SOLID PRINCIPI

S - Single Responsibility Principle
O - Open-Closed Principle
L - Liskov Substitution Principle
I - Interface-Segregation Principle
D - Dependency-Inversion principle

1. Single Responsibility Principle - Princip pojedinačne odgovornosti

Svaka klasa treba da ima jednu jedinu odgovornost.

Klase **Korisnik**, **Popust**, **Izvještaj**, **Proizvod**, **Narudžba** i **Plaćanje** su ispunile ovaj princip jer svaka ima jasno definisanu jedinstvenu odgovornost: čuvanje atributa, upravljanje narudžbom ili obradom plaćanja.

Klasa **Zaposlenik** ima odgovornost pravljenja narudžbi, dok **Kupac** ima odgovornost plaćanja narudžbi, što je u skladu s njihovim poslovnim ulogama.
Sve klase poštuju SRP, imaju jasno definisane zadatke i nisu preopterećene dodatnim odgovornostima

2. Open/Closed Principle - Princip otvoreno/zatvoreno

Klasa treba da bude otvorena za proširenje, ali zatvorena za izmjene.

Klase poput **Popust**, **Narudžba**, **Proizvod**, **Izvještaj**, **Plaćanje** su dizajnirane tako da se mogu proširivati bez izmjene postojećeg koda.

Apstraktna klasa **Korisnik**, kao i izvedene klase **Zaposlenik** i **Kupac** koje su dobar primjer nasljeđivanja gdje proširenje ne zahtijeva izmjenu baze.

3. Liskov Substitution Principle - Liskov princip zamjene

Objekti podklasa treba da se mogu koristiti umjesto objekata nadklasa bez narušavanja tačnosti programa. Klase **Zaposlenik** i **Kupac** uspješno nasljeđuju **Korisnik**. Mogu se koristiti na mjestu bazne klase bez problema.

Ostale klase nisu u naslijeđenim odnosima, pa LSP na njih ne utiče direktno.

4. Interface Segregation Principle - Princip izoliranja interfejsa

Ovaj princip govori da korisnici ne trebaju biti pritisnuti interfejsima koje ne koriste.
Klase **Kupac**, **Popust**, **Narudžba**, **Proizvod**, **Izvještaj** i **Plaćanje** ne zavise od metoda koje ne koriste, što znači da ne sadrže nepotrebne funkcionalnosti koje ih opterećuju.

5. Dependency Inversion Principle - Princip inverzije ovisnosti

DIP je ispunjen u većini slučajeva jer se koristi apstraktna klasa **Korisnik** kao osnova za izvedene klase.

Međutim, **klasa Popust ne ispunjava** ovaj princip jer direktno zavisi od konkretnih detalja.