Kreacijski paterni

1. Singleton pattern

Singleton pattern se koristi kada želimo osigurati da postoji samo jedna instanca određene klase u cijeloj aplikaciji. Međutim, u našem sistemu ne vidimo jasnu potrebu za Singleton patternom jer ne postoji entitet ili komponenta koja bi zahtijevala ograničenje na jednu instancu. Ipak, možemo zamisliti situaciju u kojoj bismo imali klasu "Admin" koja predstavlja administratora sistema, i željeli bismo osigurati da postoji samo jedna instanca tog administratora.

Implementiranjem Singleton patterna za klasu "Admin", možemo osigurati globalni pristup toj instanci, omogućavajući drugim dijelovima naše aplikacije jednostavnu interakciju bez potrebe za stvaranjem više instanci ili upravljanjem njihovim stanjem.

Važno je pažljivo razmotriti da li upotreba Singleton patterna odgovara specifičnim zahtjevima naše aplikacije. Trebamo procijeniti da li je imati samo jednu instancu "Admin" klase suštinski važno za funkcionalnost našeg sistema i da li donosi moguća ograničenja ili nedostatke. U našem trenutnom sistemu, gdje nema jasne potrebe za Singleton patternom, možemo zaključiti da ga ne trebamo primijeniti.

2.Prototype pattern

Prototip pattern je izuzetno koristan u situacijama kada želimo generisati nove objekte sa sličnom strukturom kao već postojeći objekti, ali sa nekim individualnim razlikama.

U našem slučaju, kada se radi o receptima, prototip obrazac nam omogućava da kreiramo predloške recepata koje možemo klonirati i prilagođavati za svakog korisnika. Kada se recept obnavlja, gotovo sve informacije ostaju nepromijenjene, osim datuma. Kloniranje postojećeg recepta omogućava nam da zadržimo sve originalne informacije i samo promijenimo datume, čime se osigurava konzistentnost i brža izrada novih recepata za naše korisnike.

Ova fleksibilnost i brzina pružaju nam efikasan sistem upravljanja receptima, smanjujući mogućnost greške i omogućavajući nam brzo ažuriranje i prilagođavanje recepata prema potrebama svakog korisnika.

3.Factory method pattern

Factory method pattern je koristan u situacijama kada želimo da odvojimo logiku kreiranja objekata od koda koji ih koristi. U kontekstu naše aplikacije gdje imamo različite vrste osoblja kao što su doktori, farmaceuti i administratori, fabrička metoda nam omogućava da elegantno kreiramo objekte osoblja, ovisno o njihovim specifičnim ulogama.

Koristeći factory method za kreiranje objekata klase "Osoblje", možemo izbjeći ovisnost na konkretne implementacije i umjesto toga koristiti apstraktne fabričke klase ili interfejse. Naša fabrička metoda, neka

je "OsobljeFactory", deklariše apstraktnu metodu "kreirajOsoblje" koju će implementirati konkretne fabričke klase za svaku vrstu osoblja.

Na primjer, kada trebamo kreirati objekat doktora, klijentski kod jednostavno poziva metodu "kreirajOsoblje" na instanci "OsobljeFactory". Factory specifičan za doktore će tada izvršiti logiku kreiranja i konfiguracije objekta doktora, a klijentski kod ne mora biti svjestan tih detalja.

Upotreba factory method za kreiranje objekata osoblja omogućava nam da odvojimo logiku kreiranja od ostatka koda, što olakšava održavanje i proširivanje sistema. Također, omogućava nam da fleksibilno dodajemo nove vrste osoblja u budućnosti, bez mijenjanja postojećeg koda koji koristi fabričku metodu.

4.Abstract factory pattern

Abstract factory pattern je koristan u našem projektu za implementaciju različitih načina plaćanja. Kako bismo omogućili korisnicima da biraju između različitih načina plaćanja, kao što su plaćanje kreditnom karticom, plaćanje pouzećem ili plaćanje preko recepta, želimo koristiti abstract factory.

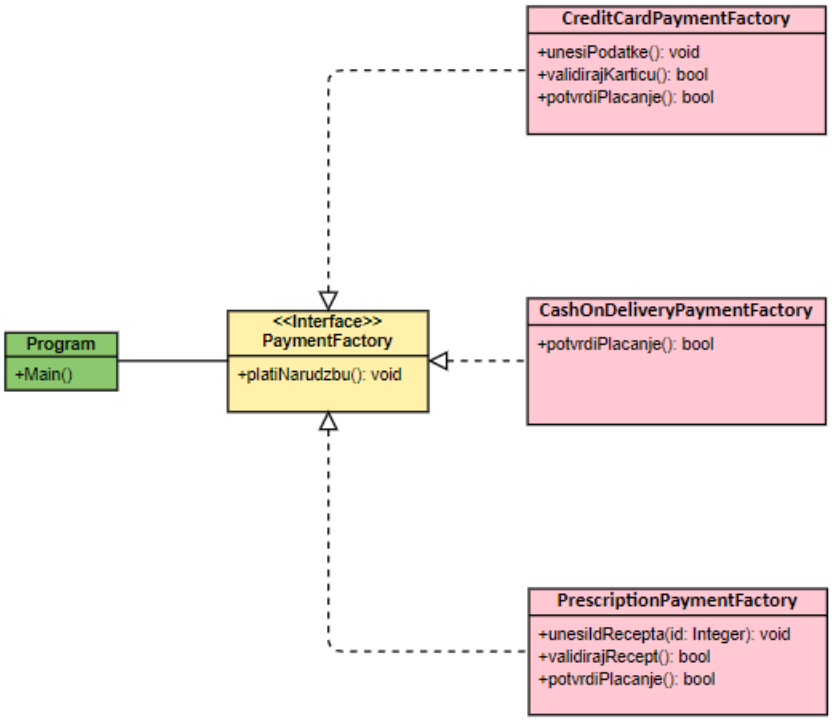
Kreirat ćemo apstraktnu fabriku "PaymentFactory" koja će definirati zajednički interfejs za kreiranje objekata plaćanja. Ova fabrika će sadržavati metodu "platiNarudzbu" koja će biti odgovorna za kreiranje odgovarajućih objekata plaćanja.

