Kreacijski patterni

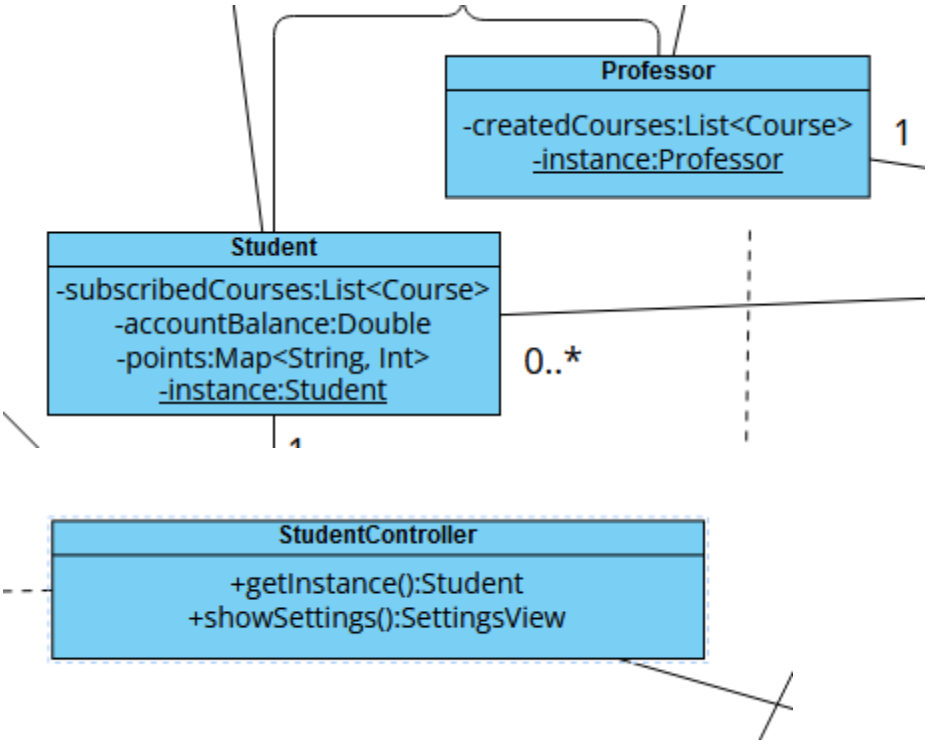
Singleton pattern:

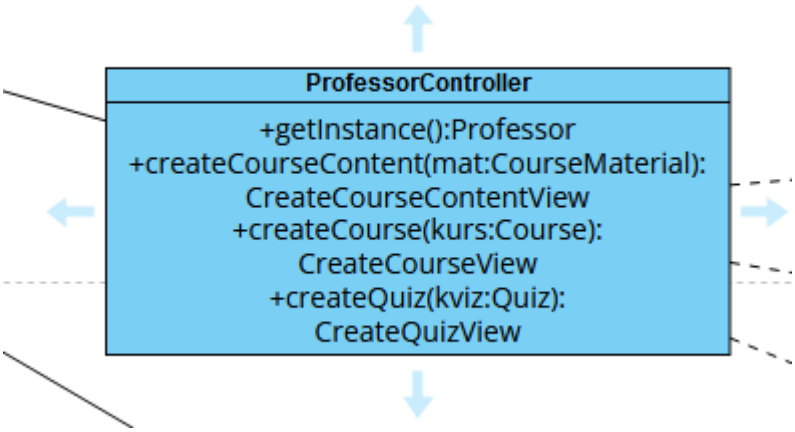
Uloga Singleton paterna je da osigura da se klasa može instancirati samo jednom i da osigura globalni pristup kreiranoj instanci klase.

Koristi se i za odgođeno (lazy) instanciranje objekata.

Primjena Singleton patterna:

U našem slučaju Singleton pattern smo primjenili kod klasa profesora i studenta, kada se user prijavi na svoj račun, kreirat će se instanca tog usera (profesora ili studenta) i sve ostale klase koje trebaju pristup tom useru će ih dobijati preko metode getInstance(). Ovim se omogućuje da postoji samo jedan objekat usera, tako da bilo kakve promjene nad njim neće imati konflikta i bit će izvršene kako treba.





Prototype pattern:

