Strukturalni paterni

1. Adapter pattern

Primjenu za adapter pattern bi mogli imati ako naiđemo na izazove prilikom integracije Google Maps API-ja u sistem zbog razlika u interfejsu ili formatu podataka API-ja. Taj adapter bi služio kao most između našeg sistema i Google Maps API-ja, rješavajući konverziju ili prilagodbu podataka i osiguravajući besprijekornu komunikaciju između njih.

Na primjer, ako Google Maps API očekuje geografske koordinate (latitude i longitude) za određivanje adrese lokacije, dok naš sistem koristi drugačiji format ili reprezentaciju adresa, možemo kreirati adapter koji će konvertirati adresni format koji se koristi u našem sistemu u traženi format za API.

2.Bridge pattern

Bridge obrazac bi u našem sistemu omogućio da odvojimo hijerarhiju klasa za načine plaćanja od hijerarhije klasa za narudžbe. To nam donosi nekoliko prednosti.

Prvo, omogućava nam fleksibilnost pri dodavanju novih načina plaćanja u sistem. Bez Bridge obrasca, bili bismo ograničeni načinima plaćanja koji su direktno povezani sa klasama narudžbi. Ali, koristeći Bridge obrazac, možemo lako dodavati nove klase načina plaćanja bez mijenjanja postojećeg koda narudžbi. Na taj način, možemo podržavati različite načine plaćanja kao što su "GotovinaPlaćanje", "KreditnaKarticaPlaćanje" ili "PayPalPlaćanje" bez značajnog utjecaja na druge dijelove sistema.

Drugo, Bridge obrazac nam omogućava da odvojimo logiku plaćanja od logike narudžbi. Svaki način plaćanja može imati svoju specifičnu logiku izvršavanja plaćanja, ali koristimo iste klase narudžbi. Ovo nam pruža modularnost i omogućava nam da lako mijenjamo ili proširujemo logiku plaćanja ili narudžbi ne utičući na ostale dijelove sistema.

Korištenje Bridge obrasca na ovaj način donosi nam veću fleksibilnost, održivost i proširivost u našem projektu. Možemo se prilagoditi promjenama zahtjeva i dodati nove funkcionalnosti bez velikih prepravki postojećeg koda.

3.Composite pattern

Composite obrazac mogao bi biti implementiran u klasi Osoblje kako bi se omogućilo organiziranje osoblja u hijerarhiju ili grupisanje. Svaki član osoblja (doktor, farmaceut, admin) može biti tretiran kao pojedinačna jedinica ili kao dio veće grupe. Na primjer, možemo imati grupu "Medicinski tim" koja se sastoji od nekoliko doktora i farmaceuta.

Ovaj obrazac omogućava jednostavno upravljanje osobljem na više nivoa, bez obzira na njihovu specifičnu ulogu. Na primjer, možemo izvršiti operacije nad cijelim Medicinskim timom, kao što je dodavanje novih članova ili prikaz plata svih uposlenih. Također omogućava izvršavanje operacija na pojedinačnim

članovima osoblja, kao što je promjena podataka o doktoru ili naručivanje nove zalihe lijekova od strane farmaceuta.

Korištenje Composite obrasca donosi fleksibilnost i olakšava dodavanje novih funkcionalnosti u budućnosti. Na primjer, ako se uvede nova uloga u osoblju, poput medicinske sestre, možemo je jednostavno dodati u postojeću hijerarhiju bez mijenjanja ostatka koda. Također omogućava da se operacije izvršavaju na svim članovima osoblja na transparentan način, bez obzira na njihovu konkretnu ulogu ili poziciju unutar hijerarhije.

4.Decorator pattern

Decorator pattern bi mogli iskoristiti da unaprijedimo funkcionalnost pretrage lijekova po kategorijama.

Kreiramo SortingDecorator koji dekorira osnovnu funkcionalnost pretrage lijekova. Ovaj dekorator omogućava korisnicima da dinamički dodaju mogućnost sortiranja lijekova po cijeni, popularnosti ili drugim kriterijima.

Kod SortingDecorator klase koordinira komunikaciju s osnovnom klasom pretrage lijekova i primjenjuje odgovarajući algoritam sortiranja na rezultate pretrage. Na taj način, korisnici mogu jednostavno odabrati željenu kategoriju sortiranja i dobiti pregled lijekova koji su razvrstani prema odabranom kriteriju.

Implementirajući Dekorator obrazac, omogućavamo fleksibilnost u našem sistemu. Možemo dodati nove kriterije sortiranja ili prilagoditi postojeće bez mijenjanja osnovne funkcionalnosti pretrage lijekova. Također, dekoratori se mogu kombinovati kako bismo pružili još složenije mogućnosti sortiranja, prilagođene potrebama naših korisnika.