Implementirat ćemo konkretne fabrike koje nasljeđuju apstraktnu fabriku "PaymentFactory" i pružaju implementaciju za kreiranje specifičnih objekata plaćanja. Na primjer, imat ćemo "CreditCardPaymentFactory" koja implementira interfejs "PaymentFactory" i pruža logiku za kreiranje objekata plaćanja kreditnom karticom. Slično tome, imat ćemo "CashOnDeliveryPaymentFactory" i "PrescriptionPaymentFactory".

Abstract factory pattern nam omogućava da odvojimo klijentski kod od specifičnih implementacija plaćanja. Klijent samo treba raditi sa apstraktnim interfejsom "PaymentFactory" i ne treba znati konkretne klase plaćanja. Ovo olakšava promjenu između različitih načina plaćanja ili uvođenje novih, bez uticaja na klijentski kod.

Korištenjem abstract factory patterna za kreiranje objekata plaćanja, možemo održavati konzistentan interfejs za rad sa različitim načinima plaćanja, upravljati logikom kreiranja u odvojenim konkrentim fabrikama i lako dodavati ili mijenjati načine plaćanja u budućnosti.

Implementirali bi na sljedeći način:



5.Builder pattern

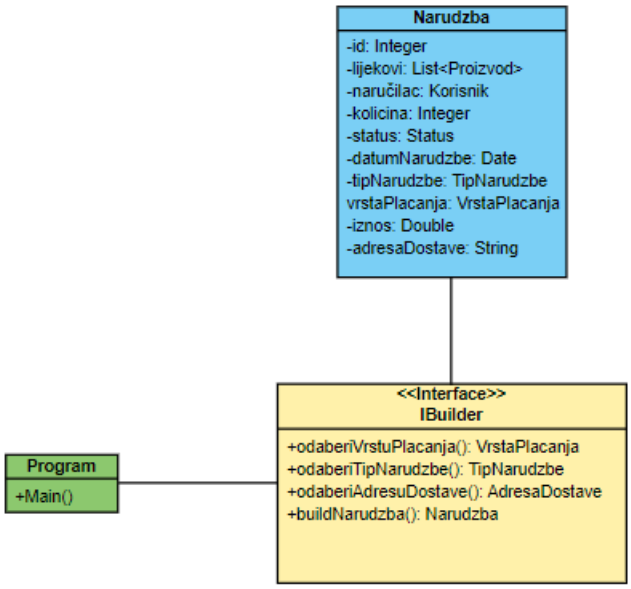
Builder pattern je idealan za generisanje instanci klase Narudzba u našem projektu.

Klasa Narudzba ima više atributa kao što su lijekovi, naručilac, količina, vrsta plaćanja i adresa dostave. Umjesto da koristimo konstruktor sa mnogo parametara, Builder pattern nam omogućava postepenu izgradnju objekta Narudzba tako što postavljamo atribute korak po korak.

Možemo kreirati klasu NarudzbaBuilder koja pruža metode za postavljanje svakog atributa pojedinačno, kao što su setLijekovi, setNaručilac, setKolicina, itd. Ovaj pristup omogućava jasan i čitljiv proces izgradnje, gdje možemo postavljati atribute u bilo kojem redoslijedu i čak preskočiti opciona polja. Na kraju, možemo pozvati metodu build() koja vraća instancu klase Narudzba sa specificiranim vrijednostima.

Korištenje Builder patterna za kreiranje instanci klase Narudzba poboljšava čitljivost koda, olakšava rukovanje opcionim atributima i omogućava fleksibilnost u dodavanju ili mijenjanju procesa konstrukcije u budućnosti.

Implementirali bi ga na sljedeći način:


Narudzba
-id: Integer
-lijekovi: List<Proizvod>
-naručilac: Korisnik
-kolicina: Integer
-status: Status
-datumNarudzbe: Date
-tipNarudzbe: TipNarudzbe
vrstaPlacanja: VrstaPlacanja
-iznos: Double
-adresaDostave: String
<<Interface>>
IBuilder
+odaberiVrstuPlacanja(): VrstaPlacanja
+odaberiTipNarudzbe(): TipNarudzbe
+odaberiAdresuDostave(): AdresaDostave
+buildNarudzba(): Narudzba
Program
+Main()