1. **Princip pojedinačne odgovornosti:**

KLASA BI TREBALA IMATI SAMO JEDAN RAZLOG ZA PROMJENU

Na našem dijagramu nijedna klasa nije opterećena metodama za koje direktno nije vezana. Uzmimo za primjer klasu Cjenovnik. Ona brine samo o cjenovniku, što se vidi iz njene jedine metode koja se koristi za izmjenu cjenovnika. Ista je situacija i sa ostalim klasama (Artikal, Film...), koje u sebi sadrže metode za izmjenu, dodavanje i brisanje (artikla, filma...). Zaključujemo da je ovaj princip zadovoljen.

1. **Otvoreno zatvoren princip:**

ENTITETI SOFTVERA (KLASE, MODULI, FUNKCIJE) TREBALI BI BITI OTVORENI ZA NADOGRADNJU, ALI ZATVORENI ZA MODIFIKACIJE

Naše klase je moguće nadograditi, uz restrikcije da se postojeće ne smiju mijenjati. Dakle, trenutni atributi moraju ostati nepromjenjeni, zato što svaki atribut i svaka metoda ima svoju određenu funkcionalnost koja je ključna za ispravno funkcionisanje softvera.

1. **Liskov princip zamjene**:

PODTIPOVI MORAJU BITI ZAMJENJIVI NJIHOVIM OSNOVNIM TIPOVIMA

Apstraktna klasa Osoba ima dva podtipa Korisnik i Klijent i oba su zamjenjiva sa apstraktnom klasom. Metode koje se odnose na izmjenu, registraciju i brisanja novog korisnika su smještena upravo u klasi Korisnik, jer su sasvim nepotrebne u klasi Klijent. Prema tome, možemo slobodno zaključiti da je i ovaj princip zadovoljen.

1. **Princip inverzije ovisnosti**:

**A.**MODULI VISOKOG NIVOA NE BI TREBALI O VISITI OD MODULA NISKOG NIVOA. OBA BI TREBALO DA OVISE OD APSTRAKCIJA.

**B.** MODULI NE BI TREBALI OVISITI OD DETALJA. DETALJI BI TREBALI BITI OVISNI OD APSTRAKCIJA.

Klase ne ovise o konkretnim klasama, već isključivo o interfejsu i apstraktnoj klasi. To je uočljivo ako pogledamo sve strelice na dijagramu i vidimo da je cilj svake strelice interfejs, odnosno apstraktna klasa. Tako je smanjen utjecaj koji promjena može imati na sistem. Dakle, i ovaj princip je zadovoljen.

1. **Princip izoliranja interfejsa:**

KLIJENTI NE TREBA DA OVISE O METODAMA KOJE NEĆE UPOTREBLJAVATI.

Na našem dijagramu, ne postoji interfejs ili apstraktna klasa sa velikim brojem metoda, nego su sve razdvojene po funkcionalnostima koje trebaju da izvršavaju. U našem slučaju korisnici neće pasti pod utjecaj promjena koje su napravljene nad metodama koje oni ne pozivaju. Korisnik je tako potpuno zaštićen od promjene metoda koje ga se ne tiču.