STRUKTURALNI PATERNI

Strukturalni paterni se bave kompozicijom i predstavljaju načine za definisanje odnosa među objektima. Strukturalni paterni daju mogućnost ukoliko je potrebna promjena u jednom dijelu sistema da se ne mora ostatak sistema mijenjati. Grupu strukturalnih paterna sačinjava sljedećih 7 paterna koji slijede u nastavku:

1. Adapter pattern

Osnovna namjena Adapter paterna je da omogući širu upotrebu već postojećih klasa. Koristi se u situacijama kada je potreban drugačiji interfejs već postojeće klase, a ne želimo mijenjati postojeću klasu. Njegova uloga je da kreira novu adapter klasu koja služi kao posrednik između originalne klase i interfejsa, te tako dobijamo željenu funkcionalnost bez izmjena na originalnoj klasi i bez ugrožavanja integriteta cijele aplikacije.

Primjena Adapter paterna u sistemu:

Adapter patern bi u našem sistemu mogli iskoristiti za slike. U sistemu trenutni tip atributa Slika je string, navedeni tip bi mogli zamijeniti uz pomoć metoda konverzije Adapter klase i omogućiti prihvatanje raznih tipova formata slika što neće ugroziti integritet cijele aplikacije.

2. Decorator pattern

Osnovna namjena Decorator paterna je da omogući dinamičko dodavanje novih elemenata i funkcionalnosti postojećim objektima. Objekat pri tome ne zna da je urađena dekoracija što je veoma korisno za ponovnu upotrebu komponenti softverskog sistema.

Primjena Decorator paterna u sistemu:

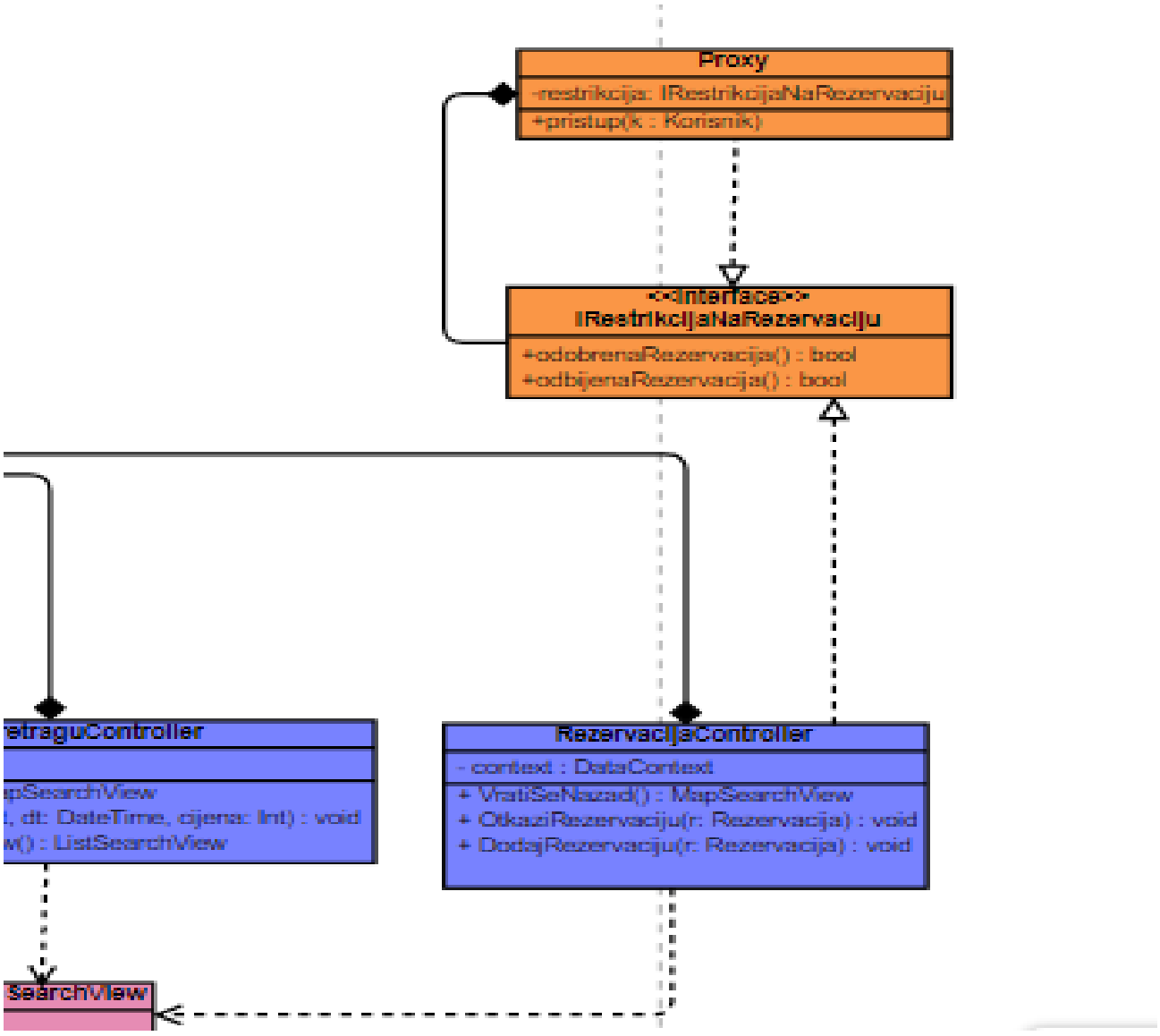
Decorater patern bi u našem sistemu mogli iskoristiti također za slike. Time bi dodali mogućnost uređivanja slika kao što je promjena veličine, rezanje, rotacija, povezivanje, izoštravanje.. U sistem bi dodali interfejs ISlika koji bi imao metode za uređivanje slike. Pošto su slike jedan od najbitnijih komponenti nekog oglasa samim tim bi trebale biti i omogućene neke aktivnosti nad slikama kako bi se osigurao kvalitetan prikaz korisnicima sistema.

