DESIGN PATTERNS

KREACIJSKI

1. SINGLETON PATTERN

Uloga Singleton patterna je da osigura da se klasa može instancirati samo jednom i da osigura globalni pristup kreiranoj instanci klase. Koristi se za objekte koje je potrebno samo jednom instancirati i nad kojim je potrebna jedinstvena kontrola pristupa.

U našem sistemu bi ovaj pattern mogli primijeniti pri registraciji korisnika na sistem. Pri tome je zamišljeno da se bilježe i pamte pristupi sistemu kao i osnovni opisi aktivnosti korisnika. Ovo je najbolje obaviti kroz Logger klasu koja će biti singleton i čija će jedna instanca bilježiti potrebne informacije. Važno je da bude kreirana samo jedna instanca kako se ne bi otvaralo više log fajlova i tako se izbjegli problemi konkurentnog pristupa istom fajlu a i uštedila memorija. Tako će biti omogućen globalni pristup Logger klasi, odnosno bilo koja klasa će moći pristupiti instanci te Logger klase i preko nje dobiti sve željene informacije u svrhu analize posjećenosti ili pronalaska greške pri radu.

2. FACTORY METHOD PATTERN

Uloga Factory Method patterna je da omogući kreiranje objekata na način da podklase odluče koju klasu instancirati. Različite podklase mogu na različite načine implementirati interfejs.

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

U našem sistemu bi ovaj pattern mogli primijeniti ukoliko odlučimo da imamo više vrsta recepata koje će korisnik moći izabrati, npr. ReceptDoručak, ReceptRučak, ReceptVečera, pa bi bilo potrebno napraviti te klase i one bi implementirale interfejs IRecept sa metodama za prikaz i filtriranje, a onda bi svaka od 3 nove klase vrsta recepata na drugačiji način implementirale te metode. Pored njih trebamo napraviti i klasu Kreator koja bi imala Factory metodu koja bi na osnovu logike koju sadrži određivala koju će klasu instancirati, a logika za određivanje u našem sistemu bi bila da se na osnovu doba dana instancira odgovarajuća klasa i ponude korisniku recepti za nastojeći obrok.

3. ABSTRACT FACTORY PATTERN

Uloga Abstract Factory patterna je da omogući da se kreiraju familije povezanih objekata/produkata. Na osnovu apstraktne familije produkata kreiraju se konkretne fabrike (factories) produkata različitih tipova i različitih kombinacija.

U našem sistemu bi ovaj pattern mogli primijeniti kada korisnik bude vršio filtriranje vježbi ili recepata togom pretrage. Na osnovu filtera bi se kreirala fabrika objekata različitih tipova vježbi ili recepata. Time bi se njihov prikaz učinio efikasnijim jer bi Abstract Factory pattern upravlja familijama objekata i čuva njihove detalje neovisnim od klijenta.

4. Prototype pattern

Uloga Prototype patterna je da kreira nove objekte klonirajući jednu od postojećih prototip instanci (postojeći objekat). Ako je trošak kreiranja novog objekta velik i kreiranje objekta je resursno zahtjevno tada se vrši kloniranje već postojećeg objekata. Prototype dizajn patern dozvoljava da se kreiraju prilagođeni objekti bez poznavanja njihove klase ili detalja kako je objekat kreiran.

U našem sistemu najkompleksnija je klasa FitnesProgram koja vrši kreiranje programa na osnovu analize korisnikovog profila. Ovaj pattern bi se mogao iskoristiti u svrhu da se jedan takav program kreira prvi put, a kasnije klonira i modifikuje, jer korisnik obično nema potrebu mijenjati sve ili većinu podataka o sebi, već neke pojedinačne detalje sklone čestim promjenama kao što je tjelesna masa. U tom slučaju bi bilo potrebno implementirati interfejs IProgramPrototip koji bi omogućio kloniranje instanci klase FitnesProgram.

5. Builder pattern

Uloga Builder paterna je odvajanje specifikacije kompleksnih objekata od njihove stvarne konstrukcije. On služi za apstrakciju procesa konstrukcije objekata, kako bi se kao rezultat mogle dobiti različite specifikacije objekta koristeći isti proces kontrukcije. Isti konstrukcijski proces može kreirati različite reprezentacije.

U našem sistemu bi se mogao iskoristiti za kreiranje objekata tipa Korisnik jer konstruktor te klase sadrži veliki broj parametara i ako bi u budućnosti htjeli dodati još dodatnih osobina odnosno atributa korisnika došlo bi do komplikacija i kontruktor bi imao prevelik broj parametara, od kojih neki možda ne bi bili iskorišteni. Koristeći ovaj pattern kreiranje objekata bi obavljali u klasi KorisnikBuilder, što bi olakšalo i dodavanje novih atributa u klasu Korisnik ukoliko bi bilo potrebno. Potreban je i interfejs IKorisnikBuilder u kojem bi se definirale metode za izgradnju korisnika (postavljanje atributa na osnovu registracije odnosno prijave korisnika) kao i Director klasa koja definiše redoslijed pozivanja metoda za izgradnju čija se implementacija nalazi u KorisnikBuilder. IKorisnikBuilder bi implementirale klase Director i KorisnikBuilder.