Univerzitet u Sarajevu
Elektrotehnički fakultet u Sarajevu
Odsjek za računarstvo i informatiku

Strukturalni paterni

Objektno orijentisana analiza i dizajn

Članovi tima:
Nađa Alijagić
Hana Biševac
Anida Nezović

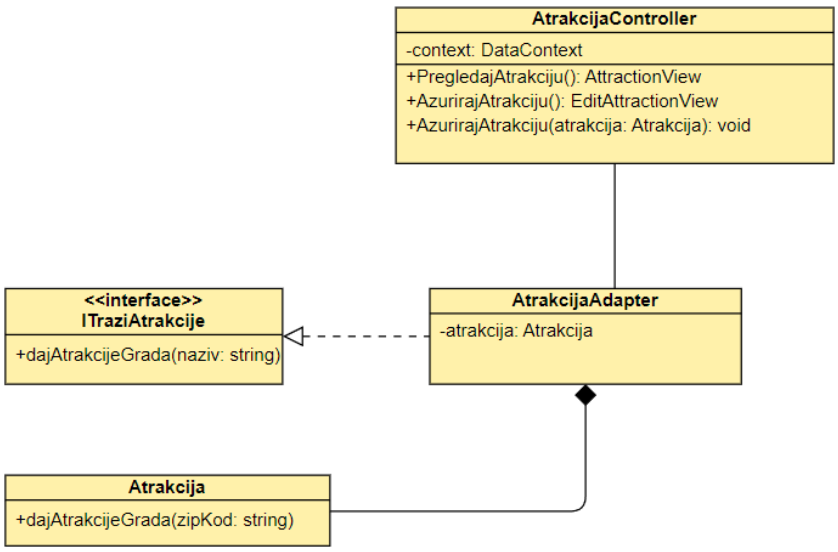
1. Adapter patern

Adapter patern koristi se u situacijama kada je potreban drugačiji interfejs već postojeće klase, a ne želimo mijenjati postojeću klasu. U našem slučaju, ovaj patern bismo mogli iskoristiti za jednu novu funkcionalnost zaposlenika. Naime, zaposelnik može tražiti od sistema da mu da spisak svih atrakcija u nekom gradu. U klasi Lokacija bismo dodali atribut zipKodGrada. Sada želimo da zaposlenik dobije listu atrakcija kada pošalje zip kod grada, umjesto naziva grada. Kako smo očekivali pretragu samo na osnovu naziva, trebat će nam adapter, koji će implementirati novi zahtijevani interfejs.

Druga ideja

Kada bi u budućnosti, odlučili dati neke pogodnosti turistima koji ponovo dolaze u BiH, mogli bismo kreirati klasu *RegistrovaniKorisnikAdapter* i interfejs *IPonovniDolazak*. *RegistrovaniKorisnikAdapter* bi sadržavala te dodatne mogućnosti, koje bismo implementirali bez ikakvih promjena u klasi *RegistrovaniKorisnik*.

Implementirana je prva ideja.

