Kreacijski paterni

- ## Singelton pattern

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

Naš sistem omogućuje korisnicima korištenje materijala za instrukcije. Pristup tim materijalima se vrši putem pristupa serverima koji su zaduženi za pohranu i dobavljanje materijala putem internet konekcije.
S obzirom da je za brzu, efikasnu i pouzdanu uslugu neophodno koristiti više nezavisnih servera, proširujemo sistem s klasom ServerBalanser koja će biti singleton. Njen zadatak je da upravlja svim raspoloživim serverima i za odredjeni zadatak, kao što je dobavljanje materijala, odabira optimalno rjesenje koje je moguće u smislu opterećenosti sistema i brzine dostave podataka.

Da bi ispunila singleton pravilo ServerBalanser mora imati privatni statički atribut koji je tipa ServerBalanser, privatni konstruktor i javnu pristupnu metodu odnosno getter.

- ## Prototype pattern

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

Ukoliko bi naš sistem bio proširen sa korisničkim zahtjevom da aplikacija prati rezultate korisnika studenata, kao dodatni kriterij kvalitete tutora i instrukcija bilo bi neophodno prosiriti i bazu podataka. To bi znatno oslabilo performance pristupa bazi podataka ukoliko bi bilo potrebno obrađivati podatke za potrebe statistike ili pretrage optimalnog tutora po željenim kriterijima. Zbog toga je korisno da se implementira Prototype pattern koji bi omogućio da već učitane podatke “recikliramo”, odnosno ponovno iskoristimo putem kloniranja.

Bilo bi potrebno kreirati interfejs IKorisnikPrototip koji bi omogućio kloniranje instanci klasa Tutor i Student, kojim bi upravljao klijent.

- ## Factory Method pattern

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

Sistem je moguće proširiti korisničkim zahtjevom da razlikujemo predmete za koje nudimo usluge po semestrima. Kako bi instanciranje različitih klasa predmeta bilo pojednostavljeno kreirali bi Factory Method pattern.
Kreirali bi dvije klase ZPredmet i LjPredmet za zimski i ljetni semester respektivno, koje bi implementirale interfejs IPredmet. Također neophodna je klasa Kreator koja ima metodu Fabrika koja sadrži logiku koja determinira koja će se klasa instancirati.

Konkretna usluga je obrađena u sklopu Kreator klase je pristup ECTS bodovima pojedinacnih predmeta, dajECTS() koja vraća realan broj.

● Abstract Factory pattern

Abstract Factory patern omogućava da se kreiraju familije povezanih objekata/produkata. Na osnovu apstraktne familije produkata kreiraju se konkretne fabrike (factories) produkata različitih tipova i različitih kombinacija. Patern odvaja definiciju (klase) produkata od klijenta.

Naš sistem je trenutno fokusiran samo na potrebe studenata jednog fakulteta (ETF). Ukoliko bi nekada htjeli da proširimo potencijalno tržište bilo bi potrebno da se aplikacija skalira na cijeli univerzitet, eventualno i na kompletni školski sistem.

Da bi to bilo izvodivo u budućnosti korisno je implementirati Abstract Factory pattern. Recimo za univerzitet bi kreirali interfejse za svaki fakultet kao IMedicinski, IFilozofski, IPravni, IMašinski, itd. Ovi interfejsi bi bili korišteni od strane univerziteta(recimo UNSA, BURCH, IUS,) Zatim bi se kreirale klase za specifične predmete svakog fakulteta i svi bi imali specifična imena po određenom pravilu kodiranja (za filozofski bi se zvali PredmetF1, PredmetF2; za medicinski PredmetM1, PredmetM2).

● Builder pattern

Uloga Builder paterna je odvajanje specifikacije kompleksnih objekata od njihove stvarne konstrukcije. Isti konstrukcijski proces može kreirati različite reprezentacije.

Naslanjajući se na prethodnu situaciju proširenja sistema, kreiranje različitih instanci predmeta dosta komplicira stvari jer je čest slučaj da dosta fakulteta dijeli resurse. Kako bi se logika kreiranja svih instanci objediila i zadržala na jednom mjestu uvodimo Builder pattern koji nam olakšava kreiranje kompleksnih objekata.

Morali bi imati interfejs IGraditeljPredmeta koji definira specifične metode potrebne za izgradnju predmeta, zatim klasu Direktor koja sadrži logiku i neophodne sekvence za stvaranje objekta. Dalje GraditeljPredmeta klasu, koju poziva Direktor, koja će obaviti stvarno kreiranje predmeta.

Također neophodno je imati Product klasu koja je u našem slučaju specifična klasa Predmet koja nam je potrebna.

Dalje imamo builder ili graditelj klase kao što su: GraditeljSviOdsjeci (koji kreiraju sve zajedničke predmete, metoda izgradiZajednickePredmete()), GraditeljTeoretskiPredmeti (kreira sve teoretske predmete, metoda izgradiTeoretskePredmete()), GraditeljStrucniPredmeti (za izgradnju strucnih predmeta, metoda izgradiStrucnePredmete()), GraditeljPrakticniPredmeti (za izgradnju prakticnih predmeta, metoda izgradiPrakticnePredmete()). Sve izgradi metode vraćaju listu odgovarajućih predmeta.