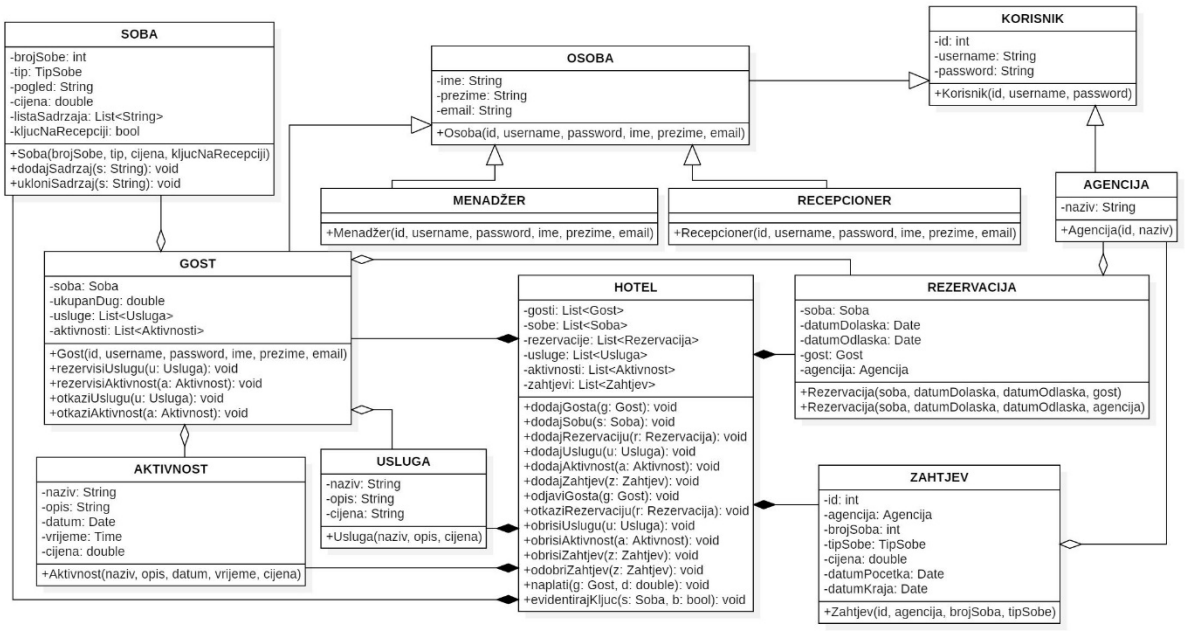
# UPOTREBA SOLID PRINCIPA U PROJEKTU 'HOTEL'

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.




*Class Diagram za projekt 'Hotel'*

## 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 ne koristi metode koje bi eventualno pristupile bazi podataka ili nekoj lokalnoj datoteci. Ovim vidimo da, zaista svaka klasa ima samo jednu odgovornost, osim 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. U našem dijagramu, većina klasa sadrži jednostavne atribute i metode tipa gettera i settera, a to su klase: Soba, Menadžer, Recepcioner, Agencija, Zahtjev, Usluga, Aktivnost i Rezervacija. Ove jednostavne klase ne predstavljaju problem, nadogradnja je vrlo lagana i izvodljiva, a modifikacija neće biti, jer su atributi klasa dobro isplanirani. Kod ostale četiri klase: Korisnik,

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 ostale dvije klase vrijedi da su one apstraktne klase, iz kojih se naslijeđuju neke od gore pobrojanih, te također podržava zatvorenost na modifikaciju. Iz ovih apstraktnih klasa, sasvim je moguće izvoditi nove klase, bez da je potrebno modificirati već postojeću aptrakciju, 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 postoje dva nasljeđivanja. Iz klase 'Korisnik' nasljeđuju se klase 'Osoba' i 'Agencija' dok se iz klase 'Osoba' nasljeđuju klase 'Gost', 'Menadžer' i 'Recepcioner'. Gost, menadžer i recepcioner u realnom svijetu jesu osobe, pa je ovakvo nasljeđivanje uvedeno i radi lakše implementacije svake od navedenih klasa. Sistem bi radio bez problema ukoliko bi se klasa 'Osoba' zamijenila sa nekim od ova tri podtipa. Što se tiče klase 'Agencija', ona nije mogla biti razmatrana kao podtip klase 'Osoba'. Obzirom da 'Agencija' ipak ima zajedničku tačku sa svim podtipovima klase 'Osoba' jer koristi sistem i predstavlja aktera sistema, uveli smo i nasljeđivanje klase 'Korisnik' kao što je objašnjeno na početku. Sada se klasa 'Korisnik' može zamijeniti klasama 'Osoba' ili 'Agencija' tako da ne dođe do narušavanja ovog principa.

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 svakom naslijeđivanju. Klase 'Osoba' i 'Agencija' koje su naslijeđene iz bazne klase 'Korisnik', koju smo napravili apstraktnom, te klase 'Gost', 'Menadžer' i 'Recepcioner' koje su također izvedene iz bazne klase 'Osoba', koja je postala apstraktna klasa. U našem sistemu je omogućena promjena komponenti višeg i nižeg nivoa bez uticaja na druge klase. I ovaj princip je ispunjen.