Primjena SOLID principa na klase

1. Opis ispunjenja SOLID principa

1.1. Single Responsibility Principle – Princip pojedinačne odgovornosti

Ovaj princip je ispunjen jer svaka klasa zna za samo jedan pojam. Recimo, sve korisničke klase posjeduju metode koje samo taj korisnik može koristiti. Slično se ponašaju i klase `Predmet`, `Ocjena`, `Razred` i `Izostanak`, tako što opisuju isključivo pojmove na kojima su bazirane.

1.2. Open/Closed Principle – Otvoreno/zatvoreni princip

Ovaj princip je zadovoljen zato što smo kao atribute jedne klase navodili objekte druge klase, tako da dodavanje nove operacije unutar jedne klase neće povlačiti promjenu druge klase. Također smo maksimizirali korištenje relacija agregacija i kompozicija da utvrdimo odnose između cjeline i njenih dijelova. Na primjer, dodavanje nove operacije unutar klase `Predmet` neće zahtijevati izmjene klase `Razred` iako postoji relacija između njih.

1.3. Liskov Substitution Principle – Liskov princip zamjene

U dizajnu sistema prikazanom na dijagramu klasa, Liskov princip zamjene je zadovoljen kroz korištenje apstraktne klase Korisnik koja služi kao zajednička osnova za sve tipove korisnika (učenik, nastavnik, roditelj, administrator), pri čemu se svi specifični oblici korisnika mogu koristiti umjesto osnovnog tipa bez narušavanja funkcionalnosti sistema. Metode unutar sistema su definisane tako da prihvataju parametre tipa Korisnik ili njegovih podtipova, čime se omogućava zamjenjivost objekata bez potrebe za dodatnim provjerama tipa u kodu. Dodatno, korištenje enumeracija za uloge i statuse omogućava standardizovano ponašanje i jednostavnu zamjenu vrijednosti, dok su metode jasno ograničene na odgovarajuće uloge korisnika (npr. samo nastavnici mogu unositi ocjene), čime se izbjegava rušenje principa putem neodgovarajuće upotrebe metoda. Ovakav pristup osigurava da promjene u implementaciji podtipova ne utiču negativno na funkcionalnost sistema koji koristi osnovni tip Korisnik, čime je Liskov princip zamjene u potpunosti ispoštovan.

1.4. Interface Segregation Principle – Princip izoliranja interfejsa

Interfejsi koji predstavljaju funkcionalnosti specifične za pojedine koncepte, poput `Učenik`, `Predmet`, `Ocjena` ili `Izostanak`, ne prisiljavaju klijente da implementiraju nepotrebne metode koje se odnose na funkcionalnosti koje nisu relevantne za učenika. Na primjer, `Učenik` bi mogao sadržavati samo metode relevantne za učenike, kao što je `dajOcjenu()`, čime se osigurava da klijenti koriste samo one metode koje su im potrebne.

1.5. Dependency Inversion Principle – Princip inverzije ovisnosti

Moduli visokog nivoa ovise o apstrakcijama, a ne o konkretnim implementacijama. Na primjer, ako postoje moduli koji trebaju pristupiti informacijama o učesnicima, mogu ovisiti o apstrakcijama kao što su `Učenik`, `Ocjena` ili `Izostanak`, umjesto o konkretnim implementacijama tih klasa. Ovo omogućuje zamjenu implementacija bez utjecaja na kôd koji ih koristi.