SOLID Principi

1. Single responsibility principle (SRP)

Prema ovom principu klasa bi trebala imati samo jedan razlog za promjenu, odnosno jedna klasa da ima jednu odgovornost

U našem dijagramu ovaj princip je ispunjen jer svaka klasa pored standarding get i set metoda sadrži samo metode koje se tiču njene odgovornosti. Tako npr. klasa Reservation sadrzi metode koje se tiču samo nje, kao npr. da vrati broj dana rezervacije, neku rezervaciju ili sve rezervacije. Također može se vidjeti iz klase za popust da je ovaj princip zadovoljen jer će ona izvršiti obračun i postaviti atribute ove klase koji će se koristiti u rezervacijama dalje.

2. Open closed principle (OCP)

Sve klase, moduli i funkcije trebaju biti otvorene za nadogradnju, ali zatvorena za modifikacije.

Ovaj princip je zadovoljen, a to se može vidjeti npr. kada Admin treba mijenjati podatke nekog Usera. Da nije uveden interfejs izmjena u UserDB bi mogla forsirati izmjenu i Admin klase, međutim dodan je interfejs kojeg će implementirati klasa UserDB. Ovim je razdvojen admin direktno od baze i samim time moguća je i dalja nadogradnja raznim metodama koje bi bile potrebne. Nigdje drugo također nema kršenja ovog principa. Primjena se može vidjeli i kod klase Discount, ukoliko dođe do bilo kakvih promjena u Reservation ili Discount neće direktno jedna uticati na drugu, već je potrebno interfejs izmjeniti samo.

3. Lisk substitution principle (LSP)

Svaki podtip mora biti zamjeniv osnovnim tipom

Ovo je princip koji se veže za apstraktne klase i izvođenje iz njih. Ovaj princip je zadovoljen zato što je zaista moguće bilo gdje zamijeniti admina ili registrovanog korisnika sa User klasom. Na drugim mjestima ne može se desiti da se naruši ovaj princip.

4. Interface segregation principle (ISP)

Klijenti ne treba da ovise o metodama koje neće upotrebljavati.

Ovaj princip se koristi kod klasa sa velikim brojem metoda, gdje se mogu izazvati neželjene povezanosti između klasa. U našem primjeru nema takvih klasa.

5. Dependency inversion principle (DIP)

Moduli visokog nivoa ne bi trebali ovisiti o onima niskog nivoa, već treba da ovise od apstrakcija. Apstrakcije ne trebaju ovisi od detalja, već detalji od apstrakcija.

U našem primjeru da klasa User nije apstraktna ovaj princip bi mogao biti narušen. Moglo bi se reći i da je dio za računanje popusta neki modul, pa je samim time radi interfejsa ovaj princip zadovoljen.