



Univerzitet u Sarajevu
Elektrotehnički fakultet u Sarajevu
Odsjek za računarstvo i informatiku



Projektni zadatak

Studentski dom

Dženana Huseinspahić, Esma Karahodža,
Dženeta Kudumović

Objektno orjentisana analiza i dizajn

Akadska godina: 2019/2020

Sarajevo, maj 2020.

Sadržaj

1	Opis projekta	2
2	Klase	3
3	Solid principi	6
3.1	Single Responsibility Principe	6
3.2	Open/Closed Principle	6
3.3	Liskov Substitution Principle	6
3.4	Interface Segregation Principle	6
3.5	Dependency Inversion Principle	6
4	Paterni	7
4.1	Strukturalni paterni	7
4.2	Kreacijski paterni	7
5	MVC	9

1 Opis projekta

Na svim većim univerzitetima širom svijeta veliki broj studenata živi u studentskim domovima. Namjena ovog softvera je da studentima olakša izvršavanje svakodnevnih aktivnosti u domu i da uposlenicima pomogne da što efikasnije izvršavaju neke od resursno zahtjevnih poslova i studentskih servisa. Na taj način se automatiziraju određeni procesi koji oduzimaju dosta vremena studentima i osoblju koje ih izvršava.

Funkcionalnosti

- Prijem studenata u studentski dom – popunjavanje aplikacije, kontrola dokumentacije, odobravanje zahtjeva
- Održavanje informacija o studenatima - lični podaci i osnovne informacije vezane za servise u studentskom domu
- Dnevna rezervacija obroka u kuhinji od strane studenata
- Prijava zahtjeva za tekućim održavanjem studentskih apartmana i zajedničkih prostorija
- Upravljanje zahtjevima za održavanje – dodjela zadataka tehničkom osoblju, evidentiranje završetka poslova i izvještavanje

Akteri

- Student
- Uposlenik uprave
- Šef kuhinje
- Šef odjela tehničkog održavanja

2 Klase

Student

- atributi
 - imeIPrezime: String
 - JMBG: String
 - adresaStanovanja: String
 - fakultet: String
 - godinaStudiranja: int
 - brojTelefona: int
 - brojSobe: int
 - email: String
 - brojIndeksa: int
 - brojBonova: int
- metode
 - <<create>>Student(in s:String, in jmbg:String, in a:String, in f:String, in g:int, in br:int, in brS:int, in e:String, in i:int, in b:int)

StudentskiDom

- atributi
 - studenti: List<Student>
- metode
 - <<create>>studentskiDom()
 - prijava(in student:Student): void
 - odjava(in student:Student): void

PrijavaObroka

- atributi
 - student: Student
 - rucak: Boolean
 - vecera: Boolean
 - zaPonijet: Boolean
- metode
 - <<create>>prijavaObroka(in s:Student)
 - smanjiBonove(): void
 - dodajPrijavu(in s:Student): PrijavaObroka

PrijavaUDom

- atributi
 - student: Student
 - vrijemePrijava: Date
- metode
 - <<create>>prijavaUDom(in s:Student, in d:Date)

PrijavaKvara

- atributi
 - student: Student
 - tipKvara: enum
 - opisKvara: String
 - vrijemeKvara: Date
- metode
 - <<create>>prijavaKvara(in s:Student, in t:enum, in o:String, in v:Date)
 - dodajPrijavu(in s:Student): PrijavaKvara

UposlenikDoma

- interface
- atributi
 - imeIPrezime: String

SefKuhinje

- izvedena klasa iz klase UposlenikDoma
- atributi
 - obroci: List<PrijavaObroka>
- metode
 - <<create>>sefKuhinje()
 - odobriPrijava(): void

UposlenikUprave

- izvedena klasa iz klase UposlenikDoma
- atributi
 - prijave: List<PrijavaUDom>
- metode
 - <<create>>uposlenikUprave(s: String)
 - odobriPrijavuUDom(prijava: PrijavaUDom): Boolean
 - promijeniPodatkeOStudentu(student: Student): void
 - unesiStudenta(student: Student): void

SefTehnickogOdrzavanja

- izvedena klasa iz klase UposlenikDoma
- atributi
 - radnici: List<Radnik>
 - kvarovi: List<PrijavaKvara>
- metode
 - <<create>>sefTehnickogOdrzavanja()
 - pronadjiRadnika(prijava: PrijavaKvara): PrijavaKvara

Radnik

- izvedena klasa iz klase SefTehnickogOdrzavanja
- atributi
 - Usluga: enum
 - brojTelefona: int
- metode
 - <<create>>radnik(ustuga: enum, brojTelefona: int)

3 Solid principi

3.1 Single Responsibility Principle

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.

