Univerzitet u Sarajevu
Elektrotehnički fakultet
Sarajevo

# Osmi projektni zadatak

Objektno orijentisana analiza i dizajn


Naziv grupe: Spotifive
Članovi grupe: Nadina Miralem 18937
Amina Hromić 19084
Nerma Kadrić 19030
Amila Kukić 19065
Una Hodžić 19044

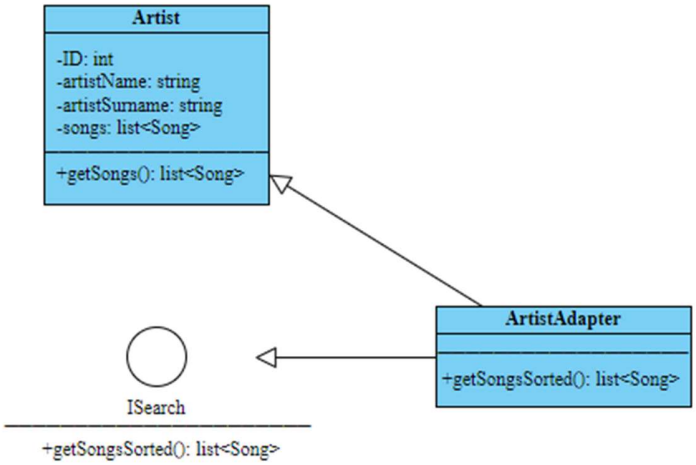
*Objektno orijentisana analiza i dizajn*
*Grupa Spotifive*
*Osmi projektni zadatak*

# Strukturalni paterni

U nastavku slijedi opis strukturalnih paterna koji će biti primijenjeni u sistemu.

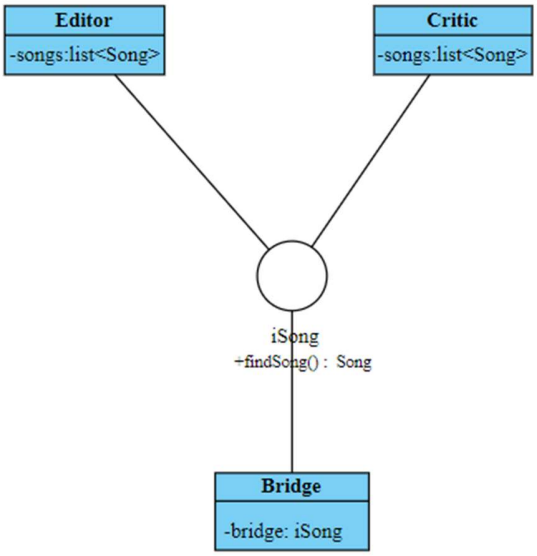
## *Adapter*

Da bismo korisnicima sistema omogućili pregled pjesama određenog izvođača sortiranih po nazivu, u tu svrhu možemo iskoristiti Adapter pattern. Realizacija ovog patterna je omogućena na sljedeći način: definisat ćemo interface ISearch sa metodom getSongsSorted() koja vraća listu pjesama izvođača sortiranih po nazivu. Zatim definišemo klasu ArtistAdapter koja će implementirati interface ISearch. Unutar metode getSongsSorted() pozivamo metodu getSongs() klase Artist. Na ovaj način nismo vršili nikakvu promjenu Artist klase ali smo unaprijedili njen interface.




## *Bridge*

Bridge pattern omogućava da se iste operacije primjenjuju nad različitim podklasama. Također se koristi kod izbjegavanja kreiranja novih metoda za postojeće funkcionalnosti. Ono što je karakteristično za ovaj pattern je to da te operacije imaju jedan zajednički korak, nakon čega su različite. Ako bi naš sistem zahtijevao traženje nekih pjesama, pronalazak pjesama bi izgledao ovako:




U nastavku slijedi opis ostalih strukturalnih patterna, te načina na koje bi se oni mogli implementirati u našem sistemu.

## *Proxy*

Ovaj pattern služi za dodatnu zaštitu objekata od pogrešne ili zlonamjerne upotrebe, odnosno onemogućava manipulaciju objektima ukoliko određeni uslov nije ispunjen. Kako u našem sistemu recenziju pjesme mogu ostavljati samo korisnici kritičari, odličan način da osiguramo taj proces je upravo upotreba Proxy paterna. Ukoliko je korisnik registrovan kao korisnik kritičar, moći će izvršiti akciju komentarisanja i ocjenjivanja pjesme, a ako nije onda bi mu se ponudila opcija da se ponovo registruje, ali kao kritičar.

## *Composite*

Composite pattern je pogodan u slučaju kada je potrebno obrisati određenog korisnika kritičara iz sistema. Kako korisnik kritičar ima mogućnost ostavljanja recenzija na pjesme, kada njegov profil bude obrisan, trebale bi biti obrisane i sve recenzije koje je on napisao. Klasa koja je direktno povezana sa ovim patternom je Administrator klasa, s obzirom na to da je administrator jedini akter koji ima mogućnost brisanja korisničkih računa.

## *Decorator*

Kako korisnici našeg sistema imaju mogućnost dodavanja profilne slike, Decorator pattern bi mogao biti iskorišten kako bi se te profilne slike uredile pri postavljanju. Korisnik bi prilikom postavljanja profilne slike imao mogućnost rotiranja ili rezanja slike, te dodavanja različitih filtera.

## *Flyweight*

Pomoću ovog patterna više različitih objekata može koristiti isto glavno stanje. Primjer korištenja ovog patterna u našem sistemu bio bi također vezan za postavljanje profilne slike korisnika. Za sve korisnike koji ne žele postaviti profilnu sliku bila bi iskorištena unaprijed zadana slika. Dakle, da bi korisnici mogli zadržati tu default-nu sliku, bilo bi potrebno implementirati Flyweight pattern kako bi svi ti korisnici koristili jedan, zajednički resurs.

## *Facade*

Ovaj pattern se koristi kada je potrebno "zamaskirati" komplikovani sistem, odnosno kada koristimo samo mali dio funkcionalnosti kompleksnog sistema. Kako je naš sistem vrlo jednostavan, ovaj pattern nećemo implementirati. Kada bismo u sistemu imali neku klasu koja vrši vrlo kompleksan proces, tada bismo implementirali pomoćne interface-e koje bismo smjestili u klasu Facade.