

Kreacijski paterni

1. Singleton pattern

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

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

Obzirom da naša aplikacija posjeduje samo jedan sveobuhvatni forum poželjno bi bilo kreirati SingletonForum klasu koja bi osiguravala jedinstveno postojanje klase Forum. Prikladno Singleton-pattern sintaksi klasa bi u sebi sadržala privatni i privatni statički 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 poželjno je korištenje Prototype patterna. Ukoliko je trošak kreiranja novog objekta velik i zahtijeva ogromne resurse moguće je izvršiti kloniranje već postojećeg objekta. U slučaju da su klase Korisnik i Pas po strukturi malo sličnije moglo bi se uz određene izmjene atributa, korištenjem interfejsa izbjeći višestruko kreiranje i na taj način smanjiti potrebne resurse. Također, prilikom kreiranja podsjetnika moguće je korištenjem interfejsa IPodsjetnik omogućiti kloniranje instanci te klase uz izmjenu atributa sa proizvoljnim vrijednostima.

3. Factory method

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

Obzirom na primitivnost našeg projekta i postojanje klase Dokument moguće je kreiranjem podklasa koje predstavljaju različite tipove dokumenata (pdf, docx, txt,..) koje app podržava omogućiti korisniku upload-ovanje istih na jednostavan način. Svaka od tih klasa bi implementirala interfejs Idokument. Odluku o tome koja vrsta dokumenta se učitava se donosi analiziranjem ekstenzija dokumenta tj. posredstvom određene metode klase Creator koja ulazi u strukturu Factory patterna. Na taj način je zadovoljena sintaksa tj. uključeni su svi elementi Factory patterna ali i pojednostavljeno instanciranje klasa različitih tipova dokumenata.

4. Abstract Factory pattern

Abstract Factory pattern omogućava da se kreiraju familije povezanih objekata/produkata. Na osnovu apstraktne familije produkata kreiraju se konkretne fabrike (factories) produkata različitih tipova i različitih kombinacija. Pattern odvaja definiciju (klase) produkata od klijenta. Zbog toga se familije produkata mogu jednostavno izmjenjivati ili ažurirati bez narušavanja strukture klijenta. Što se tiče našeg projekta, mogle bismo kreirati apstraktnu fabriku TipsFactory, iz koje bi bile naslijedjene konkretne fabrike FoodTipsFactory, HealthTipsFactory... Klase bi predstavljale različite probleme ili savjete u zavisnosti od vrste Tips-a (savjeta). Za HealthTipsFactory imale bismo određene klase koje su vezane za bolesti (npr. gojaznost, bolesti jetre, anksioznost pasa...). Na taj način klijent ima bolji pregled svih savjeta, a adminu je lakše dodati/obrisati određene savjete koje on dodaje.

5. Builder pattern

Builder pattern Uloga Builder patterna je odvajanje specifikacije kompleksnih objekata od njihove stvarne konstrukcije. Isti konstrukcijski proces može kreirati različite reprezentacije. U našem projektu, Builder pattern bi se mogao iskoristiti kod pravljenja podsjetnika. Korisnik sam upiše

vazne dogadjaje (vakcinacija, trening, kastracija...), te to moze predstavljati gradivne elemente podsjetnika. Postojao bi interfejs IBuilder i lista podsjetnika.