Univerzitet u Sarajevu
Elektrotehnički fakultet Sarajevo

Strukturalni patterni

OOAD 2020-2021

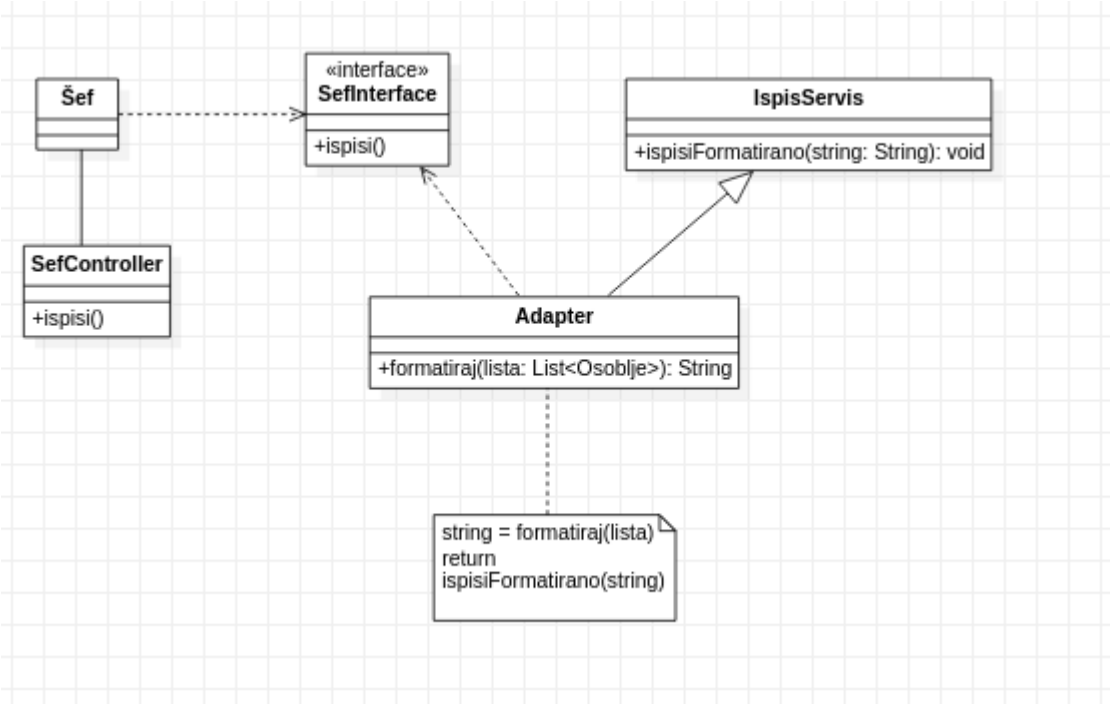
Naziv grupe: Hilbert’s Grand Hotel

Članovi: Nedim Badžak
Harun Alagić
Emil Fejzagić

1. Adapter

Adapter pattern služi da se postojeći objekat prilagodi za korištenje na neki novi način u odnosu na postojeći rad, bez mijenjanja same definicije objekta. Na taj način obezbjeđuje se da će se objekti i dalje moći upotrebljavati na način kako su se dosad upotrebljavali, a u isto vrijeme će se omogućiti njihovo prilagođavanje novim uslovima.