3. Proxy pattern (primijenjen u sistemu)

Osnovna namjena Proxy paterna je da omogući pristup i kontrolu pristupa stvarnim objektima. Omogućava se kontrola pristupa objektima, te se i onemogućava manipulacija objektima ukoliko neki uslov nije ispunjen ili ukoliko korisnik nema prava pristupa traženom objektu. Proxy patern rješava probleme kada se objekt ne može instancirati direktno (npr. zbog restrikcije pristupa).

Primjena Proxy paterna u sistemu:

Kako je Proxy pattern vezan za kontrolu pristupa, mi smo na našem dijagramu dodali klasu Proxy koja implementira interfejs IRestrikcijaNaRezervaciju. Zamislili smo da svaki korisnik ima opciju da rezerviše smještaj, međutim potrebno je da ograničimo mogućnost Gost korisnicima, jer oni nemaju mogućnost rezervacije. Tako da ćemo putem metode pristup iz Proxy klase moći da zaključimo da li je korisnik Gost ili nije. Ukoliko je Gost njemu neće biti dozvoljeno da rezerviše smještaj (imat će permisiju), dok će Registrovanim i Premium korisnicima to biti dozvoljeno.



4. Composite pattern

Osnovna namjena Composite paterna je da omogući formiranje strukture stabla pomoću klasa, u kojoj se individualni objekti i kompozicije individualnih objekata jednako tretiraju. Koristi se za kreiranje hijerahije objekata, tačnije kada svi objekti imaju različite implementacije nekih metoda kojima je potrebno pristupati na isti način.

Primjena Composite paterna u sistemu:

U sistemu imamo dvije model klase pod imenom Rezervacija i Smještaj i te klase imaju metodu pod nazivom DodajSobu. Metoda treba da ima različite implementacije za te dvije klase, iako se isto zove. U rezervaciji smo zamislili da putem te metode dodamo sobu koju smo rezervisali u listu rezervisanih soba, dok smo u klasi smještaj zamislili da putem te metode dodamo potpuno novu sobu koja je omogućena u tom smještaju od nekog trenutka. Stoga ćemo pozivati tu metodu nad primjercima različitih klasa ali je implementacija u potpunosti drugačija.

