STRUKTURALNI PATERNI

Na samom početku ovog dokumenta nabrojati ćemo sve paterne i ukratko navesti njihove primjene.

ADAPTER PATERN - kreira novu adapter klasu koja služi kao posrednik između originalne klase i željenog interfejsa te se timedobija željena funkcionalnost bez izmjena na originalnoj klasi i bez ugrožavanja integriteta cijele aplikacije.

FASADNI PATERN - osigurava više pogleda visokog nivoa na podsisteme (implementacija podsistema skrivena od korisnika). Ovaj patern olakšava korištenje aplikacije za korisnike koji ne moraju znati na koji način je izvedena implementacija funkcionalnosti koje koriste.

DEKORACIJSKI PATERN – omogućava dinamičko dodavanje novih elemenata i ponašanja (funkcionalnosti) postojećim objektima. Dakle, ukoliko imamo objekte koji imaju istu osnovu i razlikuju se u samo nekim pojedinostima, umjesto da pravimo niz izvedenih klasa, upotrijebit ćemo Dekoracijski patern.

FLYWEIGHT PATERN – omogućava da se “ubaci” više objekata u slobodni prostor RAM-a tako što se dijele zajednički atributi između više objekata umjesto da se čuvaju svi podaci svakog objekta.

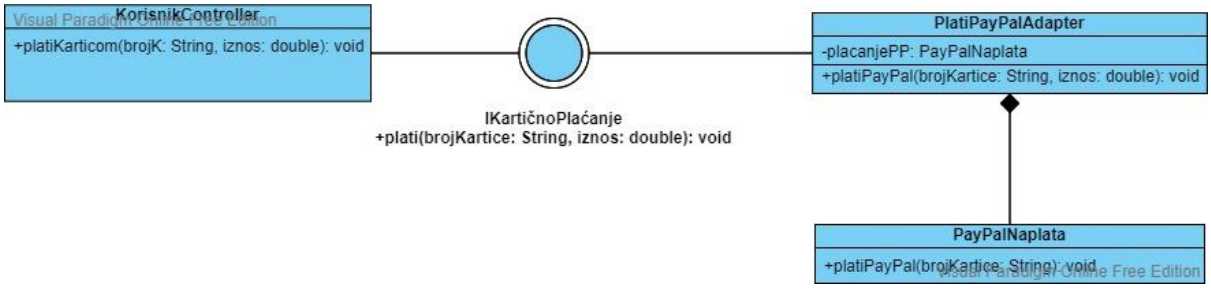
BRIDGE PATERN - odvaja kompleksnu klasu na dvije hijerarhije apstrakcija i implementacija tako da se mogu pojedinačne instance odvojeno implementirati u zavisnosti od njihove kompleksne strukture koja je “neovisna” od bazne klase. Ovaj patern je veoma važan jer omogućava ispunjavanje Open-Closed Solid principa, tj. nadogradnju novih potencijalnih klasa.

COMPOSITE PATERN - omogućava kompoziciju objekata strukture drveta što nudi mogućnost obrade podataka kao individualnih objekata. Koristi se kada svi objekti imaju različite implementacije nekih metoda, ali svakoj od njih se pristupa na isti način, pa Composite patern pojednostavljuje njihovu implementaciju.