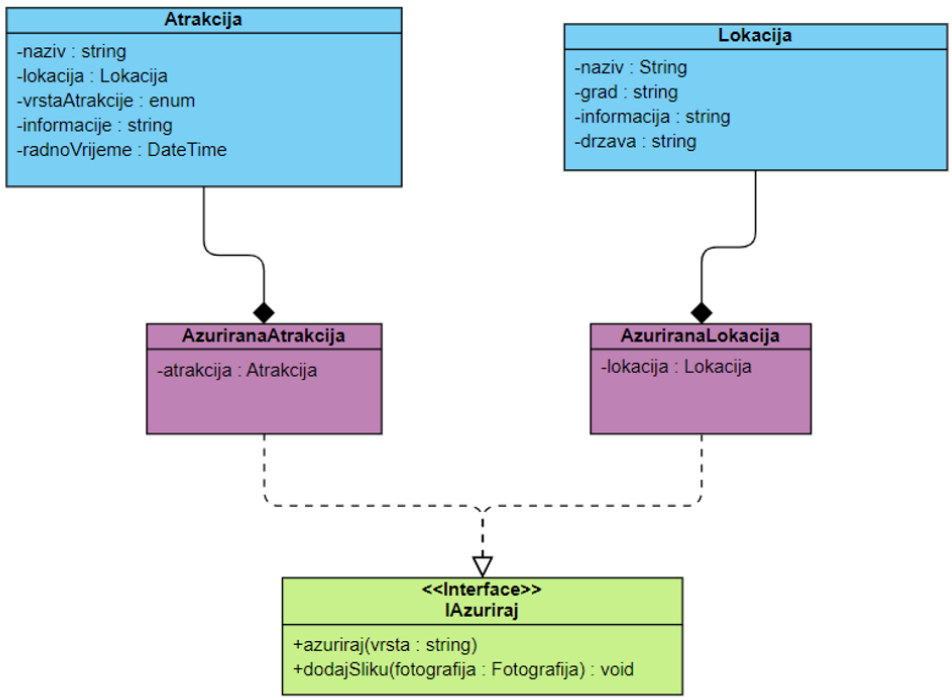


2. Fasadni patern

Fasadni patern ima za osnovni cilj da korisnicima pojednostavi korištenje kompleksnog sistema. Dakle, u slučaju da sistem ima više podsistema sa povezanim implementacijama i apstrakcijama poželjno je korštenje ovog patterna. S obzirom da je naš sistem poprilično jednostavan i jasan, nemamo potrebu za implementaciju ovog patterna. Ovaj patern bismo mogli primijeniti u slučaju kada bi naša aplikacija postala složenija, naprimjer, uključivala plaćanje karata putem naše stranice. Umjesto da se koristi veliki broj kontrolera prilikom implementacije, npr. *KorisnikKontoler* sa metodom *dajKarticuKorisnika*, *KarticaKontoler* sa metodom *skiniNovacSaRačuna* i *provjeriStanjeRačuna*, *AtrakcijaKontoler* sa metodom *rezervišiKartu*, itd. mogli bismo sve objediniti u zajednički kontroler npr. *PlaćanjeKontoler* gdje bismo sve naše metode vezane za plaćanje i informacije potrebne za plaćanje bile na jednom mjestu.

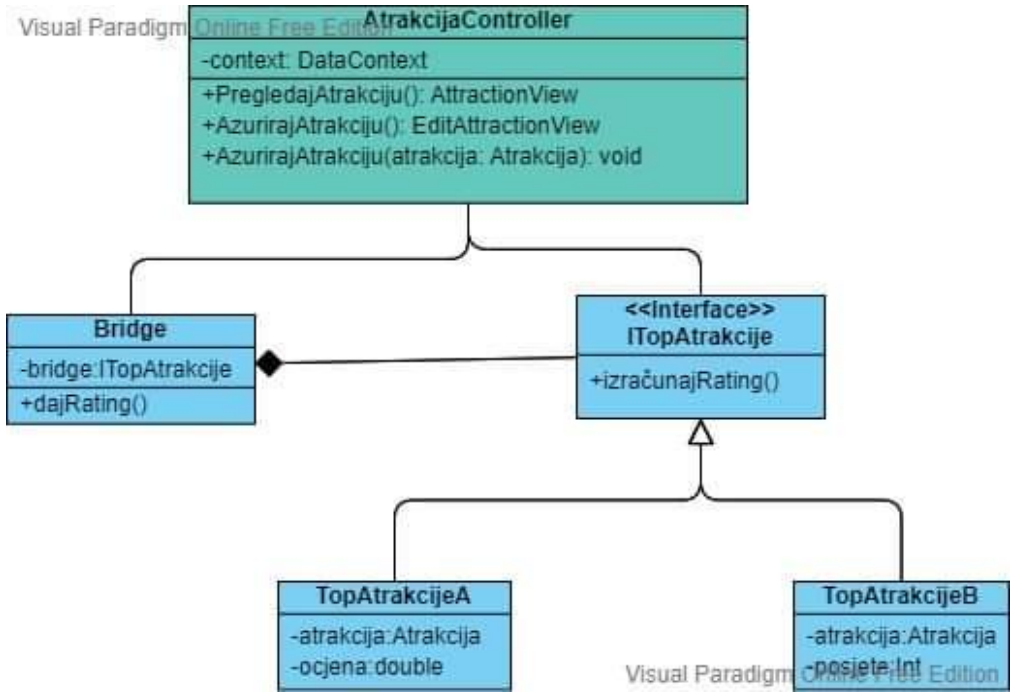
3. Dekorater patern

Osnovna namjena 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. Ovaj patern možemo iskoristiti kod ažuriranja atrakcija i lokacija koje vrši zaposlenik. Kreirat ćemo interfejs *IAžuriraj* koji će imati sljedeće operacije *ažuriraj(vrsta: string)* i *dodajSliku(slika : Fotografija)*. Ovaj interfejs će nasljeđivati klasa *Atrakcija* i klasa *Lokacija*.



