Strukuralni i kreacijski patterni

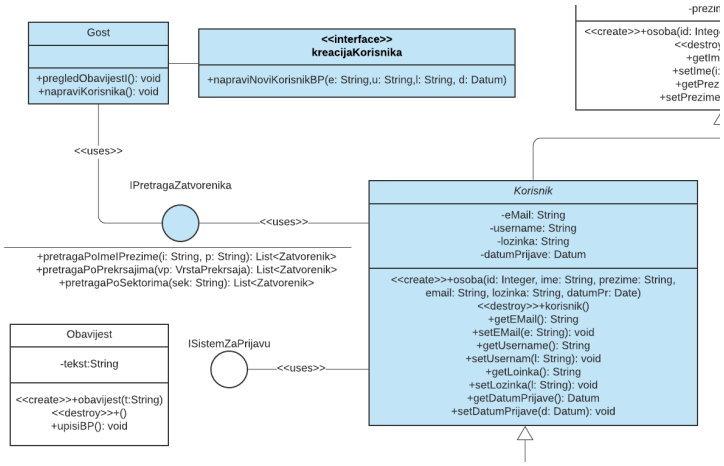
Predmet: Objektno orijentisana analiza i dizajn

Grupa: Tartarus
Hadžić Ajdin
Halilović Kemal
Mehmedović Faris

Strukuralni patterni

1. Adapter pattern

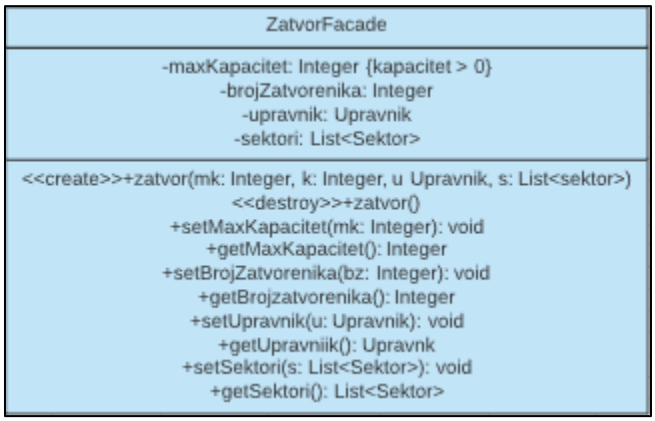
Osnovna namjena Adapter paterna je da omogući širu upotrebu već postojećih klasa. U situacijama kada je potreban drugačiji interfejs već postojeće klase, a ne želimo mijenjati postojeću klasu koristi se Adapter patern. Adapter patern kreira novu adapter klasu koja služi kao posrednik izmedu orginalne klase i željenog interfejsa. Primjer Adapter paterna se proicira unutar *Gost* klase koja omogućava da dva interfejsa (sa naizgleda različitim pravcima) funkcionišu zajedno. Klasa *Gost* implementira svaku metodu interfejsa *IPretragaZatvorenika* i *kreacijaKorisnika*.



Slika 1. Primjer Adapter pattern primjenjen na class diagramu Tartatus aplikacije

2. Facade pattern

Ovaj patern se korisiti s ciljem da osigura više pogleda visokog nivoa na podsisteme (implementacija podsistema skrivena od korisnika). Namjena facade paterna se može prepozanti unutar klase *Zatvor*, pomoću koje se pristupa upravniku, sektorima i obavijestima na nivu kazneno-popravnog zavoda. Patern je izveden tako što je klasa *Zatvor* povezana sa klasama *Upravnik* i *Sektor*, kao i interfejsom *ZatvorBP*.



Slika 2. Primjer Facade pattern primjenjen na class diagramu Tartatus aplikacije

3. Decorator pattern

Osnovna namjena Decorator paterna je da omogući dinamičko dodavanje novih elemenata i ponašanja (funkcionalnosti) postojećim objektima. Objekat pri tome ne zna da je urađena dekoracija što je veoma korisno za ponovnu upotrebu komponenti softverskog sistema. U našem sistemu konkretno ne možemo navesti primjer ovog paterna. Nakon razmatranja, primjetli smo da bismo ovaj patern mogli primjeniti na klasu *Pravnik*. Shodno tome da različite profesije pravnika imaju drugačije uloge, korištenjem dekoracijskog paterna, omogućene bi bile različite uloge, čime bi se postigao viši nivo relevantnosti unutar rada Sistema.

