Kreacijski paterni

1. Singleton pattern

Singleton pattern, jedan od najjednostavnijih patterna dizajna, osigurava da se klasa moze instancirati samo jednom te da se kreiranoj instanci klase moze pristupiti na globalnom nivou.

Prvenstveno, da bi se osigurao najefikasniji pristup samoj aplikaciji, potrebno je primijeniti Singleton pattern prilikon samog logovanja na aplikaciju. Da bismo to uradili potrebno je kreirati SingletonLog klasu koja sadrzi privatni konstruktor i statički privatni konstruktor.

Obzirom da nasa aplikacija posjeduje samo jedan sveobuhvatni forum pzeljno bi bilo kreirati SingletonForum klasu koja bi osiguravala jedinstveno postojanje klase Forum. Prikladno Singleton-pattern sintaksi klasa bi u sebi sadrzala privatni i privatni staticki konstruktor, privatni getter, kao i privatni static read-only objekat koji se interno instancira korištenjem privatnog konstruktora.

2. Prototype pattern

Da bismo sveli broj klasa u sistemu na minimum i da bismo izbjegli višestruko kreiranje novih objekata pozeljno je koristenje Prototype patterna. Ukoliko je trošak kreiranja novog objekta velik i zahtijeva ogromne resurse moguce je izvrsiti kloniranje već postojeceg objekta. U slucaju da su klase Korisnik i Pas po strukturi malo slicnije moglo bi se uz odredjene izmjene atributa, koristenjem interfejsa izbjeci visestruko kreiranje i na taj nacin smanjiti potrebne resurse. Takodjer, prilikom kreiranja podsjetnika moguce je koristenjem interfejsa IPodsjetnik omoguciti kloniranje instanci te klase uz izmjenu atributa sa proizvoljnim vrijednostima.

3. Factory method

Uloga Factory Method patterna je da omoguci kreiranje objekata na nacin da podklase odluce koju klasu instancirati.

Obzirom na primitivnost naseg projekta i postojanje klase Dokument moguce je kreiranjem podklasa koje predstavljaju razlicite tipove dokumenata (pdf, docx, txt,..) koje app podrzava omoguciti korisniku upload-ovanje istih na jednostavan nacin. Odluku o tome koja vrsta dokumenta se ucitava donosi se analiziranjem ekstenzija dokumenta. Podklase na osnovu ekstenzije odlucuju koja ce se klasa instancirati.

4. Apstract 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. Zbog toga se familije produkata mogu jednostavno izmjenjivati ili ažurirati bez narušavanja strukture klijenta Sto se tice naseg projekta, mogle bismo kreirati apstraktnu fabriku TipsFactory, iz koje bi bile naslijedjene konkretne fabrike FoodTipsFactory, HealthTipsFactory... Klase bi predstavljale razlicite probleme ili savjete u zavisnosti od vrste Tips-a (savjeta). Za HealthTipsFactory imale bismo odredjene klase koje su vezane za bolesti (npr. gojaznost, bolesti jetre, anksioznost pasa...). Na taj nacin klijent ima bolji pregled svih savjeta, a adminu je lakse dodati/obrisati odredjene savjete koje on dodaje.

5. Builder pattern

Builder patern Uloga Builder paterna je odvajanje specifikacije kompleksnih objekata od njihove stvarne konstrukcije. Isti konstrukcijski proces može kreirati različite reprezentacije. U nasem projektu, Builder patern bi se mogao iskoristiti kod pravljenja podsjetnika. Korisnik sam upise vazne dogadjaje (vakcinacija, trening, kastracija...), te to moze predstavljati gradivne elemente podsjetnika. Postojao bi interfejs IBuilder i lista podsjetnika.