KREACIJSKI PATERNI

1. SINGLETON PATERN

Singleton patern služi kako bi se neka klasa mogla instancirati samo jednom. Na ovaj način može se omogućiti i tzv. lazy initialization, odnosno instantacija klase tek onda kada se to prvi put traži. Osim toga, osigurava se i globalni pristup jedinstvenoj instanci - svaki put kada joj se pokuša pristupiti, dobiti će se ista instanca klase. Ovo olakšava i kontrolu pristupa u slučaju kada je neophodno da postoji samo jedan objekat određenog tipa

- ### Primjer upotrebe Singleton paterna:

  Što se tiče Singleton paterna, primjer za njegovu upotrebu u našem program smo uočili kod klase Administrator. Naime, korisnik posjeduje mogućnost pristupa našem ITShopu, kojem pored korisnika pristupaju uposlenici i jedan administrator, koji ima punu moć nad ITShopom. Naprimjer, administrator ITShopa je vlasnik ITShopa, koji se ne može promijeniti sve dok on to sam ne odluči.

- ### Kako bi se to postiglo, potrebno je izvršiti sljedeće:

  1. Dodati statički atribut u klasu AdministratorSingleton tipa AdministratorSingleton, koji se označava kao statički;

  2. Dodati statičku metodu dajAdministratora () u klasi AdministratorSingleton koja će vršiti vraćanje jedinstvenog statičkog atributa iz ove klase;

  3. Dodati metodu postaviAdministratora() u klasu AdministratorSingleton koja će kontrolisati promjenu administatora ukoliko za to bude potrebno.

2. PROTOTYPE PATERN

Prototype patern omogućava smanjenje kompleksnosti kreiranja novog objekta tako što se uvodi operacija kloniranja. Na taj način prave se prototipi objekata koje je moguće replicirati više puta a zatim naknadno promijeniti jednu ili više karakteristika, bez potrebe za kreiranjem novog objekta nanovo od početka. Ovime se osigurava pojednostavljenje procesa kreiranja novih instanci, posebno kada objekti sadrže veliki broj atributa koji su za većinu instanci isti.

- Primjer upotrebe Prototype paterna:

  Što se tiče Prototype paterna, primjer za njegovu upotrebu smo uočili između klasa Kupovina i Uposlenik. Naime, nakon kupovine, uposlenik mora da reguliše količine kupljenih proizvoda koje su trenutno na stanju, te smatramo da bi upotreba ovog paterna smanjila pristup bazi podataka.

- Kako bi se to postiglo, potrebno je izvršiti sljedeće:

  1. Definisati interfejs IPrototip koji se sastoji od metode kloniraj();

  2. Naslijediti interfejs IPrototip od klase Kupovina te implementirati metodu kloniraj() koja će kreirati duboku kopiju objekta;

  3. Promijeniti implementaciju uposlenika tako da se vrši kloniranje proizvoda pri njegovom brisanju.

3. FACTORY METHOD PATERN

Factory method patern služi za omogućavanje instanciranje različitih vrsta podklasa koristeći factory metodu koja odlučuje koja će se podklasa instancirati i koja programska logika izvršiti. Na ovaj način

osigurava se ispunjavanje O SOLID principa, jer se kod za kreiranje objekata različitih naslijeđenih klasa ne smješta samo u jednu zajedničku metodu, već svaka podklasa ima svoju logiku za instanciranje željenih klasa, a samo instanciranje kontroliše factory metoda koju različite klase implementiraju na različit način.

- Primjer upotrebe Factory Method paterna:

  Što se tiče Factory Method paterna, primjer za njegovu upotrebu smo uočili prilikom same registracije, jer na osnovu informacija od strane korisnika moguće je instancirati objekte različitih tipova (Kupac, Uposlenik).

- Kako bi se to postiglo, potrebno je izvršiti sljedeće:

  1. Definisati interfejs za registariju – Iregistracija;

  2. Definisati klase RegistracijaKupac, RegistracijaUposlenik koje implementiraju interfejs;

  3. Definisati klasu Creator koja posjeduje FactoryMethod() metodu koja odlučuje koju klasu instrancirati

4. ABSTRACT FACTORY PATERN

Abstract factory patern služi kako bi se izbjeglo korištenje velikog broja if-else uslova pri kreiranju različitih hijerarhija objekata. Ukoliko postoji više tipova istih objekata te različite klase koriste različite podtipove, te klase postaju fabrike za kreiranje objekata zadanog podtipa bez potrebe za specificiranjem pojedinačnih objekata. Na ovaj način se, korištenjem nasljeđivanja, ukida potreba za postojanjem if-else uslova jer određeni tip fabrike sadrži određene tipove objekata i zna se tačno koju podklasu će instancirati.

- Primjer upotrebe Abstract Factory paterna:

  Što se tiče Abstract Factory paterna još uvijek nismo sigurni za primjer za njegovu upotrebu.

- Kako bi se to postiglo, potrebno je izvršiti sljedeće:

  /

5. BUILDER PATERN

Builder patern služi za apstrakciju procesa konstrukcije objekta, kako bi se kao rezultat mogle dobiti različite specifikacije objekta koristeći isti proces konstrukcije. Ovaj patern koristi se kako bi se izbjeglo kreiranje kompleksne hijerarhije klasa te kako bi se izbjegao kompleksni programski kod konstruktora jedne klase koja može imati različite konfiguracije atributa. Različiti dijelovi konstrukcije objekta izdvajaju se u posebne metode koje se zatim pozivaju različitim redoslijedom ili se poziv nekih dijelova izostavlja, kako bi se dobili željeni različiti podtipovi objekta bez potrebe za kreiranjem velikog broja podklasa.

- Primjer upotrebe Builder paterna:

  Što se tiče Builder paterna, primjer za njegovu upotrebu smo uočili kod sastavljanja idealnog računara. Odlučili smog a iskoristiti kod klase Računar jer bi njegovo korištenje znatno smanjilo komplikovanost ove klase, te omogućili bismo jednostavno korištenje različitih dijelova procesa konstrukcije računara za dobivanje različitih rezultnih proizvoda.

- Kako bi se to postiglo, potrebno je izvršiti sljedeće:

    1. Dodati interfejs IBuilder koji će omogućiti implementaciju različitih metoda za različite načine građenja objekata računara;

    2. Dodati klase ManuelniBuilder i AutomatskiBuilder koje će na različit način implementirati metode građenja računara;

    3. Povezati kupca sa interfejsom IBuilder, a ne direktno sa klasom Računar, jer on neće direktno graditi ovaj objekat.