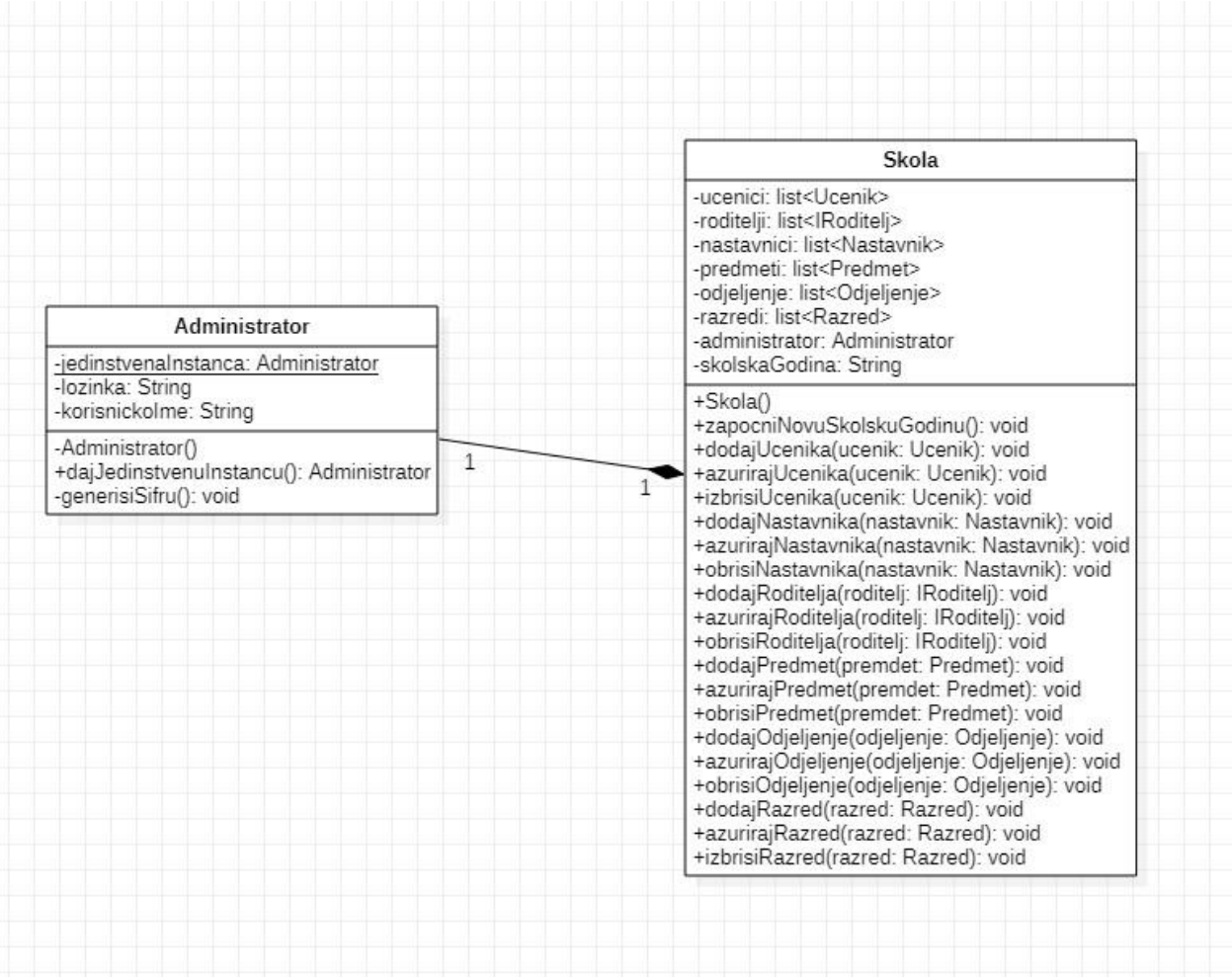
Kreacijski paterni

Singleton pattern (implementiran)

Uloga Singleton paterna je da osigura da se neka klasa može instancirati samo jednom.

U našem sistemu takva klasa je klasa *Administrator* pa smo na nju primijenili ovaj patern kako bi uštedili memoriju. Konstruktor ove klase smo proglasili privatnim, zatim smo dodali statički atribut *jedinstvenaInstanca*:Administrator i metodu *dajJedinstvenuInstancu*():Administrator preko koje se pristupa toj jedinstvenoj instanci.

