SOLID PRINCIPI

SINGLE RESPONSIBILITY PRINCIPLE - PRINCIP POJEDINAČNE ODGOVORNOSTI

Princip pojedinačne odgovornosti glasi:

KLASA BI TREBALA IMATI SAMO JEDAN RAZLOG ZA PROMJENU.

Klase trebaju imati pojedinačnu odgovornost. Glavni cilj ovog principa jeste smanjenje kompleksnosti. Potrebno je da klasa vrši samo jedan tip akcija kako ne bi ovisila o velikom broju konkretnih implementacija.

Ukoliko pogledamo dijagram klasa, možemo primijetiti da se svaka klasa bavi isključivo temama koje se tiču direktno nje. Tako klasa Korisnik sadrži kao atribute lične podatke koje bi svaki korisnik trebao imati. Također, napravljena je i klasa Instrukcija. Pridruživanje informacija koje posjeduje klasa Instrukcija nekoj drugoj klasi u sistemu, narušavalo bi ovaj SOLID princip. Slično, da je ovaj princip ispunjen, možemo vidjeti i na osnovu klasa Predmet i Materijal. Da smo klasi predmet pripisali i odgovornost za materijale, to bi činilo klasu Predmet suvišno kompleksnom. Također, odvojeno je plaćanje sa posebnim klasama kako ne bismo bespotrebno zasitili klasu Korisnik i sa tim dijelom.

OPEN CLOSED PRINCIPLE - OTVORENO ZATVOREN PRINCIP

Otvoreno zatvoren princip glasi:

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

Dodavanje novih metoda i atributa postojećim klasama neće uzrokovati promjenu na trenutnim stanjem klasa. Dakle, klase su otvorene u smislu da ih možemo proširiti, možemo stvoriti podklase, dodati nove atribute i metode.

Svaka klasa treba istovremeno biti otvorena za nadogradnju, ali zatvorena za modifikacije. Ne bi se smjelo desiti da nadogradnja klase zahtijeva modifikaciju neke druge klase. Upravo je to ispunjeno u našem dijagramu klasa. U našem dijagramu klasa najzastupljenija je veza agregacije, što bi značilo da većina klasa sadrži objekte ili kolekcije drugih klasa, tako da izmjena u nekoj klasi neće implicirati promjenu druge klase. Na primjer, ukoliko se u sistemu odlučimo omogućiti još jedan način održavanja instrukcije (na primjer: održavanje instrukcija u paru), to ni na koji način neće zahtijevati modifikaciju klasa, nego ćemo samo postojeći sistem dodatno nadograditi. Također, klase jedna drugu ne sprječavaju pri nadogradnji kako one koriste samo već implementirane metode i atribute te će njihovo dodavanje ili

eventualna promjena u pozadinskom izvršavanju određenih metoda biti izvodiva dok god ne modifikujemo finalni rezultat.

LISKOV SUBSTITUTION PRINCIPLE - LISKOV PRINCIP ZAMJENE

Liskov princip zamjene glasi:

> PODTIPOVI MORAJU BITI ZAMJENJIVI NJIHOVIM OSNOVNIM TIPOVIMA

Dijagram klasa posjeduje samo jednu apstraktnu klasu, a to je Korisnik. Iz nje su izvedene klase Student i Tutor. Jasno je da na svakom mjestu gdje se Koriste klase Student i Tutor, je moguće koristiti klasu Korisnik, jer svaki Student i Tutor su ujedno i korisnici. Klasa Korisnik ima samo osnovne atribute i metoda kako ne bi dolazilo do problema pri eventualnom dodavanju novih izvedenih klasa.

INTERFACE SEGREGATION PRINCIPLE - PRINCIP IZOLIRANJA INTERFEJSA

Princip izoliranja interfejsa glasi:

> KLIJENTI NE TREBA DA OVISE O METODAMA KOJE NEĆE UPOTREBLJAVATI

U našem sistemu ne postoje interfejsi kako nemamo tzv. „debele“ klase koje bi imale previše metoda, tako da smatramo da je ovaj princip zadovoljen.

DEPENDENCY INVERSION PRINCIPLE - PRINCIP INVERZIJE OVISNOSTI

Princip inverzije ovisnosti glasi:

- MODULI VISOKOG NIVOA NE BI TREBALI OVISITI OD MODULA NISKOG NIVOA. OBA BI TREBALO DA OVISE OD APSTRAKCIJA.
- MODULI NE BI TREBALI OVISITI OD DETALJA. DETALJI BI TREBALI BITI OVISNI OD APSTRAKCIJA.

To možemo razumjeti i na sljedeći način:
Princip inverzije ovisnosti zahtijeva da pri nasljeđivanju od strane više klasa bazna klasa uvijek bude apstraktna. Razlog za ovo je što je teško koordinirati veliki broj naslijeđenih klasa i konkretnu baznu klasu ukoliko ista nije apstraktna, a da pritom kod bude čitak i jednostavan za razumijevanje.

Ovaj princip je ispoštovan pošto su klase Tutor i Student naslijeđene iz klase Korisnik koja je *apstraktna.* Da klasa Korisnik nije apstraktna, princip inverzije ovisnosti bi bio narušen, što bi za posljedicu imalo osjetljivost sistema na promjene. Osim klase Korisnik imamo još ovisnosti o klasama kao što su List ili Map koje su tipovi podataka koji se neće zasigurno mijenjati tako da i tu nema narušenja ovog principa.