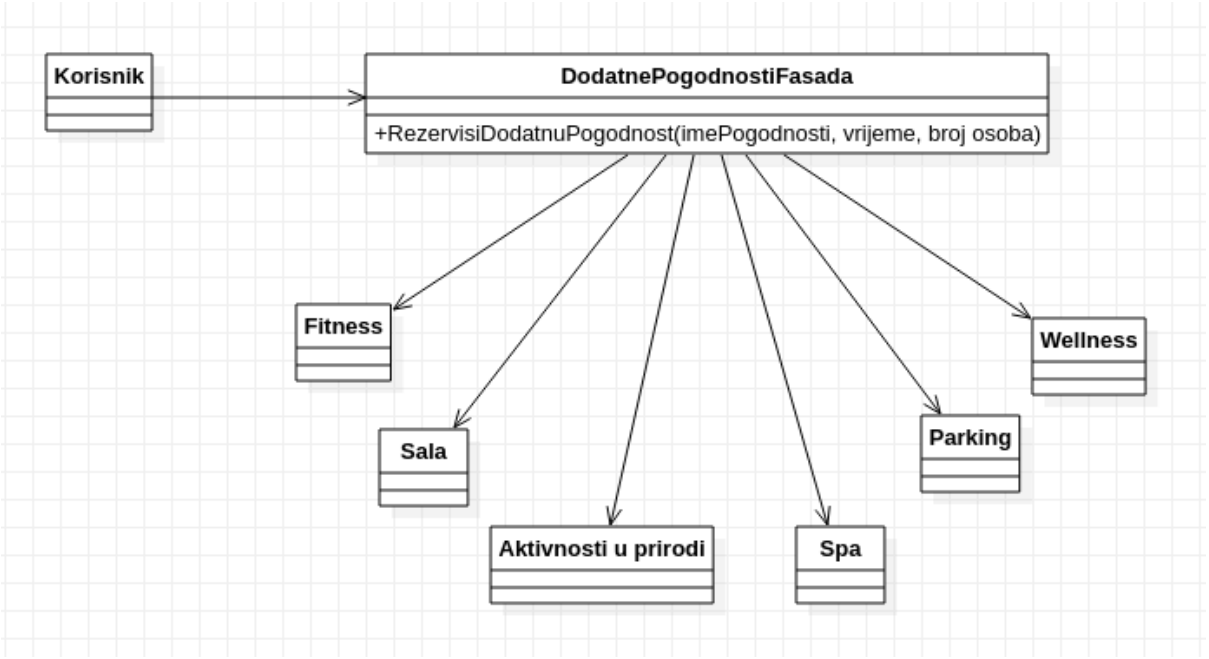
U klasi “Šef” imamo samo gettere i settere za listu listaOsoblja, mogli bismo napraviti adapter koji bi nam umjesto liste vraćao uređen prikaz osoblja (u vidu neke kolone kao jednog člana osoblja ili nešto sl). S obzirom da je bolje onemogućiti pozivanje get metode za obične liste za potrebe ispisa, tu bi nam pomogao Class Adapter pattern.



2. Facade

Fasadni pattern služi kako bi se klijentima pojednostavilo korištenje kompleksnih sistema. Klijenti vide samo fasadu, odnosno krajnji izgled objekta, dok je njegova unutrašnja struktura skrivena. Na ovaj način smanjuje se mogućnost pojavljivanja grešaka jer klijenti ne moraju dobro poznavati sistem kako bi ga mogli koristiti.

Smatramo da je naš sistem jednostavan i da nema potrebe za uvođenjem facade strukturalnog patterna. Ako bi morali na kraju primijeniti ovaj pattern, to bi vjerovatno učinili za neke još neimplementirane funkcionalnosti aplikacije koje bi mogle biti kompleksne i zahtijevati pojednostavljenje pomoću ovog strukturalnog patterna. Primjer koji ćemo na ilustraciji prikazati je za rezervaciju dodatnih pogodnosti čija bi implementacija bila jako kompleksna da nismo koristili Facade pattern koji radi teži i neuredniji dio posla, dok je “Korisnik”-u olakšan pristup.



