

# KREACIJSKI PATERNI

## 1. SINGLETON PATTERN

- U našem sistemu login je potreban da bi se koristila većina funkcionalnosti. Ovaj patern se koristi kada se odvija više logiranja, pa možemo dodati novu klasu LogSingleton koja će vršiti logiranje u sistemu na najefikasniji mogući način. Ona će sadržavati privatni statički atribut, privatni konstruktor i gettere.
- Također, može se iskoristiti klasa ErrorSingleton kao error manager koja će upravljati svim errorima umjesto da kreiramo više error manager-a.

## 2. PROTOTYPE PATTERN

Pretpostavljajući da će neki atributi kao što su tipDogađaja, opisDogađaja, posebneNapomene i dobnoOgraničenje biti isti za većinu događaja, pri kreiranju novog događaja trebali bismo svaki put prolaziti kroz sve te attribute i kopirati njihove vrijednosti u novi objekat. Zbog toga možemo napraviti interfejs IDogađaj koji bi nam omogućio da kloniramo instance Događaj klase.

## 3. FACTORY METHOD PATTERN

U slučaju da imamo više tipova notifikacija, patern bi mogao biti primjenjen na način da postoji FactoryKlasa koja sadrži metodu Factory() koja na osnovu prosljeđenih podataka o događaju kreira odgovarajuće notifikacije.

## 4. ABSTRACT FACTORY PATTERN

Mi ne vidimo konkretnu primjenu ovog paternu u našem sistemu.

## 5. BUILDER PATTERN

Ovaj patern možemo iskoristiti pri kreiranju događaja pri čemu bismo u par koraka razdvojili informacije o događaju tako da bismo ih ponovo mogli koristiti za kreiranje nekog novog događaja. Za implementaciju ovog paternu moramo imati Klijent klasu koja će pozivati Direktor klasu u kojoj su neophodne sekvence za stvaranje objekta. Zatim bismo imali interfejs IKreatorDogađaja koja definiše specifične metode potrebne za kreiranje događaja. Zatim KreatorDogađaja klasu koju poziva Direktor i koja stvarno kreira događaj, dok nam Produkt klasu čini klasa Događaj,