UPOTREBA SOLID PRINCIPA U PROJEKTU 'HOTEL'

SOLID je akronim za 5 važnih načela dizajna pri izvođenju objektno orijentisanog programiranja. U nastavku će biti objašnjena primjena SOLID principa tokom kreiranja dijagrama klasa u našem projektu te način na koji je sistem učinjen razumljivijim i lakšim za održavanje.

S – Single Responsibility Principle

Prema Single Responsibility principu svaka klasa bi trebala imati jednu odgovornost za dio funkcionalnosti koje pruža softver.
U našem dijagramu nijedna od njih, osim klase 'Hotel' ne koristi metode koje bi eventualno pristupile bazi podataka ili nekoj lokalnoj datoteci. Ovim vidimo da, zaista svaka klasa ima samo jednu odgovornost, sa izuzetkom klase 'Hotel', koja je svakako klasa koja upravlja cijelim sistemom i vrši komunikaciju sa bazom. Dakle ovaj princip je ispunjen.

O – Open/Closed Principle

U skladu sa Open/Closed principom svaka klasa bi trebala biti otvorena za nadogradnje, ali zatvorena za modifikacije.
Iz dijagrama možemo vidjeti da većina klasa sadrži jednostavne atribute i metode kao što su konstruktori, eventualno gettere i settere, a to su klase: 'Soba', 'Agencija', 'Zahtjev', 'Usluga', 'Aktivnost', 'Rezervacija' i 'Zaposlenik'. Ove jednostavne klase ne predstavljaju problem, nadogradnja je vrlo lagana i izvodljiva, a modifikacija neće biti, jer su atributi klasa dobro isplanirani. Kod ostale tri klase: 'Osoba', 'Gost', i 'Hotel', imamo dublju analizu. Klase 'Hotel' i 'Gost' su kontejnerske klase, koje za svoje atribute imaju liste drugih klasa, a metode im omogućavaju lagano dodavanje i brisanje elemenata tih listi. Zbog ovog tipa uređenja, modifikacije se neće dešavati jer ne postoji dirketna veza kontejnerske klase sa atributima ostalih klasa, dok je mogućnost nadogradnje kontejnerske klase omogućena. Za klasu 'Osoba' vrijedi da je ona apstraktna klasa, iz koje su naslijeđene neke od gore pobrojanih, te također podržava zatvorenost na modifikaciju. Iz ove apstraktne klase, sasvim je moguće izvesti nove klase, bez da je potrebno modificirati već postojeću apstrakciju, dok je moguće nadograditi tu istu apstraktnu klasu. Ovim vidimo da je ovaj princip ispunjen.

L – Liskov Substitution Principle

Prema principu Liskov Substitution svaka klasa treba biti zamjenjiva svim svojim podtipovima tako da ispravnost programa ne bude narušena.
U našem dijagramu postoji jedno nasljeđivanje. Iz klase 'Osoba' nasljeđuju se klase 'Gost' i 'Zaposlenik'. Gost, i Zaposlenik (koji može biti menadžer i recepcioner, što je definsano putem enuma u klasi 'Zaposlenik') u realnom svijetu jesu osobe, pa je

ovakvo nasljeđivanje uvedeno radi lakše implementacije svake od navedenih klasa. Sistem bi radio bez problema ukoliko bi se klasa 'Osoba' zamijenila sa nekim od ova dva podtipa, te stoga vidimo da je i ovaj princip ispunjen.

I – Interface Segregation Principle

Interface Segregation princip kaže da je bolje imate više specifičnih interfejsa nego jedan generalizovani.
Kako naš model ne koristi interfejse tim ne dolazi do narušavanja ovog principa.

D – Dependency Inversion Principle

Prema ovom principu sistem klasa i njegovo funkcionisanje bi trebalo ovisiti o apstrakcijama, a ne o konkretnim implementacijama.
Zadovoljenost ovog principa je često popraćena poštivanjem principa Open/Closed i Liskov Substitution. Dva osnovna tumačenja principa su da klase visokog nivoa ne bi trebale ovisiti o klasama niskog nivoa te da apstrakcije ne bi trebale ovisiti o detaljima nego detalji o apstrakcijama. Vidimo da je to u našem sistemu ispunjeno u naslijeđivanju. Klase 'Gost' i 'Zaposlenik' su naslijeđene iz bazne klase 'Osoba', koju smo napravili apstraktnom klasom. U našem sistemu je omogućena promjena komponenti višeg i nižeg nivoa bez uticaja na druge klase. I ovaj princip je ispunjen.