3. Flyweight

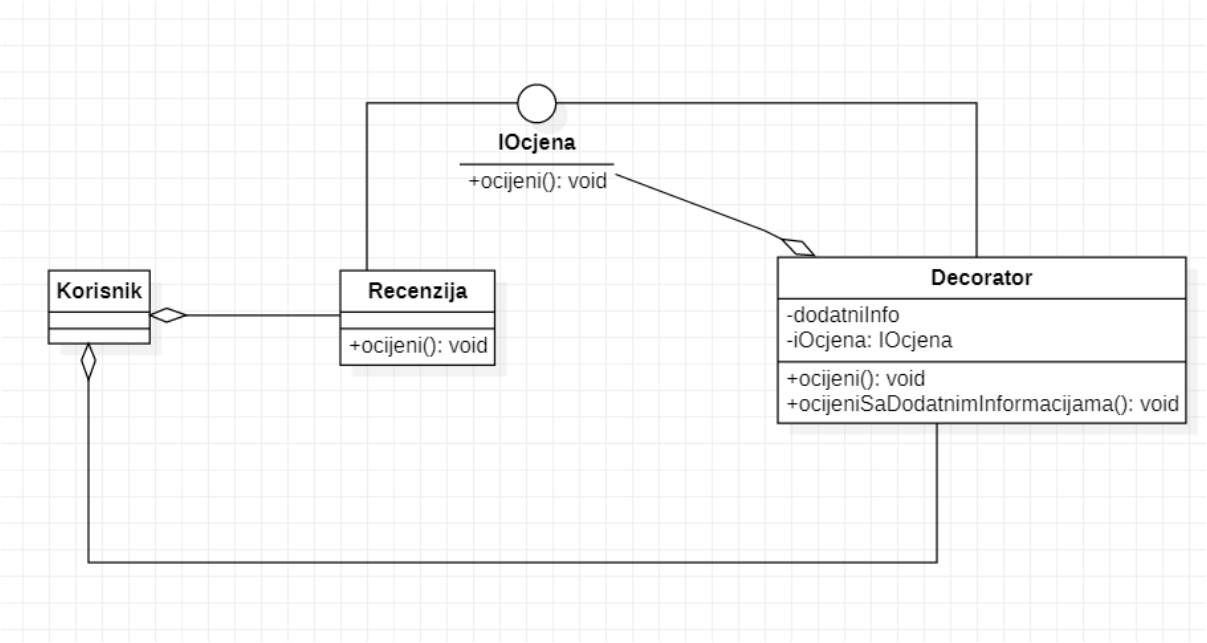
Flyweight pattern koristi se kako bi se onemogućilo bespotrebno stvaranje velikog broja instanci objekata koji svi u suštini predstavljaju jedan objekat. Samo ukoliko postoji potreba za kreiranjem specifičnog objekta sa jedinstvenim karakteristikama (tzv. specifično stanje), vrši se njegova instantacija,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

Pošto se ovaj strukturalni pattern koristi u slučaju kada model klase imaju osobinu koja se naknadno postavlja, a ujedno je neobavezna za rad same aplikacije, a mi smo dizajnirali naš sistem tako da koristimo samo podatke (atribute) koji su osnovni i neophodni u aplikaciji, nemamo potrebu za ovim patternom. Jedino gdje bi mogli iskoristiti ovaj pattern je na prikazu slika u pregledu Sobe, u slučaju da administrator ne doda slike ili ne postoje još korisnički dodane slike za tu sobu. U tom slučaju bi se prikazala neka defaultna vrijednost tj slika.

4. Decorator

Decorator pattern služi za omogućavanja različitih nadogradnji objektima koji svi u osnovi predstavljaju jednu vrstu objekta (odnosno, koji imaju istu osnovu). Umjesto da se definiše veliki broj izvedenih klasa, dovoljno je omogućiti različito dekoriranje objekata (tj. dodavanje različitih detalja), te se na taj način pojednostavljuje i rukovanje objektima i klijentima, te samo implementiranje modela objekata.