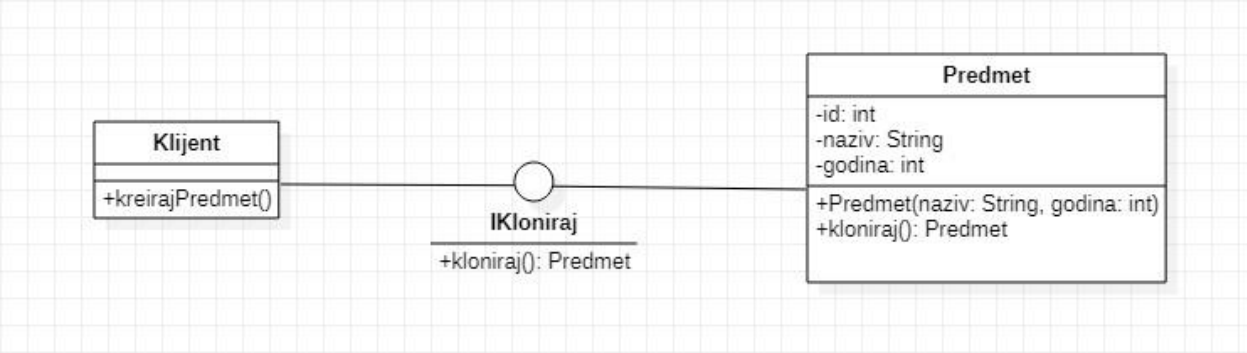


Prototype pattern (implementiran)

Uloga Prototype paterna je da kreira nove objekte klonirajući jednu od postojećih prototip instanci (postojećih objekata).

U našem sistemu ovaj patern smo primijenili na klasu *Predmet*. U toj klasi postoje tri atributa: *id*:int, *naziv*:String i *godina*:int. S obzirom na način na koji smo zamislili da funkcioniše naš sistem može postojati više predmeta koji imaju isti *naziv*, a druagčiji tip *godina*. Recimo, mogu postojati objekti čiji je *naziv* Matematika i *godina* bilo koji broj od 1 do 9.

Uz pomoć ovog paterna moguće je klonirati sve instance klase *Predmet* po istom nazivu, a zatim naknadno promijeniti atribute koji se razlikuju. Takođe, neki objekti klase *Predmet* mogu imati istu *godinu*, a različit *naziv* (svi predmeti sa iste godine). Kako bi primijenili ovaj patern dodali smo klasu *Klijent* i interfejs *IPrototype*. Klasa *Klijent* zahtjeva kloniranje postojećih objekata preko interfejsa *IKloniraj*, a klasa *Predmet* implementira kloniranje postojećih objekata.



Factory method pattern

Ovaj patern omogućava instanciranje različitih vrsta podklasa. U toku izvršavanja programa se odlučuje koja podklasa će se instancirati. Ovaj patern je moguće iskoristiti u našem projektu na način opisan u nastavku. Klase izvedene iz klase Osoba su Ucenik, Nastavnik, Roditelj. Potrebno je napraviti Creator klasu koja predstavlja klasu za pravljenje osoba. U toj klasi ćemo imati metodu FactoryMethod koja će primati sve podatke potrebne za osobu, kao i int tipOsobe koji će se odrediti na osnovu tipa korisnika koji će biti unaprijed definiran u bazi podataka. Npr. 1 – Nastavnik, 2 – Ucenik, 3 – Roditelj. Na osnovu tipa osobe će se napraviti odgovarajuća osoba. Klijent koristi FactoryMethod iz Creator klase.

Abstract factory pattern

Abstract factory pattern omogućava da se kreiraju familije povezanih objekata. Koristi se kada ne želimo da kod zavisi o konkretnim klasama tih objekata. U našem projektu ovaj patern se ne bi se mogao upotrijebiti. Hipotetički, ukoliko bi naš projekat podržavao više vrsta ocjenjivanja i školovanja (npr. po sistemu neke druge države) moglao bi se napraviti fabrički interfejs AbstractFactory i dvije konkretne fabrike za pravljenje ocjena i provjera znanja, a koje bi implementirale interfejse IOcjena i i IprovjeraZnanja. Na taj način, ovisno o sistemu školovanja ocjene i provjere znanja bi bile odgovarajuće, a da toga klijent nije svjestan.

Builder pattern

Uloga Builder paterna je odvajanje specifikacije kompleksnih objekata od njihove stvarne konstrukcije. U našem projektu, ovaj patern bi se mogao iskoristiti za pravljenje instanca klase Skola, zato što se sastoji od lista kompleksnih objekata. Klasa Product bi zapravo bila Skola. Dodali bismo interfejs iBuilder, kao i Builder klasu koja bi implementirala metode napraviUcenika(), napraviNastavnika(), napraviRoditelja(), napraviAdministratora() u kojima bi se pravile instance klasa Ucenik, Nastavnik, Roditelj i Administrator i dodavale u odgovarajuće liste instance klase Skola koju Builder klasa ima kao svoj atribut. Dodali bismo metodu dajRezultat() koja bi vraćala produkt, odnosno formiranu školu. Takodjer, napravili bismo Director klasu čiji bi konstruktor primao