SOLID PRINCIPI

- *Single Responsibility Principle (princip pojedinačne odgovornosti)* tvrdi da bi svaka klasa trebala biti odgovorna samo za jednu funkciju unutar projekta. Odnosno, svrha svake klase trebala bi biti ograničena na izvršavanje samo jedne specifične zadaće, umjesto da obavlja više funkcija. Primjeri pojedinačne odgovornosti uključuju klasu Administrator koja se bavi promjenama vezanim za ponudu letova, kao što su kreiranje, brisanje i uređivanje leta, te klasu Let koja čuva podatke o pojedinom letu (poput broja putnika, vremena polijetanja i slijetanja) i povezuje klase Karta, Avion i Destinacija.
- *Open/Closed Principle* jedan od principa objektno orijentisanog programiranja je Otvoreno-Zatvoreni princip (Open/Closed Principle) koji predstavlja ideju da bi klase trebale biti otvorene za nadogradnju, ali zatvorene za modifikacije. Na primjeru apstraktne klase Korisnik, koja ima tri izvedene klase - Putnik, Radnik i Administrator, možemo vidjeti kako dodavanje atributa ili metoda u bilo koju od tih izvedenih klasa ne bi trebalo utjecati na samu klasu Korisnik. To znači da možemo nadograditi izvedene klase bez da moramo mijenjati samu apstraktnu klasu Korisnik.
- *Liskov Substitution Principle* - Liskov princip zamjene, svi podtipovi moraju biti zamjenjivi njihovim osnovnim tipovima bez da to utječe na ispravnost rada programa iz klase Korisnik postoje tri izvedene klase - Putnik, Radnik i Administrator. Metode klaseKorisnik su takve da imaju smisla da se pozivaju i nad klasama Putnik, Radnik ili Administrator jer predstavljaju samo kreiranje korisnika i gettere i settere.
- *Interface Segregation Principle* - princip izoliranja interfejsa, bolje je imati više specifičnih interfejsa, nego jedan generalizovani odnosno klijenti ne treba da ovise o metodama koje neće upotrebljavati. Ovo čuva korisnika od promjena metoda koje ga se ne tiču. S obzirom da interfejsi nisu korišteni, ovaj princip je ispoštovan.
- *Dependency Inversion Principle* Princip inverzije ovisnosti (Dependency Inversion Principle) propisuje da bi sistem klasa trebao ovisiti o apstrakcijama, a ne o konkretnim implementacijama. Apstraktne klase i interfejsi su manje podložni promjenama od njihovih konkretnih izvedenica, pa se zbog toga preporučuje da se sistem bazira na apstraktnim klasama i interfejsima, umjesto na stvarnim klasama. Na ovaj način, utjecaj promjena na sistem je smanjen.Primjena ovog principa može se vidjeti u primjeru klase Korisnik, koja je ujedno i apstraktna klasa od koje su izvedene konkretne klase Radnik i Administrator. Ovim se osigurava da sistem ovisi o apstrakciji Korisnik, a ne o konkretnim implementacijama Radnik i Administrator, što doprinosi fleksibilnosti sistema.