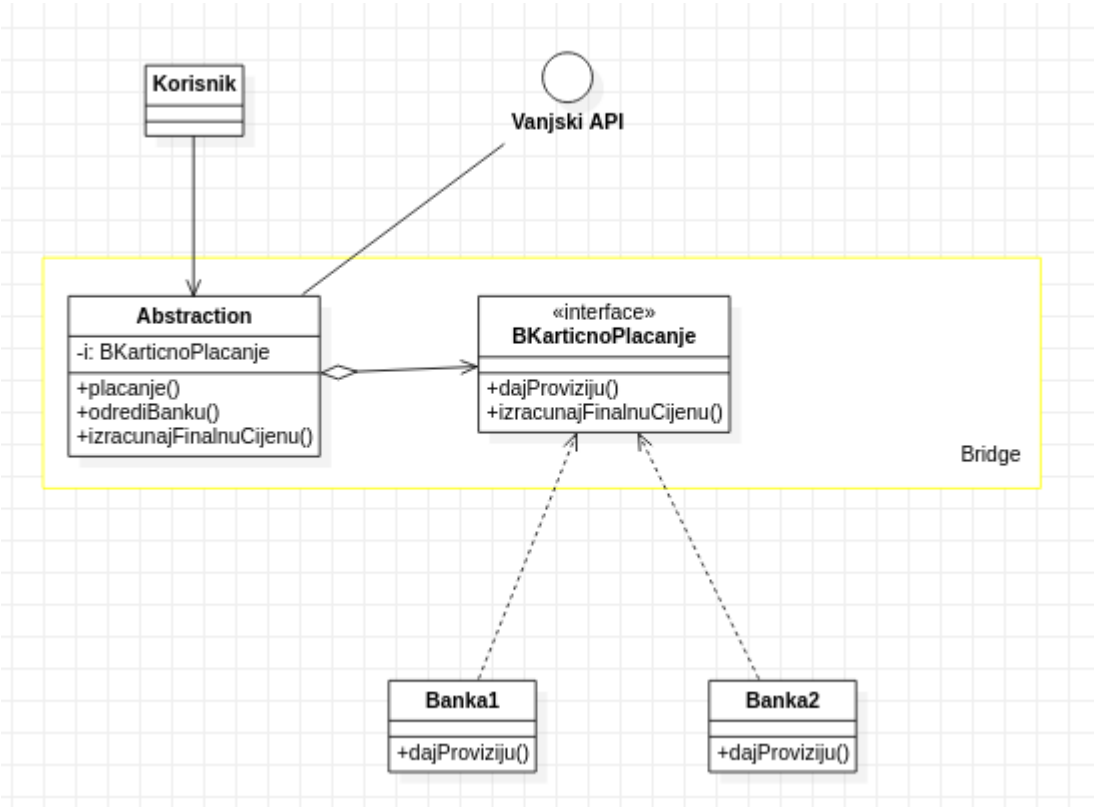
Ovaj strukturalni pattern bi se dao iskoristiti kod modula za davanje recenzija i ocjena iz razloga što omogućava višestruku iskoristivost pojedinačnih elemenata za stvaranje neke veće funkcionalnosti, pa tako kod nas postoji modul za recenziju koja je anonimna, sa slikom, anonimna sa slikom itd, te bi nam ovaj pattern uveliko pomogao da ne moramo praviti sve pojedinačne verzije.



5. Bridge

Bridge pattern služi kako bi se apstrakcija nekog objekta odvojila od njegove implementacije. Ovaj pattern veoma je važan jer omogućava ispunjavanje Open-Closed SOLID principa, odnosno uz poštovanje ovog patterna omogućava se nadogradnja modela klasa u budućnosti te osigurava da se neće morati vršiti određene promjene u postojećim klasama.