5. Facade pattern

Facade patern se koristi kada sistem ima više identificiranih podsistema pri čemu su apstrakcije i implementacije podsistema usko povezane. Osnovna namjena Facade paterna je da osigura više pogleda visokog nivoa na podsisteme. Operacije koje su potrebne određenoj korisničkoj perspektivi mogu biti sastavljene od različitih dijelova podsistema. Može se više fasada postaviti oko postojećeg skupa podsistema i na taj način formirati više prilagođenih pogleda na sistem. Facade patern služi kako bi se klijentima pojednostavilo korištenje kompleksnih sistema. Klijenti vide samo krajnji izgled objekta, dok je njegova unutrašnja struktura skrivena. Na ovaj način se smanjuje mogućnost pojavljivanja grešaka jer klijenti ne moraju dobro poznavati sistem da bi ga koristili.

Primjena Facade paterna u sistemu:

Ovaj pattern ne bismo mogli upotrijebiti u našem sistemu iz razloga što nemamo više podsistema koje bi na neki način mogli instancirati u nekoj novoj klasi Fasada koja bi imala neke smislene metode. Kada bismo imali tu klasu Fasada, korisnik bi mogao da koristi smislene metode samo primijenjene na različitim primjercima nekih drugih klasa, recimo u nekom drugom sistemu bi imala metode IscrtajLiniju, IsctajKrug, ObojiLiniju, ObojiKrug i slično. Mi ne možemo da kreiramo takvu klasu u našem sistemu jer kada bismo je keirali imala bi skup nesmislenih metoda koje bi logički bolje išle u drugim klasama nego u jednoj zajedničkoj.

6. Flyweight pattern

Flyweight patern se koristi kako bi se onemogućilo bespotrebno stvaranje velikog broja instanci objekata koji svi u suštini predstavljaju jedan objekat. Osnovna namjena Flyweight paterna je upravo da se omogući da više različitih objekata dijele isto glavno stanje, a imaju različito sporedno stanje. Samo ukoliko postoji potreba za kreiranjem specifičnog objekta sa jedinstvenim karakteristikama, vrši se njegova instantacija, dok se u svim ostalim slučajevima koristi postojeća opća instance objekta.

Primjena Flyweight paterna u sistemu:

Flyweight patern bi u našem sistemu mogli iskoristiti za prijavljivanje gosta. Time da ukoliko se gost prijavljuje više puta, umjesto instanciranja više objekata tog tipa, bit će instanciran jedan objekat tog tipa.

7. Bridge pattern (primijenjen u sistemu)

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 je pogodan kada se implementira nova verzija softvera, a postojeća mora ostati u funkciji.

Primjena Bridge paterna u sistemu:

Bridge patern bi u našem sistemu mogli iskoristiti za obračunavanje popusta prilikom rezervacija smještaja na više dana ili u slučaju posebnih sezonskih popusta. Recimo da imamo registrovanog korisnika i premium korisnika. Te klase će implementirati interfejs IPopust, a interfejs IPopust će biti u kompoziciji sa novododanom klasom Bridge. U interfejsu IPopust ćemo imati metodu ObracunajCijenu koja prima koeficijent koji određuje popust. Za sve registrovane korisnike taj koeficijent je 0, a za premium korisnike će biti veći od 0, što znači da je u implementaciji ove metode zajednički dio sama cijena koju korisnik mora platiti, samo što će se razlikovati dio koji će kod premium korisnika u obračunavanju cijene dodati i popust, odnosno izračunati cijenu kao cijena - koeficijent*cijena.
Primijenili smo Bridge patern iz razloga što za dvije instance ovih klasa imamo istu metodu koja ima jedan isti dio, a drugi dio se razlikuje.

Ovako je primijenjen na projektu: