Strukturalni paterni

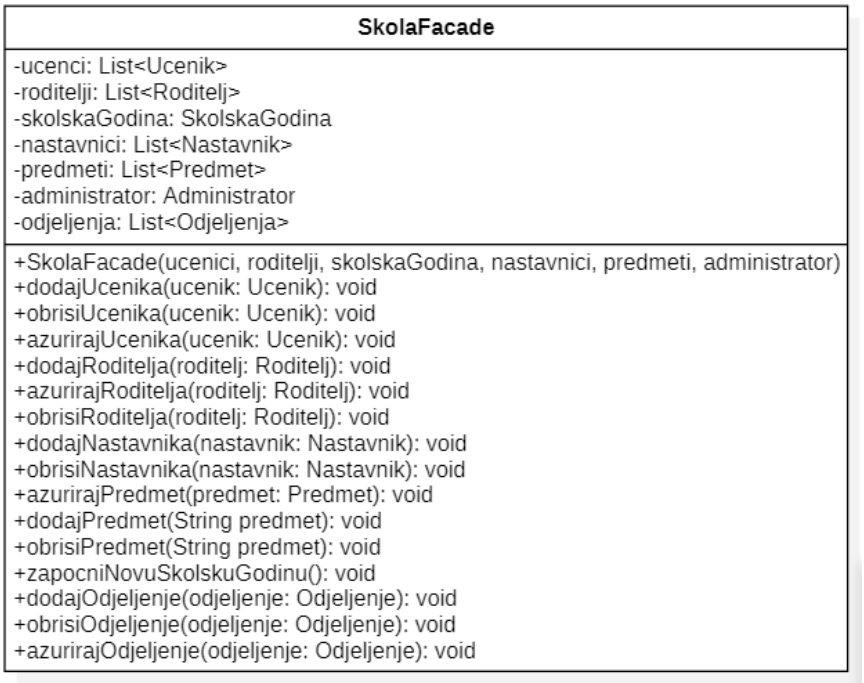
Adapter pattern

Adapter patern služi da se postojeći objekat prilagodi za korištenje na neki novi način u odnosu na postojeći rad bez mijenjanja same definicije objekta.

Ovaj patren nismo implementirali u našem sistemu. Recimo da se klasa *RoditeljskiSastanak* ne sastoji od atributa *tema* i *sadrzaj*, nego od *datuma* i *izvještaja* koji je pdf file. Dakle, u klasi *RoditeljskiSastanak* bi imali metodu *evidentiraj (izvjestaj: PDF):void*. Primjenom adapter paterna možemo omogućiti klijentu da evidentira izvještaj i u nekim drugim formatima, npr. docx bez da mijenja postojeću klasu *RoditeljskiSastanak*. Potrebno je definisati interfejs *IEvidencija* sa metodom *evidentirajDOCX():void* i klasu *RoditeljskiSastanakAdapter* koja će implementirati metodu *evidentirajDOCX* tako što će unutar nje pozvati postojeću metodu *evidentiraj* klase *RoditeljskiSastanak* nakon pretvaranja docx u pdf format.

Facade pattern (implementiran)

Ovaj patern se koristi kada sistem ima više podsistema, U našem projektu naš glavni sistem je Skola i ima podsisteme Razred, Ucenik, Odjeljenje, Nastavnik. Da bismo sakrili detalje o kreiranju i modificiranju instanca klasa podsistema, dodavanja osoba u sistem, proglasit ćemo klasu Skola Facade klasom – SkolaFacade. Ova klasa komunicira sa prethodno navedenim klasama. Korisnik sistema ne mora razmišljati šta se dešava u pozadini metoda, već je dovoljno da pozove određenu metodu iz fasadne klase.



Decorator patern

Decorator patern služi za omogućavanja različitih nadogradnji objektima koji svi u osnovi predstavljaju jednu vrstu objekta (odnosno, koji imaju istu osnovu). U našem sistemu nemamo klase čiji objekti bi se mogli nadograđivati, a za neke koji bi i mogli proširiti sa još detalja (recimo klasa Osoba, na početku mogli bi zahtijevati samo osnovne podatke kao što su ime, prezime, JMBG itd., zatim to proširiti sa brojem telefona, gmailom, zatim slikom) mogli bi to puno jednostavnije postići sa običnom set metodom.

Bridge patern

Ovaj patern nismo iskoristili u projektu. Mogli bi ga iskoristiti ukoliko bi aplikaciju koristile privatne škole gdje je potrebno i evidentirati školarinu.Privatne škole nude opcije prevoza ucenika ili produženi boravak što zahtijeva dodatne troškove. U tom slučaju potrebno bi bilo napraviti više klasa koje bi predstavljale različite tipove učenika.Svaka klasa bi implementirala interfejs koji sadrži metode za obračun školarine. Klasa Bridge bi tada na osnovni iznos školarine koji plaćaju svi učenici dodala i iznose za svakog učenika posebno.

Composite patern

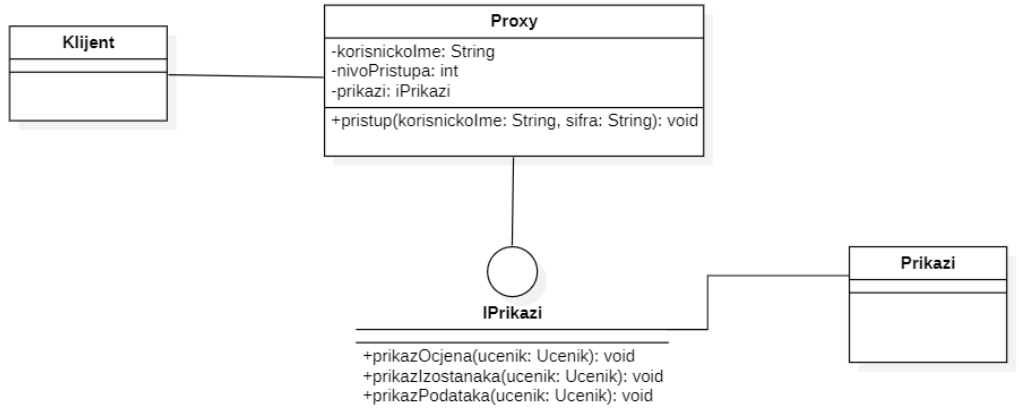
Ovaj patern nismo iskoristili u projektu. Mogli bi ga iskoristit ako je potrebno izračunati školarinu za svakog učenika posebno, na različit način. Tada je potrebno imati interfejs Iučenik koji bi ima metodu dajSkolarinu() i klasu EdnevnikComposite. Ova klasa i sve klase koje predstavljaju tip učenika bi implementovale ovaj interfejs. Metoda dajSkolarinu() u EdnevnikComposite bi obuhvatala školarine svakog učenika posebno.

Proxy pattern (implementiran)

Ovaj patern služi kako bi se kontrolisao pristup određenim objektima. Njegovom primjenom se onemogućava manipulacija objektima ukoliko neki konkretan uslov nije ispunjen. U našem projektu možemo ograničiti pristup pregledu podataka o učeniku i to na način da učenik može vidjeti isključivo svoje ocjene, podatke i izostanke. Takodjer, roditelj može vidjeti vidjeti navedene informacije samo o svom djetetu. Da bi se to realizovalo, potrebno je dodati klasu Prikazi, u kojoj će se nalaziti metode: prikazOcjena(ucenik : Ucenik) tipa void, prikazIzostanaka(ucenik: Ucenik) tipa void i prikazPodataka(ucenik: Ucenik). Ova klasa implementira interfejs iPrikazi. Zatim je potrebno dodati klasu Proxy sa atributima string username i metodom pristup (string korisnickoIme, string sifra). U ovoj metodi će se ispitati ispravnost unesenih podataka i ustanoviti vrsta korisnika i korisnicko ime.

Također, potrebno je dodati i metode prikazOcjena(ucenik: Ucenik), prikazIzostanaka(ucenik: Ucenik), prikazPodataka(ucenik: Ucenik). Ukoliko je vrsta korisnika koji pristupa Nastavnik ili

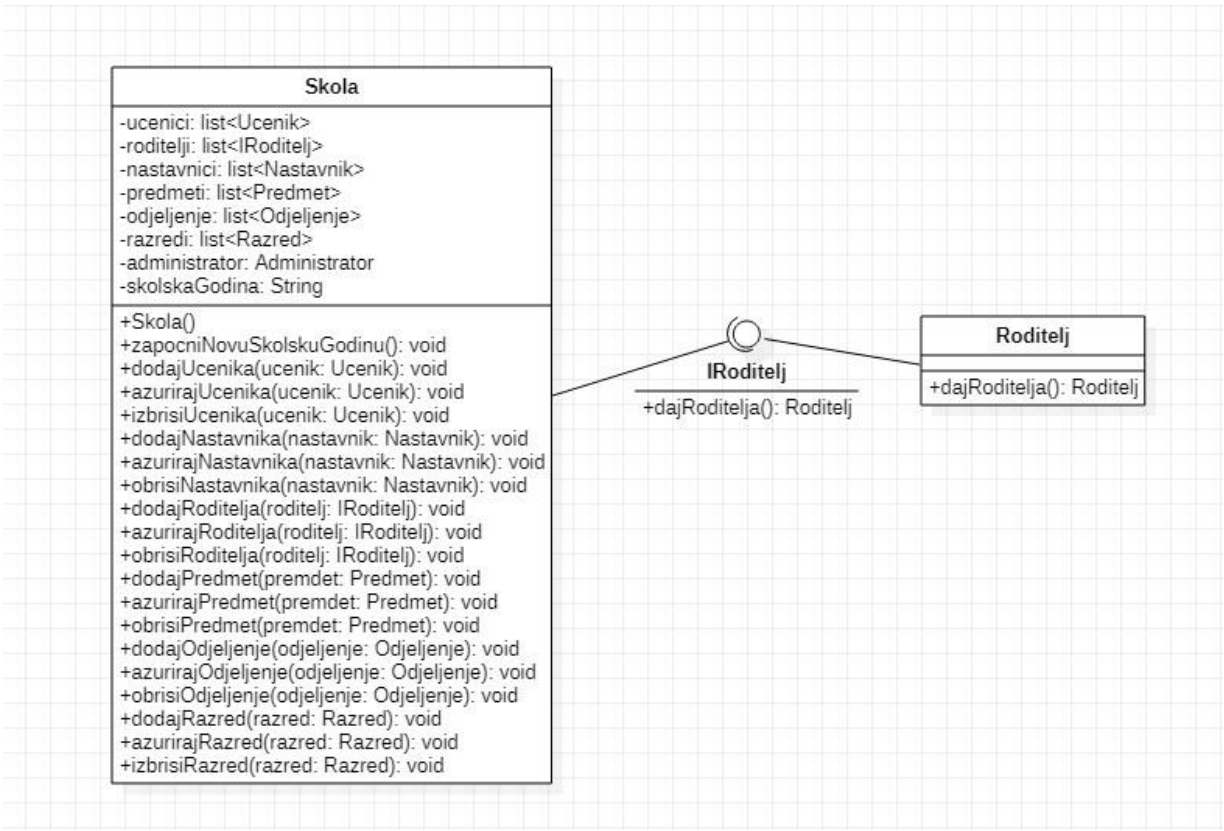
Administrator, omogućeni su mu svi prikazi. Ukoliko je vrsta korisnika Učenik provjerava se da li mu je isti username kao kod učenika kojeg prihvata metoda. Ukoliko je vrsta korisnika roditelj, provjerava se da li mu je korisničko ime jednako korisničkom imenu roditelja od učenika koji je primljen kao parametar. Ukoliko su provjere zadovoljene, poziva se odgovarajuća metoda iz klase Prikazi. Ako ipak nisu, odbija se pristup.



Flyweight pattern (implementiran)

Flyweight patern se koristi kako bi se onemogućilo suvišno stvaranje velikog broja instanci objekta koji svi u suštini predstavljaju jedan objekat.

Prilikom dodavanja novog učenika u listu u klasi *Skola*, potrebno je dodati i roditelja u listu roditelja. Međutim, roditelj može da iste školske godine ili nekad kasnije upiše još djece u tu školu. Nakon što bi dodali učenika, opet bi dodali i roditelja, tj. kreirali bi novu instancu roditelja. S obzirom da je to suvišno (jer se ona ni po čemu ne razlikuje od one prethodne) iskoristili smo Flyweight patern. Na dijagramu klasa dodan je interfejs *IRoditelj* koji sadrži metodu *dajRoditelja*():Roditelj. Klasa *Roditelj* implementira ovaj interfejs, što znači da ima metodu *dajRoditelja*(), a klasa *Skola* umjesto atributa *roditelji*:list(Roditelj) sada ima atribut *roditelji*:list(IRoditelj). Umjesto metoda *dodajRoditelja* (roditelj:Roditelj):void, *azurirajRoditelja*(roditelj:Roditelj):void i *obrisiRoditelja*(roditelj:Roditelj):void, sada su metode *dodajRoditelja* (roditelj:IRoditelj):void, *azurirajRoditelja*(roditelj:IRoditelj):void i *obrisiRoditelja*(roditelj:IRoditelj):void. Ovim bi uštedili memoriju u slučaju da postoji veći broj roditelja koji imaju više od jedno dijete.


Skola
-ucenici: list<Ucenik>
-roditelji: list<IRoditelj>
-nastavnici: list<Nastavnik>
-predmeti: list<Predmet>
-odjeljenje: list<Odjeljenje>
-razredi: list<Razred>
-administrator: Administrator
-skolskaGodina: String
+Skola()
+zapocniNovuSkolskuGodinu(): void
+dodajUcenika(ucenik: Ucenik): void
+azurirajUcenika(ucenik: Ucenik): void
+izbrisiUcenika(ucenik: Ucenik): void
+dodajNastavnika(nastavnik: Nastavnik): void
+azurirajNastavnika(nastavnik: Nastavnik): void
+obrisiNastavnika(nastavnik: Nastavnik): void
+dodajRoditelja(roditelj: IRoditelj): void
+azurirajRoditelja(roditelj: IRoditelj): void
+obrisiRoditelja(roditelj: IRoditelj): void
+dodajPredmet(premdet: Predmet): void
+azurirajPredmet(premdet: Predmet): void
+obrisiPredmet(premdet: Predmet): void
+dodajOdjeljenje(odjeljenje: Odjeljenje): void
+azurirajOdjeljenje(odjeljenje: Odjeljenje): void
+obrisiOdjeljenje(odjeljenje: Odjeljenje): void
+dodajRazred(razred: Razred): void
+azurirajRazred(razred: Razred): void
+izbrisiRazred(razred: Razred): void
IRoditelj
+dajRoditelja(): Roditelj
Roditelj
+dajRoditelja(): Roditelj