKorisnici(Korisnici : DbSet<Korisnik>) : void +getGrupe() : DbSet<Grupa> +setGrupe(Grupe : DbSet<Grupa>) : void +getSale() : DbSet<Sala> +setSale(Sale : DbSet<Sala>) : void +getLaboratoriji() : DbSet<Laboratorij> +setLaboratoriji(Laboratoriji : DbSet<Laboratorij>) : void +getZahtjevi() : DbSet<Zahtjev> +setZahtjevi(Zahtjevi : DbSet<Zahtjev>) : void +getTermini() : DbSet<Termin> +setTermini(Termini : DbSet<Termin>) : void +OnConfiguring(optionsBuilder : DbContextOptionsBuilder) +DajSaluById(id : int) : Sala +DajKorisnikById(id : int) : Korisnik +DajGrupuById(id : int) : Grupa +DajZahtjevById(id : int) : Zahtjev

Istu funkcionalnost bismo mogli postići kad bismo klasu proglasili za statičku (što nam C# omogućava), čime bi sve operacije nad klasom morale biti proglašene statičkim.

### 1.2 PROTOTYPE

U suštini, u sklopu našeg projekta nema velike potrebe za korištenjem prototipa, jer nema naročito velikih objekata.

### 1.3 FACTORY METHOD

Mogao bi se koristiti za klase koje služe za slanje zahtjeva, iako bi bilo donekle nepotrebno. Klasa ZahtjevRezervacija je izvedena iz klase Zahtjev. Možemo izbjeći eksplicitno instanciranje različitih tipova klasa ako ih, u slučaju da korisnik šalje zahtjev sa dodatnom informacijom, enkapsuliramo u Factory method.

### 1.4 ABSTRACT FACTORY

Nema ovolikog nivoa kompleksnosti u projektu.

### 1.5 BUILDER

Kao i za abstract factory, nema klasa koje enkapsuliraju dovoljan broj drugih da bi korištenje ovog paternu imalo smisao

## 2 STRUKTURALNI DIZAJN PATERNI

---

### 2.1 ADAPTER

U suštini, svaka komponenta Viewmodela je predstavnik adapter paternu. Same interakcije između pogleda i modela još nisu do kraja implementirane u projektu, ali su tipičan primjer povezivanja kakvo vrši adapter. Adapterom se i nadogradnja novih pogleda čini lakšom, u smislu da ne moramo dirati biznis logiku klasa da bismo dodali novu stranicu u layout.

### 2.2 FAÇADE

Fasada obezbjeđuje pojednostavljeni pogled na podsistem (poput abstract factory), kakvih u suštini nemamo u projektu, tako da nema mogućih primjena.

### 2.3 DECORATOR

Nema razloga za korištenje dekoratera, pošto se svaka instanca klase tretira isto, te predstavlja red u tabeli, samo bi narušilo strukturu interne logike, iako direktno ne mijenja originalni objekat.

### 2.4 BRIDGE

Kako promjene u implementaciji ne bi trebale imati utjecaja na klijenta, korisno je bridge patternom razdvojiti implementacionu logiku od interfejsa. Ovo bi mogao biti neki sloj apstrakcije između modela i viewmodela.

### 2.5 COMPOSITE

-

### 2.6 PROXY

-

### 2.7 FLYWEIGHT

-

## 3 PATERNI PONAŠANJA

---

### 3.1 CHAIN OF RESPONSIBILITY

-

### 3.2 COMMAND

-

### 3.3 INTERPRETER

-

### 3.4 ITERATOR

-

### 3.5 MEDIATOR

-

### 3.6 MEMENTO

-

### 3.7 OBSERVER

-

### 3.8 STATE

-

### 3.9 STRATEGY

-

### 3.10 TEMPLATE

-

### 3.11 VISITOR

-

### 3.12 NULL OBJECT

-