Kreacijski paterni

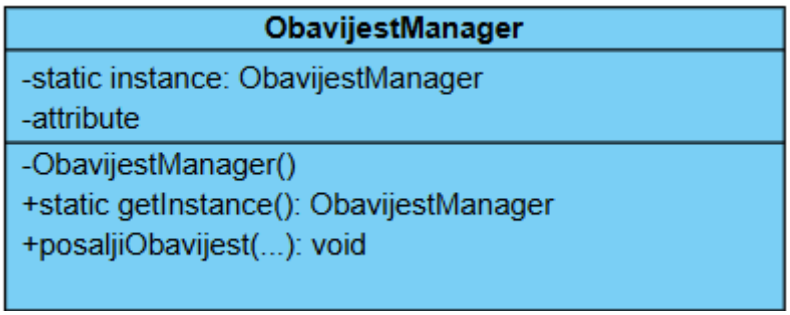
1. *Singleton pattern*

Uloga Singleton paterna je da osigura da se klasa može instancirati samo jednom i da osigura globalni pristup kreiranoj instanci.

Naš sistem ima funkcionalnost slanja obavještenja korisnicima o statusu narudžbe putem email-a. Potreban nam je jedan centralizovani sistem za slanje obavještenja koji će koordinirati sve email poruke i osigurati da se ne šalju duplirane poruke.

To ćemo implementirati sa ObavijestManager klasom gdje ćemo imati:

- **private static instance** - čuva jednu/jedinstvenu instancu klase
- **privatni konstruktor** - sprečava kreiranje novih instanci spolja
- **static getInstance() metoda** - preko koje se pristupa Singleton klasi
- **poslovne metode** - funkcionalnosti specifične za obavještenja



2. *Prototype pattern*

Prototype dizajn patern dozvoljava da se kreiraju prilagođeni objekti bez poznavanja njihove klase ili detalja kako je objekat kreiran. Prototype obezbjeđuje interfejs za konstrukciju objekata kloniranjem jedne od postojećih prototip instanci i modifikacijom te kopije. Koristi se kada je kreiranje objekta skupo ili kompleksno. U tom slučaju rezonski je vršiti kloniranje već postojećeg objekta, umjeto njihovog kreiranja od nule.

Naš sistem ima mogućnost ponovnog naručivanja narudžbe iz sekcije za praćenje historije narudžbi. Ovdje možemo koristiti ovaj patern jer korisnik može poručiti istu narudžbu ili modificirati istu, što direktno odgovara kloniranju objekata.

3. *Factory Method*

Factory Method patern omogućava kreiranje objekata na način da podklase odluče koju klasu instancirati na osnovu informacije od strane klijenta ili tekućeg stanja. Različite podklase mogu na različit način implementirati interfejs za kreiranje.

U našem sistemu postoji jasna potreba za kreiranje različitih tipova proizvoda (Hrana i Pice) na fleksibilan način. Umjesto da imamo složene blokove grananja svugdje u kodu, Factory Method centralizuje logiku kreiranja.

To se može implementirati kao:

- **IProduct** - interfejs za proizvode koji može sadržavati npr. metode getNaziv(), getCijena(), getOpis()
- **Creator** - ProizvodFactory koja može sadržavati metodu kreirajProizvod(): Proizvod, koja odlučuje koju klasu instancirati
- **ConcreteProductA, ConcreteProductB** - klase koje implementiraju interfejs **Hrana, Pice**
- **ConcreteCreatorA, ConcreteCreatorB** - klase koje predstavljaju **HranaFactory, PiceFactory**

4. *Abstract Factory*

Abstract Factory patern omogućava da se kreiraju familije povezanih objekata bez specificiranja konkretnih klasa. Koristi se kada postoji hijerarhija objekata koja enkapsulira različite platforme sastavljene od grupe povezanih objekata.

Naš sistem ima jednu konzistentnu ponudu bez različitih stilova. Svi proizvodi spadaju u istu kategoriju - standardna kafić ponuda. Kompleksnost Abstract Factory pattern-a nije opravdana jer nema potrebe za kreiranjem različitih familija proizvoda koje moraju raditi zajedno.

5. *Builder pattern*

Builder patern omogućava postepeno kreiranje složenih objekata korak po korak. Njegova uloga je odvajanje specifikacije kompleksnih objekata od njihove stvarne konstrukcije. Posebno je koristan kada objekat ima mnogo opcionalnih parametara i koristi se kada se objekti mogu podijeliti u skupove objekata koji se razlikuju samo po permutaciji njihovih dijelova.

U našem sistemu narudžbe mogu biti veoma složene - sadrže različite proizvode, dodatke, načine dostave, specijalne zahtjeve, itd. Builder patern omogućava elegantno kreiranje ovakvih složenih narudžbi.

To ćemo implementirati na sljedeći način:

- **Director** - klasa sadrži neophonu sekvencu operacija za izgradnju produkta, **NarudzbaDirector**
- **IBuilder** interfejs - deklaracija konstrukcijskih koraka produkata koji su zajednički za sve tipove bildera, **INarudzbaBuilder**
- **ConcreteBuilder** - obezbjeđuje različite implementacije za konstrukcijske korake, **NarudzbaBuilder**
- **Product** – rezultirajući objekat - > **Narudzba**