4. Brigde pattern

Osnovna namjena Bridge paterna je da omogući odvajanje apstrakcije i implementacije neke klase tako da ta klasa može posjedovati više različitih apstrakcija i više različitih implementacija za pojedine apstrakcije. U našem sistemu konkretno ne možemo navesti primjer ovog paterna, međutim brigde patern može se iskoristiti nad klasama *Pravnik, Upravnik* (ukoliko bismo ih povezali sa intefejsom *posaljiObavijest*). Također trebamo dodati novu klasu *Bridge*, koja će sadržavati apstrakciju i kojoj će *Zatvor* jedino imati pristup

5. Composite pattern

Composite patern služi za kreiranje hijerarhije objekata. Koristi se kada svi objekti imaju različite implementacije nekih metoda, no potrebno im je svima pristupati na isti način, te se na taj način pojednostavljuje njihova implementacija. Primjena ovog paterna unutar sistema nije moguća, jer odstupa od generalnog uzorka po kojem bi trebala funkcionisati aplikacija.

6. Proxy pattern

Proxy patern služi za dodatno osiguravanje objekata od pogrešne ili zlonamjerne upotrebe. Primjenom ovog paterna omogućava se kontrola pristupa objektima, te se onemogućava manipulacija objektima ukoliko neki uslov nije ispunjen, odnosno ukoliko korisnik nema prava pristupa traženom objektu. U našem sistemu konkretno ne možemo navesti primjer ovog paterna, međutim možemo implementirati patern nad klasama *Pravnik, Upravnik* kao i intefejsom *posaljiObavijest*. Najprije definšemo klasu *Proxy* koja će implementirati metodu *pristup()* u okviru koje će biti izvršena autentifikacija, te koja će naslijediti interfejs *Obavijest* i njegove metode.

7. Flyweight pattern

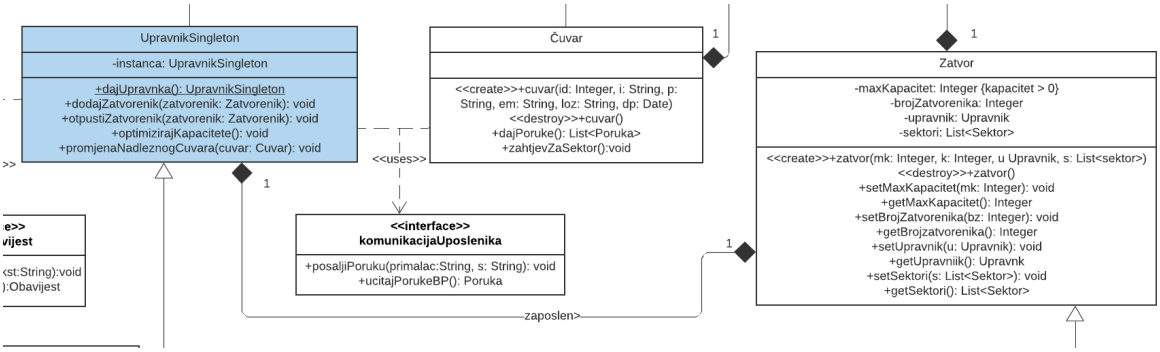
Flyweight patern koristi se kako bi se onemogućilo bespotrebno stvaranje velikog broja instanci objekata koji svi u suštini predstavljaju jedan objekat. Samo ukoliko postoji potreba za kreiranjem specifičnog objekta sa jedinstvenim karakteristikama (tzv. specifično stanje), vrši se njegova instantacija, dok se u svim ostalim slučajevima koristi postojeća opća instance

objekta (tzv. bezlično stanje). Primjena ovog paterna unutar sistema nije moguća, jer odstupa od generalnog uzorka po kojem bi trebala funkcionisati aplikacija.

Kreacijski patterni

1. Singleton pattern

Singleton patern služi kako bi se neka klasa mogla instancirati samo jednom. Na ovaj način može se omogućiti i tzv. *lazy initialization*, odnosno instantacija klase tek onda kada se to prvi put traži. Osim toga, osigurava se i globalni pristup jedinstvenoj instanci - svaki put kada joj se pokuša pristupiti, dobiti će se ista instanca klase. Ovo olakšava i kontrolu pristupa u slučaju kada je neophodno da postoji samo jedan objekat određenog tipa. Kako bi se osigurao singleton patern postoji samo jedna instanca upravnika kazneno-popravnog doma (klasa *Upravnik*).