4. Bridge patern

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. Bridge patern pogodan je kada se implementira nova verzija softvera a postojeća mora ostati u funkciji. Na samom početku kreiranja ideje o našem sistemu, postojala je verzija sa funkcionalnosti chata sa botom. Međutim, mislile smo da je previše kompleksna i nismo je zadržale. Da je ta ideja zaživila, bez problema bismo mogle iskoristiti Bridge patern u njenoj implementaciji. Postoji još jedan način na koji bismo mogle primjeniti Bridge patern, a to je prilikom *Top atrakcija*. Registrovani korisnici bi bili u mogućnosti da svakoj posjećenoj atrakciji dodijele ocjenu, na osnovu koje bi se kreirala lista sa najbolje ocjenjenim atrakcijama. Također, s obzriom da korisnici mogu označiti koje su lokacije i atrakcije pojsetili, pomoću tih informacija sistem može kreirati listu top posjećenih atrakcija. Sada imamo dvije kategorije top atrakcija: top atrakcija po ocjenama i top atrakcija po posjećenosti. Ukoliko bismo željeli na početnoj stranici sistema prikazati top atrakcije, tada bismo mogli imati klase *TopAtrakcijeA* i *TopAtrakcijeB* koje bi implementirale interfejs koji će formirati listu za iste radove, ali svaka prema svom kriteriju.

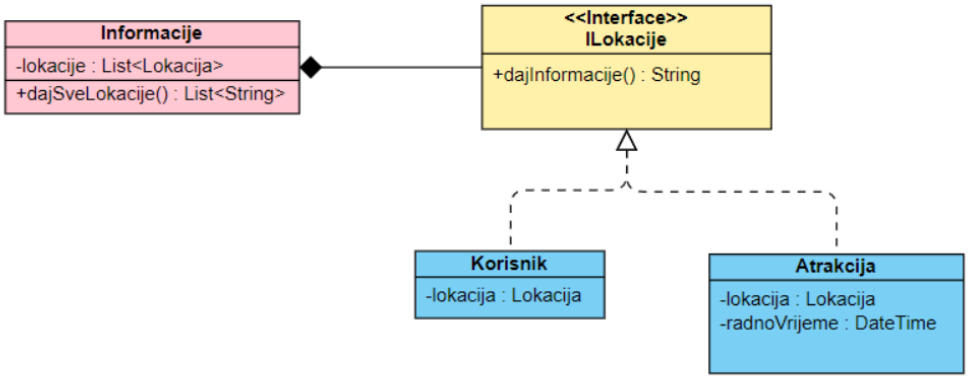


5. Proxy patern

Namjena Proxy paterna je da omogući pristup i kontrolu pristupa stvarnim objektima. U našem sistemu, zaposlenik često pristupa i mijenja podatke o atrakcijama. Kako je autentičnost tih podataka jako bitna, trebamo sa sigurnošću znati da sve izmjene vrši zaposlenik. U tome nam može pomoći ovaj patern. Kreiranjem Proxy klase koja ima metodu *pristup(zaposlenik: Zaposlenik, password: string)*, čuvamo objekte tipa klase *Atrakcija* od eventualnih slučajnih i ilegalnih pristupa.

6. Kompozitni patern

Osnovna namjena ovog paterna je da omogući formiranje strukture stabla pomoću klasa, u kojoj se individualni objekti i kompozicije individualnih objekata jednako tretiranju. U našem sistemu bismo mogli ovaj patern iskoristiti za prikaz svih lokacija koje se pojavljuju u sistemu. Kod korisnika bismo tražili lokaciju njegovog prebivališta, dok bi nas kod atrakcija zanimalo i radno vrijeme.



7. Flyweight patern

Postoje situacije u kojima je potrebno da se omogući razlikovanje dijela klase koji je uvijek isti za sve određene objekte te klase od dijela klase koji nije uvijek isti za sve određene objekte te klase. Osnovna namjena Flyweight paterna je upravo da se omogući da više različitih objekata dijele isto glavno stanje, a imaju različito sporedno stanje. Na našem projektu bismo ovo mogli primjeniti nad turama. Postojale bi dvije vrste tura: stalne ture koje se dešavaju redovno, naprimjer svaki dan i specijalne ture koje su jedinstvena ponuda koja će se desiti u tom trenutku i možda više nikada. Kako bismo realizovale ovaj pattern moramo imati klasu FlyweightTure koja

će u sebi imati atribut tipa ITure. ITure bi bio naš Flyweight interface iz kojeg bi proizilazile dvije klase: TureSpecial i TureRegular. Na fotografiji ispod je prikazan način implementacija. Ono što se na fotografiji ne vidi jeste da je Flyweight povezan sa VodicController s obzirmo da je vodič onaj koji kreira ture.