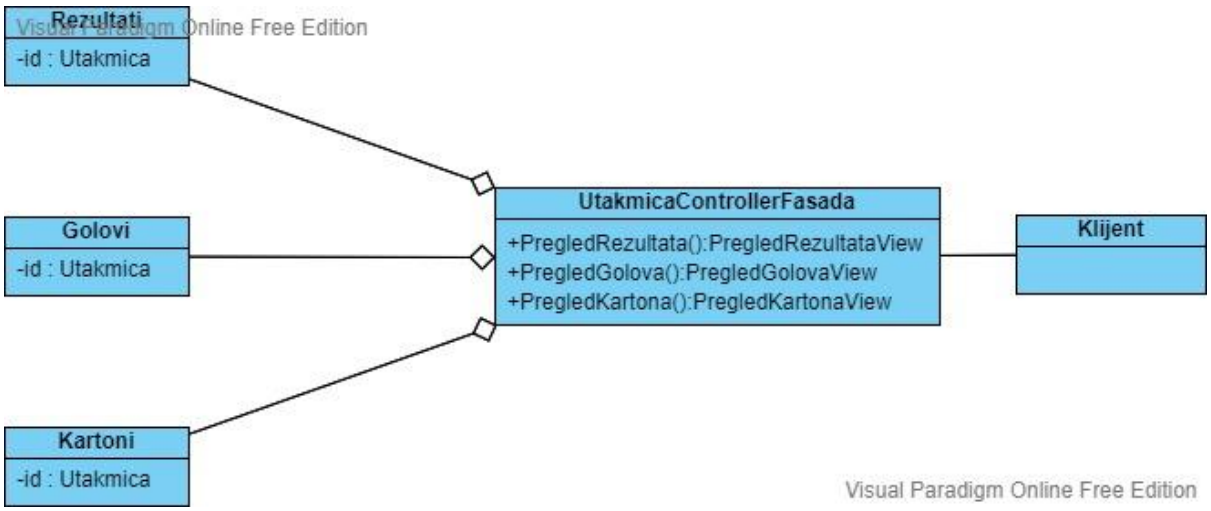
PROXY PATERN - predstavlja kontrolu pristupa originalnom objektu i koristi se ukoliko je potrebno dodatno osiguranje objekata od pogrešne i zlonamjerne upotrebe. Često se koristi i za klase koje imaju osjetljive podatke ili spore operacije.

Sada ćemo razmotriti svaki od navedenih paterna u našem sistemu.

Adapter patern – u našem sistemu jeste implementiran ovaj patern. U sistemu imamo implementiranu funkcionalnost plaćanja. Pored kartičnog plaćanja, klijentu je omogućeno i plaćanje putem PayPal-a sistema. To smo uradili pomoću adapter paterna tako što smo kreirali interfejs IKarticnoPlacanje koji će naslijediti adapter klasu, te klasu PayPalNaplata. Kreirat ćemo i PlatiPayPalAdapter klasu koja će izvršiti adaptiranje vrste plaćanja.



Fasade patern – u našem sistemu jeste implementiran ovaj patern. Primjer upotrebe ovog paterna su klase Klient i klasa UtakmicaControllerFasada, gdje klijent može izvršiti pregled rezultata, kartona, golova za klasu Utakmica bez ikakvih znanja o načinu implementacije klase Utakmica.



Decorator patern – u našem sistemu ovaj patern je implemetiran tako što smo dodali funkcionalnosti našem administratoru koji brine o izgledu naslovne stranice i ostalih interfejsa sistema. Naime, admin može manipulisati tekstom, slikom, mijenati boje i slično u cilju postizanja sto ljepšeg i estetičnijeg interfejsa.

Flyweight patern – u našem sistemu nije implementiran ovaj patern jer smo dizajnirali sistem tako da su korišteni samo podaci tj. atributi koji su osnovni i samim tim i potrebni u sistemu. Mogli bi iskoristiti ovaj patern pri radu sa slikama. Npr, ako administrator ne doda slike za određeni Sport ili Utakmicu, oni dobiju neku, unaprijed definisanu, defaultnu vrijednost.

Bridge patern – u našem sistemu nije iskorišten ovaj patern ali bismo mogli da realizujemo različit način prikazivanja ponuda za klađenje VIP i običnim korisnicima. Naime, ako želimo da se običnom korisniku prikazuje samo ponuda klase Lutrija te da je uz VIP članstvo moguće odabrati sve opcije klađenja (jer kod nas je moguće vidjeti sve pogodnosti lutrije s tim da su običnom korisniku klađenja zabranjena za odabir).

Composite patern – u našem sistemu nije korišten ovaj patern , bi ga mogli iskorititi ukoliko bismo u klasu Igrač i u klasu Tim dodali metodu dajUkupanBrojGolova(), ona bi se isto pozivala ali bi se njena implementacija razlikovala s obzirom da se u klasi Igrač računa za samo jednog igrača, a u klasi Tim se računa za cijelu listu Igrača.

Proxy patern – u našem sistemu nije iskorišten ovaj patern ali bi ga mogli iskorititi za zaštitu svih editovanja sistema. Naime, mogli bismo napravit „Editproxy“ koji će ograničiti korisnika tako što će za svaki edit koji korisnik pokuša a ne bi smio moći izvršiti, biti tražen password. Također, mogli bi implementirati klasu „PotvrdaProxy“ koja nasljedjuje interfejs sa metodom za log in koja prima lozinku i korisničko ime te odobrava ili odbija pristupanje korisnickom racunu ukoliko neki podatak nije tačan.