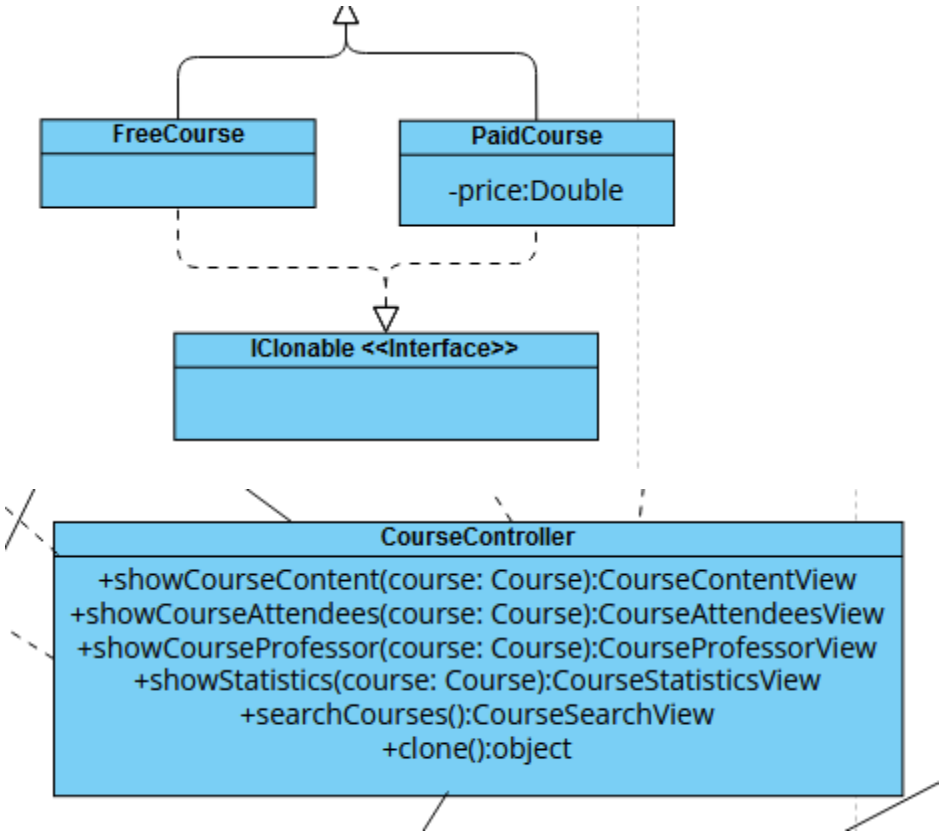
Prototype dizajn patern dozvoljava da se kreiraju prilagođeni objekti bez poznavanja njihove klase ili detalja kako je objekat kreiran.

Prototype obezbjeđuje interfejs za konstrukciju objekata kloniranjem jedne od postojećih prototip instanci i modifikacijom te kopije.

Prototype patern se koristi ako je kreiranje novog objekta resursno zahtjevno. U tom slučaju rezonski je vršiti kloniranje već postojećeg objekata.

Primjena Prototype patterna:

U našem slučaju Prototype pattern smo primjenili za slučaj da će se kursevi ponavljati. Na primjer, ako je neki kurs završen i ponovo će biti dostupan i držat će ga isti profesor, umjesto da se ponovo popunjavaju isti podaci, možemo koristiti Prototype pattern pomoću kojeg ćemo samo promjeniti spisak studenata koji će biti upisani.



Factory pattern:

Uloga Factory Method paterna je da omogući kreiranje objekata na način da podklase odluče koju klasu instancirati.

Factory Method instancira odgovarajuću podklasu (izvedenu klasu) preko posebne metode na osnovu informacije od strane klijenta ili na osnovu tekućeg stanja.

Factory patern odvaja (decoupling) zahtijevanje objekata od kreiranja objekata.

Kijent ne treba da zna koji tip objekta je kreiran.

Moguća primjena Factory patterna:

U našem slučaju Factory pattern bi se mogao iskoristiti za kreiranje kurseva zavisno da li su plaćeni ili besplatni. Factory pattern bi odlučivao koja instanca treba da se kreira.

Abstract Factory pattern:

Abstract Factory patern omogućava da se kreiraju familije povezanih objekata/produkata bez specificiranja konkretnih klasa.

Na osnovu apstraktne familije produkata kreiraju se konkretne fabrike (factories) produkata različitih tipova i različitih kombinacija.

Patern odvaja definiciju (klase) produkata od klijenta.

Zbog toga se familije produkata mogu jednostavno izmjenjivati ili ažurirati bez narušavanja strukture klijenta.

Klijent (calling code) radi sa apstraktnim interfejsima i klasama i ne zna stvarne tipove objekata kreiranih sa fabrikom.

Abstracy Factory patern se koristi kada postoji hijerarhija objekata koja enkapsulira različite platforme sastavljene od grupe povezanih objekata.

Moguća primjena Abstract Factory patterna:

U našem slučaju Abstract Factory pattern bi se mogao primjeniti kod familija kurseva, tj. da se prave različiti tipovi kurseva. Npr. online, offline, video, interaktivni kursevi, itd.

Builder pattern:

Uloga Builder paterna je odvajanje specifikacije kompleksnih objekata od njihove stvarne konstrukcije.

Isti konstrukcijski proces može kreirati različite reprezentacije.

Koristi se ako se objekti mogu podijeliti u skupove objekata koji se razlikuje samo po permutaciji njihovih dijelova.

Moguća primjena Builder patterna:

U našem slučaju Builder pattern bi se mogao primjeniti ako bismo željeli da elementi kurseva ne moraju biti isti, tj. da neki kursevi imaju određene atribute dok drugi ne. Npr. cijena, trajanje, opis itd.