SOLID principi

1. S – Single Responsibility Principle (SRP)

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

• obrisiProizvod

• izmijeniCijenu

• dajProizvod

Primijetimo da ovu klasu možemo podijeliti u dvije klase, i to *Arikal* i *StavkaNarudzbe*, budući bi klasa *Proizvod* "znala previše". Npr. ukoliko bi došlo do promjene u metodi izmijeniCijenu to bi uticalo i na već postojeće narudžbe, tj. na već postojeće artikle koje je korisnik naručio prije promjene cijene.

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 *Artikal* koja sadrži određene attribute i akcije nad artiklima. Ukoliko bi se nakon određenog vremena pojavila potreba da unutar klase *Artikal* dodamo neke nove atribute ili metode, to bi povlačilo da se i vrši izmjena i već postojećih artikala u narudžbi ili korpi (kada bismo unutar njih koristili instance klase *Artikal*).

Ovaj problem rješavamo tako što pravimo dvije različite klase, *Artikal* i *StavkaNarudzbe,* koji će imati različite atribute i metode.

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, budući da na našem dijagramu klasa nemamo naslijeđivanje.

S druge strane, da smo recimo 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)

U slučaju da klasa *Korisnik* ne bude interface, postojala bi mogućnost da dodavanjem različitih vrsta korisnika klasa postane preopterećena. Rješenje za tu situaciju je dodavanje interface-a *KorisnikInterface* i da iz njega naslijedimo pojedine tipove korisnika. U tom slučaju je 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 *Narudzba* direktno zavisi od *Korisnik* klase, budući da sam korisnik ima mogućnost da obriše i uredi narudžbu, zbog čega se implicira da će *Narudzba* biti afektirana akcijama instance klase *Korisnik* .

Na primjer, umjesto da klasa *Narudzba* direktno koristi klasu *Korisnik*, možemo dizajnirati interfejs ili apstrakciju koji će definisati funkcionalnost koja je potrebna klasi *Narudzba*, a zatim implementirati tu funkcionalnost u klasi *Korisnik*. Na taj način, *Narudzba* neće direktno zavisiti od implementacije *Korisnik* klase, već će zavisiti samo od interfejsa ili apstrakcije.