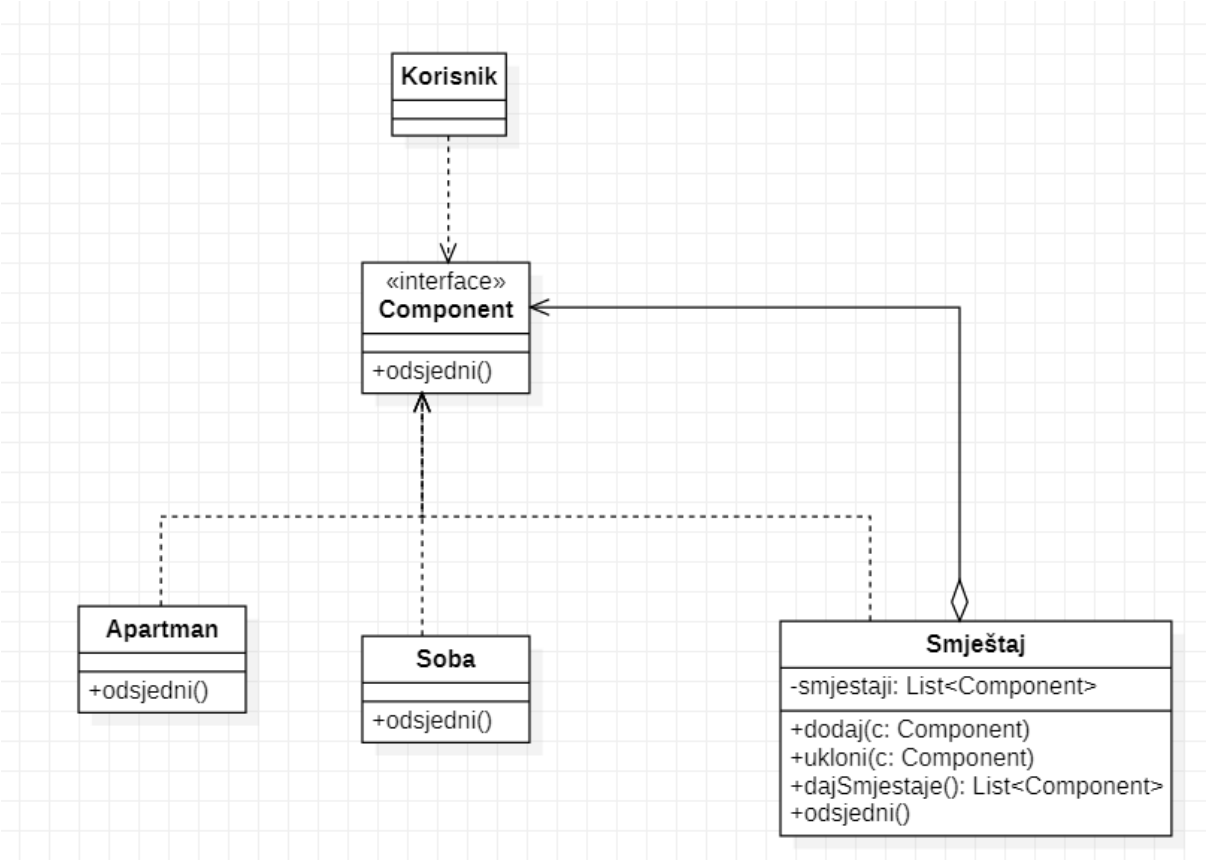
U našem sistemu postoji mogućnost plaćanja karticom.
Budući da korisnici ne koriste iste banke i iste tipove kartica, a znamo da u svakoj banci uplaćivanje na račun i skidanje novca sa računa snosi različite naknade i provizije, ovaj pattern bi omogućio da svi korisnici mogu vršiti plaćanje pomoću svojih banaka tako što bi napravili interface “BKarticnoPlacanje“ koji ima različite implementacije, u zavisnosti od banke (uz neki vanjski API koji omogućava prepoznavanje o kojoj se banci radi).



6. Composite

Composite pattern 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 vrši pojednostavljenje njihove implementacije.

Mjesto na kojem bi se ovaj pattern mogao iskoristiti je skup različitih vrsta smještaja koje bi hotel potencijalno mogao nuditi. Kao primjer smo uzeli apartmane i sobe (leafove) koji su više vrsta smještaja (composite). S obzirom da svaki composite sadrži dodatne composite-e ili leafove, a ne zna konkretne klase svoje “djece” tj smještaja, radi sa svim primjercima djece pomoću component interface-a. Nakon što “Smještaj” primi zahtjev, on delegira zadatak djeci koji izvršavaju potrebne zadatke i dalje Smještaj parsira rezultate operacija i vraća finalni rezultat “Korisnik” klasi.



7. Proxy

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.

Ovaj pattern bi se najoptimalnije mogao iskoristiti za kontrolu pristupa administratorskom control panelu, recepcionerovom control panelu itd, kao i za mogućnost rezervacije u zavisnosti od toga da li je korisnik prijavljen ili ne. Način implementacije ovoga bi mogao biti neki interface ili neka klasa “ProxyAuth” koja nasljeđuje interface za login i na taj način kontrolisao da li je dopušteno prikazivanje određenog sadržaja korisniku ili ne.