Grupa2- Zamger

OOAD

SOLID principi

Studenti:

Huso Hamzić

Paša Džumhur

Rijad Handžić

S princip – Svaka klasa mora imati samo jedan razlog za promjenu. Dakle, u jednoj klasi se grupišu funkcionalnosti koje se mijenjaju iz istog razloga, te sa na taj način ograničava utjecaj promjena koje se mogu desiti. Ukoliko se primijeni ovaj princip, postiže se upravo to da se promjene odraze na manji broj klasa (u odnosu na situaciju kada Single responsibility princip nije zadovoljen).

- Da bi se ispoštovao ovaj princip, sistem je dizajniran tako da su u klasama grupisane samo one metode koje se mijenjaju iz istog razloga. Ovime je postignuto da ukoliko dođe do promjena one se dešavaju na jednom mjestu, u jednom modulu, te znatno olakšava izmjene u kodu koje je potrebno napraviti da bi isti funkcionisao pravilno.
- Kao primjer možemo uzeti klasu Aktivnost. Ona sadrži get i set metode koje postavljaju i vraćaju vrijednosti atributa date klase. U slučaju da dođe do nekih promjena, potrebno ih je izvršiti samo na jednom mjestu i na taj način je ograničen utjecaj promjena na sistem.

O princip – Ovaj princip nalaže da sistem treba biti otvoren za nadogradnje, a zatvoren sa modifikacije. Ovo znači da se promjene u samoj implementaciji sistema ne trebaju dešavati svaki put kada se zahtjevi koje sistem treba ispunjavati promijene. Dakle, treba uzeti u obzir mogućnost da će u budućnosti biti potrebno dodati nove funckionalnosti u sistem, ali tako da se postojeći kod ne modifikuje, nego nadogradi novim mogućnostima. Ovaj princip djeluje možda bespotreban kada ga razmatramo na primjeru programa od nekoliko desetina ili stotina linija, ali kada je u pitanju kod od više hiljada linija, ovaj princip je veoma važan i koristan.

- Klase imaju veliki broj metoda te ih se još može dodavati bez da se mijenja sami nukleus klase, odnosno dodavanje novih metoda neće implicirati da se čitava klasa mora mijenjati odnosno modificirati. Također većina klasa sadrži samo kolekcije objekata drugih klasa tako da izmjena u nekoj klasi neće implicirati da se neka druga klasa radi toga mora mijenjati. Napomenimo da je čitava ideja(logika) iza sistema razrađena(dani planiranja) i sve je pravljeno sa jasnom vizijom implementacije svega.
- Primjer ispunjenja ovog principa možemo vidjeti u klasi PredmetZaStudenta. Ona sadrži listu *List<Aktivnosti>.* Aktivnosti je apstraktna klasa iz koje se nasljeđuju tipovi aktivnosti kao što je već urađeno za ispite i zadaće. Da smo imali dvije odvojene liste za aktivnosti i zadaće, Open closed princip bi bio narušen. Naime, ako bismo u sistem dodali novi tip aktivnosti, recimo seminarski rad, u klasi PredmetZaStudenta bismo morali dodati novu kolekciju koja sadrži tip SeminarskiRad. Rješenje ovog problema je da se sve vrste aktivnosti nasljeđuju iz apstraktne bazne klase Aktivnosti, a da u klasi PredmetZaStudenta imamo kolekciju koja sadrži Aktivnosti. Korištenjem polimorfnog kontejnera je riješen ovaj problem.

L princip – sva nasljeđivanja moraju biti ispravno implementirana

- Na dijagramu vidimo jednu apstraktnu klasu *Aktivnost* i dvije bazne klase *Student* i *NastavnoOsoblje.* Očito je da je na mjestima gdje se koristi osnovni objekat može koristiti i izvedeni objekat po potrebi, ali slijedit ćemo pravilo da uvijek vratimo pokazivač na osnovni tip iz metoda npr *List<Aktivnost>* , a ne *List<Zadaća>* po konvenciji.

I princip – svaki interfejs mora obavljati samo jednu vrstu akcija

- Kako nemamo niti jedan interfejs u sistemu(za sada, no ne znači da u daljem razvoju neće doći do potrebe za nekim) ovaj princip ćemo preskočiti.

D princip – pri nasljeđivanju od više klasa, bazna klasa mora biti apstraktna

- Imamo jednu apstraktnu klasu *Aktivnost* i iz nje su izvedene klase *Ispit* i *Zadaća* odnosno *Aktivnost* se neće nigdje koristiti ni instancirati(jer će se koristiti samo ili Ispit ili Zadaća) pa je to ok da bude apstraktna klasa. No međutim imamo dvije bazne klase *Student* i *NastavnoOsoblje* koje su bazne klase a nisu apstraktne klase jer će se po defaultu podrazumijevati da je npr BSC student objekat tipa *Student* i moći će se instancirati dok će se moći i instancirati objekat tipa *MasterStudent* . Analogno je i za *NastavnoOsoblje* i *Profesor* (gdje će se bilo koja osoba iz ansambla moći instancirati kao objekat tipa *NastavnoOsoblje* ali će se moći posebno odvojiti i osobe iz ansambla koji su i *Profesor* -i). Tako da smatramo da je i ovaj princip ispoštovan, odnosno da je ovo osmišljeno na ispravan način.