U ovom projektu princip S je ispoštovan jer svaka klasa vrši samo jednu vrstu akcija, npr. postoje tri različite klase za različite prijave umjesto jedne koja bi slala podatke svim klasama.

3.2 Open/Closed Principle

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

Vidimo da za sve klase na klasnom dijagramu vrijedi da možemo mijenjati okruženje oko klase bez promjene same klase. Npr. na našem klasnom dijagramu nijedna klasa ne zavisi od neke druge klase na način da bi promjenom te druge klase morali modificirati i prvu.

3.3 Liskov Substitution Principle

Princip L zahtijeva da nasljeivanje 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.

Liskov princip je ovdje zadovoljen jer se sve naslijeene klase mogu zamijeniti svojim osnovnim tipom, npr. klasu šefKuhinje možemo zamijeniti klasom Uposlenik Doma bez da stvorimo konflikt.

3.4 Interface Segregation Principle

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

Klasni dijagram sadrži samo jedan interfejs, UposlenikDoma. On obavlja samo jednu akciju a to je da povezuje tri vrste uposlenika doma. Zbog toga je i ovaj princip zadovoljen.

3.5 Dependency Inversion Principle

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

Ovaj projekat ne sadrži nijednu baznu klasu koja nije apstraktna i samim time je ovaj princip automatski ispunjen.

4 Paterni

4.1 Strukturalni paterni

Proxy

Proxy patern služi za dodatno osiguravanje objekata od pogrešne ili zlonamjerne upotrebe. Primjenom ovog paternu omogućava se kontrola pristupa objektima, te se onemogućava manipulacija objektima ukoliko neki uslov nije ispunjen, odnosno ukoliko korisnik nema prava pristupa traženom objektu.

Proxy patern je korišten kako bi samo dreeni stanovnici mogli pristupati određenim dijelovima sistema, np. ne bi trebalo da šef kuhinje odobri prijave u dom ili da korisnik doma pristupi informacijama o drugom korisniku doma. U klasi Proxy svakom korisniku dodjeljujemo odgovarajući nivo pristupa, a zatim korisnik može pristupati odgovarajućim metodama iz klase IStudentskiDom.

Composite

Composite patern služi za kreiranje hijerarhije objekata. Koristi se kada svi objekti imaju različite implementacije nekih metoda, no potrebno im je svima pristupati na isti način, te se na taj način pojednostavljuje njihova implementacija.

U našem projektu je postojala apstraktna klasa Prijava koja je zamijenjena interfejsom IPrijava koji su sve 3 vrste prijava naslijedile i tako je hijerarhija pojednostavljena.

4.2 Kreacijski paterni

Singleton

Singleton patern služi kako bi se neka klasa mogla instancirati samo jednom. Na ovaj način može se omogućiti i tzv. lazy initialization, odnosno instantacija klase tek onda kada se to prvi put traži. Osim toga, osigurava se i globalni pristup jedinstvenoj instanci - svaki put kada joj se pokuša pristupiti, dobiti će se ista instanca klase. Ovo olakšava i kontrolu pristupa u slučaju kada je neophodno da postoji samo jedan objekat određenog tipa.

U našem projektu singleton patern je iskorišten u klasi koja simulira bazu podataka i koja se kreira samo jednom.

Builder

Builder patern služi za apstrakciju procesa konstrukcije objekta, kako bi se kao rezultat mogle dobiti različite specifikacije objekta koristeći isti proces konstrukcije. Ovaj patern koristi se kako bi se izbjeglo kreiranje kompleksne hijerarhije klasa te kako bi se izbjegao kompleksni programski kod konstruktora jedne klase koja može imati različite konfiguracije atributa. Različiti dijelovi

konstrukcije objekta izdvajaju se u posebne metode koje se zatim pozivaju različitim redoslijedom ili se poziv nekih dijelova izostavlja, kako bi se dobili željeni različiti podtipovi objekta bez potrebe za kreiranjem velikog broja podklasa.

Builder pattern je ovdje iskorišten tako da se pri kreiranju studenta razlikuje prijava u kojoj student ima prijavu za cimeraj od studenta kome će se nasumično dodijeliti soba.

5 MVC