



Strukturalni i kreacijski patterni

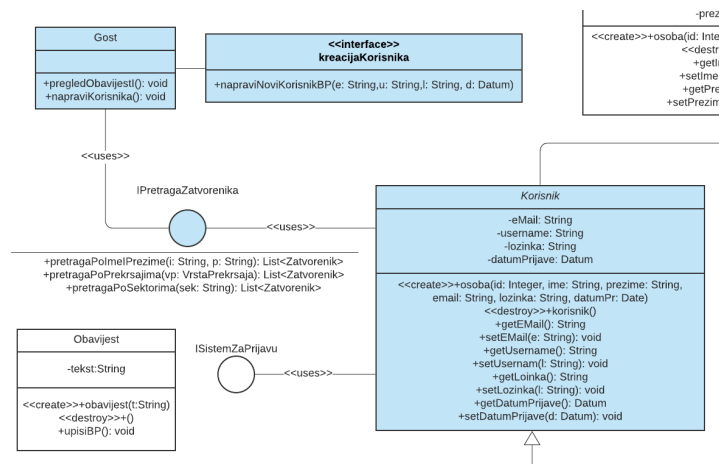
Predmet: Objektno orijentisana analiza i dizajn

Grupa: Tartarus
Hadžić Ajdin
Halilović Kemal
Mehmedović Faris

Strukuralni patterni

1. Adapter pattern

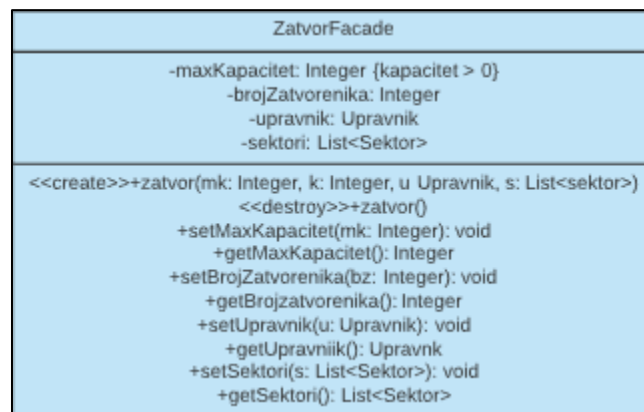
Osnovna namjena Adapter paterna je da omogući širu upotrebu već postojećih klasa. U situacijama kada je potreban drugačiji interfejs već postojeće klase, a ne želimo mijenjati postojeću klasu koristi se Adapter patern. Adapter patern kreira novu adapter klasu koja služi kao posrednik između originalne klase i željenog interfejsa. Primjer Adapter paterna se proicira unutar *Gost* klase koja omogućava da dva interfejsa (sa naizgleda različitim pravcima) funkcioniraju 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 koristiti s ciljem da osigura više pogleda visokog nivoa na podsisteme (implementacija podsistema skrivena od korisnika). Namjena facade paterna se može prepoznati 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 patterna 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 patterna. Nakon razmatranja, primjetili smo da bismo ovaj pattern mogli primjeniti na klasu *Pravnik*. Shodno tome da različite profesije pravnika imaju drugačije uloge, korištenjem dekoracijskog patterna, omogućene bi bile različite uloge, čime bi se postigao viši nivo relevantnosti unutar rada Sistema.

4. Bridge pattern

Osnovna namjena Bridge patterna 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 patterna, međutim bridge pattern 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 pattern služi za kreiranje hijerarhije objekata. Koristi se kada svi objekti imaju različite implementacije nekih metoda, no potrebno im je svima pristupiti na isti način, te se na taj način pojednostavljuje njihova implementacija. Primjena ovog patterna unutar sistema nije moguća, jer odstupa od generalnog uzorka po kojem bi trebala funkcionisati aplikacija.

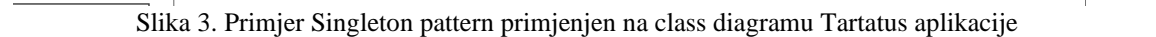
6. Proxy pattern

Proxy pattern služi za dodatno osiguravanje objekata od pogrešne ili zlonamjerne upotrebe. Primjenom ovog patterna 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 patterna, međutim možemo implementirati pattern nad klasama *Pravnik*, *Upravnik* kao i intefejsom *posaljiObavijest*. Najprije definišemo 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

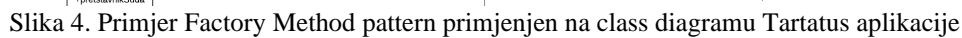
Flyweight pattern koristi se kako bi se onemogućilo bespotrebno stvaranje velikog broja instanci objekata koji svi u suštini predstavljaju jedan objekat. Samo ukoliko postoji potreba za kreiranjem specifičnog objekta sa jedinstvenim karakteristikama (tzv. specifično stanje), vrši se njegova instantacija,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

1. Singleton pattern



Uloga Prototype paterna je da kreira nove objekte klonirajući jednu od postojećih prototip

Uloga Factory Method paterna je da omogući kreiranje objekata na način da podklase odluče



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.

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.

