SOLID PRINCIPI

- Single Responsibility Principe: Svaka klasa treba imati samo jednu ulogu;

  Princip S zahtijeva da svaka klasa ima samo jednu odgovornost, odnosno da klasa vrši samo jedan tip akcija kako ne bi ovisila o prevelikom broju konkretnih implementacija.

  Posmatrajući naš dijagram klasa možemo vidjeti da je ovaj princip ispoštovan. Prije svega, veliki broj klasa implementira samo gettere i setere. Zatim, klasa "Uposlenik" vrši tip akcija vezan za upravljanje proizvodom, klasa "Administrator" vrši tip akcija vezan za upravljanje uposlenicima, dok klasa "Sistem" upravlja tipom akcija koje pokreće korisnik.

- Open/Closed Principle: Klasa treba biti otvorena za nadogradnje, ali zatvorena za modifikacije;

  Princip O zahtijeva da klasa koja koristi neku drugu klasu ne treba biti modificirana pri uvođenju novih funkcionalnosti, ili pri potrebi za mijenjanjem druge klase.

  Ponovo, posmatrajući naš dijagram klasa možemo primjetiti da dodavanje novih metoda neće zahtjevati uređivanje već postojeće klase i njenih atributa. Naravno, klase koriste druge klase, ali to su uglavnom određene provjere koje ne vrše nikakvu modifikaciju drugih klasa. Također, dosta klasa koristi druge klase u metodama za dodavanje objekata, međutim i to ne predstavlja problem, jer dodavanje u listu, kao generička operacija, radi neovisno o tome kakav objekat se dodaje u listu. Zbog toga, prilikom dodavanja novih funkcionalnosti dovoljno je dodati nove metode, ne razmišljajući o narušavanju prethodnih.

- Liskov Substitution Principle: Svaka osnovna klasa treba biti zamjenjiva svim svojim podtipovima bez da to utječe na ispravnost rada programa;

  Princip L zahtijeva da nasljeđivanje bude ispravno implementirano, odnosno da je na svim mjestima na kojima se koristi osnovni objekat moguće iskoristiti i izvedeni objekat a da takvo nešto ima smisla.

  U našem sistemu postoji više nasljeđivanja: nasljeđivanje klasa Uposlenik i Kupac iz klase Korisnik, nasljeđivanje klase Administrator iz klase Uposlenik, nasljeđivanje klase Student iz klase Kupac, te nasljeđivanje klasa Monitor, Kuciste, HardDisk, Procesor, MaticnaPloca, SSD, ZvucnaKartica, Slusalice, Mis, GrafickaKartica, Disk, Tastatura i Memorija iz klase Proizvod. Međutim, baš zbog postojanja klasa Korisnik i Proizvod i ovaj princip je ispoštovan.

- Interface Segregation Principle: Bolje je imati više specifičnih interfejsa, nego jedan generalizovani;

  Princip I zahtijeva da i svi interfejsi zadovoljavaju princip S, odnosno da svaki interfejs obavlja samo jednu vrstu akcija.

  Još uvijek u našem sistemu ne postoji nijedan interfejs, međutim ne odbacujemo mogućnost njegovog pojavljivanja prilikom olakšavanja određenih poslova i akcija u narednim koracima razvijanja ovog projekta.

- Dependency Inversion Principle: Sistem klasa i njegovo funkcionisanje treba ovisiti o apstrakcijama, a ne o konkretnim implementacijama.

  Princip D zahtijeva da pri nasljeđivanju od strane više klasa bazna klasa uvijek bude apstraktna. Razlog za ovo je što je teško koordinisati veliki broj naslijeđenih klasa i konkretnu baznu klasu ukoliko ista nije apstraktna, a da pritom kod bude čitak i jednostavan za razumijevanje.

Budući da su klasa Korisnik i klasa Proizvod jedine klase iz kojih je nasljeđivan veći broj klasa, te obzirom na to da su one apstraktne, i ovaj princip je ispoštovan.