SOLID principi

1. S – Single Responsibility Principle (SRP)

Kada bi na našem dijagramu klase postojala klasa *Uposlenik* u kojoj bismo imali metode:

• obrisiNarudzbu

• izmijeniCijenu

• dajArtikal

Primijetimo da ovu klasu možemo podijeliti u dvije klase, i to *Zaposlenik* i *Administrator*, budući bi klasa *Uposlenik* "znala previse". Npr. ukoliko bi došlo do promjene u metodi izmijeniCijenu to bi uticalo i na druge tipove uposlenika koji uopšte ne koriste tu metodu.

2. O – Open – Closed Principle (OCP)

Da bi bio ispoštovan OCP, nijedna izmjena unutar aplikacije ne smije dovesti do izmjene postojećeg koda već samo do dodavanja novog koda.

U našem slučaju imamo klasu *Administrator* koja radi neke akcije nad narudžbama i artiklima. Ukoliko bi se nakon određenog vremena pojavila potreba da instance klase *Administrator* vrše pojedine akcije nad artiklima ili narudžbama (recimo dodavanje ili brisanje) morali bismo vršiti modifikaciju implementacije klase *Zaposlenik.*

Ovaj problem se jednostavno može riješiti dodavanjem *ZaposlenikInterface* klase iz koje će se izvoditi različite 'vrste' uposlenika po potrebi.

3. L – Liskov Substitution Principle (LSP)

LSP kaže da ukoliko imamo klasu *S* koja je naslijeđena iz klase *T*, onda uvijek mora biti moguće instance klase *T* zamijeniti instancama klase *S*.

U našem slučaju Liskov princip je ispoštovan zato što imamo odvojene klase *Zaposlenik* i *Administrator.*

S druge strane, ukoliko bismo imali klasu *Uposlenik* iz koje bi bile izvedene klase *Zaposlenik* i *Administrator*, ovaj princip ne bi bio ispoštovan. Klasa *Uposlenik* bi imala neke metode koje omogućavaju upravljanje artiklima i narudžbama. Kada bismo pokušali da instancu tako definisane klase Uposlenik zamijenimo instancom klase *Zaposlenik* (koja radi samo određene akcije sa artiklima) direktno bismo prekršili LSP.

4. I – Interface – Segregation Principle (ISP)

Kako na našem dijagramu klasa postoje klase *Gost* i *RegistrovaniKorisnik* koja je izvedena iz klase *Gost*, postoji mogućnost da dodavanjem različitih vrsta korisnika klasa *Gost* postane preopterećena. Rješenje za tu situaciju je dodavanje interface-a *GostInterface* i iz njega naslijedimo klasu *Gost.* U tom slučaju bi bio ispoštovan ISP.

5. D – Dependency – Inversion Principle (DIP)

DIP kaže da moduli visokog nivoa ne bi trebali da zavise od modula niskog nivoa.

Konkretno u našem sistemu možemo primijetiti da klasa *Administrator* direktno zavisi od *RegistrovaniKorisnik* klase, budući da sam korisnik ima mogućnost da otkaže narudžbu, zbog čega se implicira da će *Administrator* biti afektiran promjenama na klasi *RegistrovaniKorisnik* (u smislu da narudžbe koju je RegistrovaniKorisnik eventualno otkazao). Ovo možemo riješiti dodavanjem *AdministratorInterface* interfejsa. *Administrator* klasa koristi apstrakcije,

međutim objekti *Administrator* klase će koristiti objekte izvedene od *RegistrovaniKorisnik* klase.