**KREACIJSKI DIZAJN PATERNI**

**Singleton**

Uloga Singleton paterna je da osigura da se klasa može instancirati samo jednom i da osigura globalni pristup kreiranoj instanci. Za praćenje rada našeg sistema, objekat koji je potrebno samo jednom instancirati i nad kojim je potrebna jedinstvena kontrola pristupa jeste objekat koji se koristi za logiranje.

To možemo implementirati:

* kreiranjem Logger klase;
* definiranjem privatne statičke read-only instance Logger klase;
* postavljanjem privatne vidljivosti konstruktora;
* dodavanjem javno vidljive osobine ili metode koja pristupa privatnom objektu;
* implementiranjem metode koja prima poruku i zapisuje je.

Korištenje:

Unutar sistema možemo koristiti Singleton kako bi pratili rad sistema, te olakšali rješavanje eventualnih problema.

**Prototype**

Prototype patern obezbjeđuje interfejs za konstrukciju objekata kloniranjem jedne od postojećih prototip instanci i modifikacijom te kopije. Ukoliko bi naše korisničke klase proširili dodatnim atributima koji su u većini slučajeva isti za mnoge pojedince, onda bi imalo smisla kreirati modele s predefinisanim poljima koja se često ponavljaju. Kreiranje novih instanci bi se tada svelo na kloniranje tih prototipova, te eventualno manje izmjene.

To možemo implementirati:

Prototype patern bi se implementirao tako što bi se definisali prototipovi korisnika (učenik, nastavnik, roditelj) s podrazumijevanim postavkama. Kada se kreira novi korisnik, sistem bi klonirao odgovarajući prototip i izvršio manje izmjene (ime, razred, predmet), umjesto da svaki put kreira objekat ispočetka. Ovo ubrzava proces i osigurava konzistentnost.

Korištenje:

Prototype patern se može za kreiranje korisničkih naloga (učenik, nastavnik, roditelj) na osnovu prethodno definisanih prototipova. Svaka uloga bi imala unaprijed postavljena prava pristupa, podešavanja i strukturu naloga. Prilikom dodavanja novog korisnika, sistem bi klonirao odgovarajući prototip i izvršio manje izmjene poput imena ili razreda. Na ovaj način pojednostavljuje se proces kreiranja korisnika i osigurava konzistentnost između naloga iste vrste.

**Factory Method**

Factory Method patern omogućava kreiranje objekata na način da podklase, na osnovu informacije od strane klijenta ili tekućeg stanja, odluče koju klasu instancirati. Ovaj obrazac može naći primjenu u našem sistemu pri kreiranju različitih vrsta korisnika, odnosno učenika, nastavnika, razrednika i administratora.

To možemo implementirati:

* definiranjem apstraktne bazne klase za korisnika;
* definiranjem konkretne klase za svaki specifični tip korisnika;
* definiranjem apstraktne bazne klase koja sadrži metodu za kreiranje korisnika;
* definiranjem konkretnih tvornica za svaki tip korisnika.

Korištenje:

Na primjer, prilikom kreiranja učenika, nastavnika, razrednika ili administratora, klijent ne mora voditi računa o tome iz koje konkretno klase će biti instanciran objekat. Time se uklanjaju grananja u raznim dijelovima kôda, odnosno logika odlučivanja se smješta na jedno mjesto.

**Abstract Factory**

Abstract Factory patern omogućava da se kreiraju porodice povezanih objekata bez specificiranja konkretnih klasa. Koristi se kada postoji hijerarhija objekata koja enkapsulira različite platforme sastavljene od grupe povezanih objekata. Razmatrajući naš sistem, pojava različitih varijacija osoblja (nastavnici, učenici) za različite tipove škola (osnovna, srednja) bi mogla opravdati korištenje ovog paterna. Međutim, uzimajući u obzir da je naš sistem fokusiran na jedan tip nastavnog procesa, odnosno da nema varijacija, kompleksnost koju nosi Abstract Factory patern nije opravdana.

To možemo implementirati:

Može se implementirati tako da imamo apstraktnu fabriku koja definiše interfejs za kreiranje korisnika (učenika, nastavnika, roditelja) za različite tipove škola. Na primjer, OsnovnaSkolaFactory i SrednjaSkolaFactory bi bile konkretne fabrike koje kreiraju varijante korisnika prilagođene za osnovnu ili srednju školu (npr. učenici sa različitim predmetima). Klijentski kod bi koristio odgovarajuću fabriku bez poznavanja konkretnih klasa, čime bi se omogućila fleksibilnost i lako proširenje sistema za različite obrazovne nivoe.

Korištenje:

Pošto kod nas nema opcija za više vrsta škola kao osnovna škola i srednja škola kod nas se ne može koristiti.

**Builder**

Builder dizajn patern je koristan za kreiranje složenih objekata korak po korak. Primjenjuje se kada objekat ima mnogo opcionalnih parametara ili kada je potrebna fleksibilnost u kreiranju objekta. U našem slučaju, ukoliko bi na osnovni set podataka o korisnicima sistema bilo potrebno dodati nove, ali koji nisu uvijek korišteni, upotreba Builder paterna bi bila opravdana. Na primjer, osim osnovnih podataka o učenicima (koje ima svaki učenik), kao što su ime i prezime, za pojedine učenike bi u toku školske godine bilo potrebno dodati uspjehe ostvarene na takmičenjima. Analogno tome možemo posmatrati karijerna napredovanja nastavnika.

To možemo implementirati:

Može se implementirati tako da se korisnici (npr. učenici) kreiraju korak po korak prvo osnovni podaci (ime, prezime), a zatim se po potrebi dodaju dodatne informacije. Time se omogućava fleksibilno i pregledno kreiranje objekata sa opcionalnim podacima, bez pretrpavanja konstruktora.

Korištenje:

Builder patern bi se mogao koristiti prilikom registracije korisnika kada se podaci unose postepeno npr. kod unosa učenika prvo se zadaju osnovni podaci (ime, prezime, razred), a zatim se po potrebi dodaju dodatne informacije kao što su posebni uspjesi. Patern omogućava da se ti podaci dodaju samo kad su relevantni, bez komplikovanja osnovne strukture objekta.