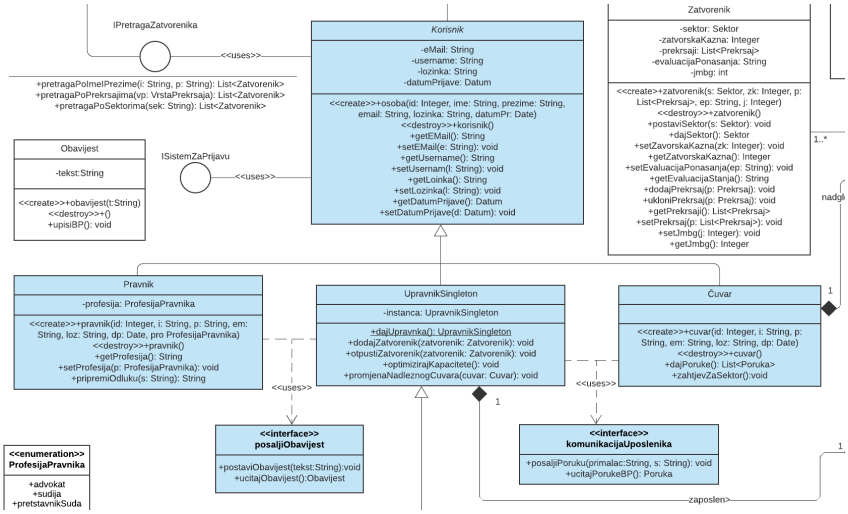
Slika 3. Primjer Singleton pattern primjenjen na class diagramu Tartatus aplikacije

2. Prototype pattern

Uloga Prototype paterna je da kreira nove objekte klonirajući jednu od postojećih prototip instanci (postojeći objekat). Ako je trošak kreiranja novog objekta velik i kreiranje objekta je resursno zahtjevno tada se vrši kloniranje već postojećeg objekata. U našem sistemu možemo iskoristiti prototype pattern prilikom dodavanja novih zatvorenika, pri čemu trebamo definisati interfejs *IPrototip* koji se sastoji od metode *kloniraj()*. Također neophodno je Naslijediti interfejs IPrototip od klase *Zatvorenik* te implementirati metodu *kloniraj()* koja će kreirati duboku kopiju objekta.

3. Factory Method pattern

Uloga Factory Method paterna je da omogući kreiranje objekata na način da podklase odluče koju klasu instancirati. Različite podklase mogu na različite načine implementirati interfejs. Ovaj patern je implementiran preko klasa Upravnik, Pravnik i Čuvar (klase nasljeđuju klasu *Korisnik*).



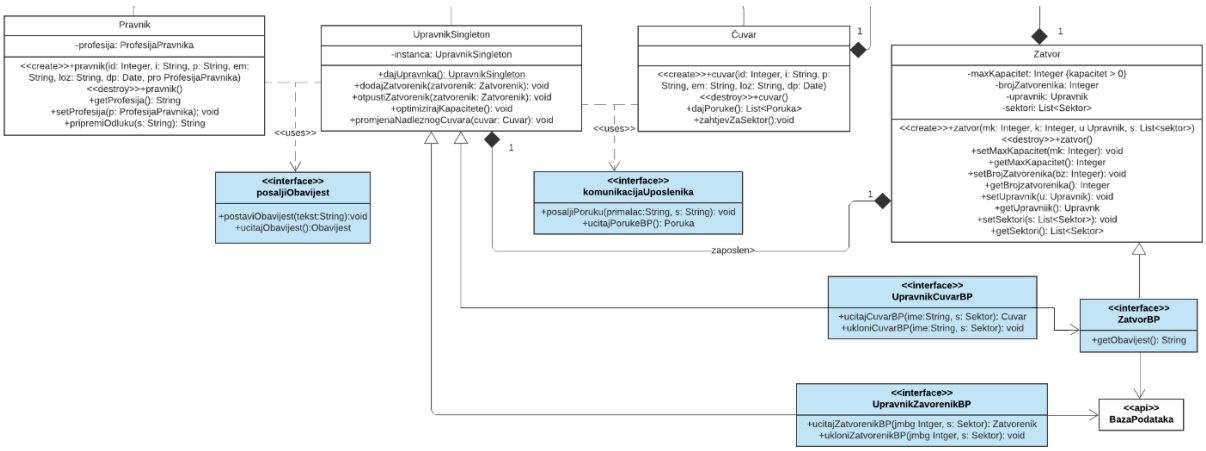
Slika 4. Primjer Factory Method pattern primjenjen na class diagramu Tartatus aplikacije

4. Abstract Factory Method 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. Primjenu ovog paterna moguće je zastupati u određenom nivou konkretizacije nad klasama, jer odstupa od generalnog uzorka po kojem bi trebala funkcionisati aplikacija.

5. Builder pattern

Uloga Builder paterna je odvajanje specifikacije kompleksnih objekata od njihove stvarne konstrukcije. Isti konstrukcijski proces može kreirati različite reprezentacije. Primjena ovog paterna evidentna je prilikom korištenja interfejsa na dijagramu, koji su vezani sa klasama unutar kojih se implementiraju.



Slika 5. Primjer Builder pattern primjenjen na class diagramu Tartatus aplikacije