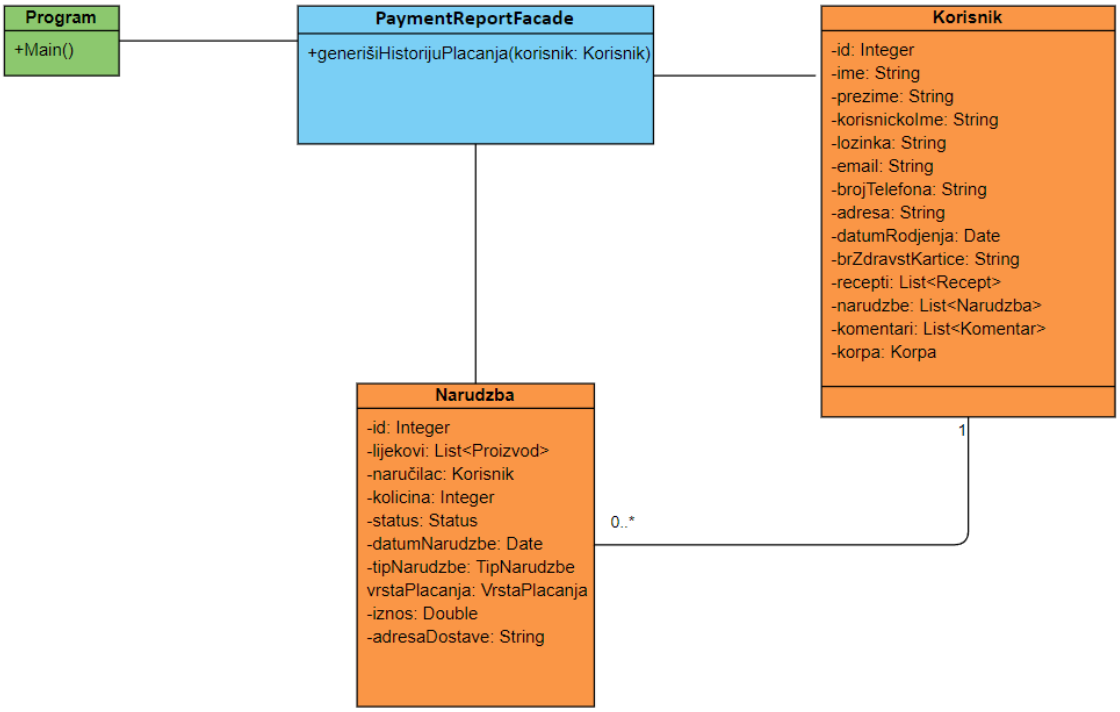
5.Facade pattern

Facade pattern bi mogli iskoristit kako bismo pojednostavili generisanje izvještaja o plaćanjima u našoj e-apoteci. Kreiramo PaymentReportFacade klasu koja pruža jednostavno sučelje za generisanje izvještaja o plaćanjima.

Interni kod PaymentReportFacade klase koordinira komunikaciju s relevantnim klasama kao što su Narudžba, Korpa i Korisnik kako bismo prikupili potrebne podatke o plaćanjima.

Kroz Fasada obrazac, sakrivamo složenosti generisanja izvještaja o plaćanjima od ostalih dijelova sistema. Time je omogućeno jednostavno pozvati generatePaymentReport(userId) koja će prikupiti potrebne podatke i generisati odgovarajući izvještaj. Također, unutar PaymentReportFacade klase, vršimo obradu podataka, formatiranje izlaza izvještaja i primjenu dodatnih transformacija ako je potrebno.

Implementirali bi na sljedeći način:



6.Flyweight pattern

Flyweight pattern je koristan kada imamo više objekata sličnih atributa i želimo optimizirati upotrebu memorije. U kontekstu klase Lijek, Flyweight obrazac se može primijeniti kako bi se smanjila potrošnja memorije za zajedničke atribute kao što su rokTrajanja i Tezina koji se dijele između različitih instanci lijekova.

Umjesto da svaki lijek ima vlastite instance atributa rokTrajanja i Tezina, Flyweight pristup podrazumijeva dijeljenje tih zajedničkih podataka između više instanci lijekova koje imaju iste vrijednosti za te atribute.

Ovim pristupom, umjesto stvaranja novih instanci atributa za svaki pojedinačni lijek, koristimo već postojeće objekte koji dijele iste vrijednosti za rokTrajanja i Tezina. To rezultira uštedom memorije jer se smanjuje broj objekata koji se stvaraju.

7.Proxy pattern

Proxy pattern je koristan u situacijama kada je potrebno kontrolirati pristup objektu i obaviti dodatne provjere ili funkcionalnosti. U kontekstu autorizacije komentara, Proxy obrazac se može primijeniti na klasu “Komentar” kako bi se osiguralo da samo registrovani i prijavljeni korisnici mogu dodavati komentare.

Prilikom klika na "Dodaj komentar", Proxy objekt će prvo provjeriti da li je korisnik prijavljen. Ako korisnik nije prijavljen, Proxy može odbiti dodavanje komentara ili preusmjeriti korisnika na stranicu za prijavu. U

slučaju kada je korisnik prijavljen, Proxy će proslijediti zahtjev stvarnom objektu Komentar da izvrši dodavanje komentara.

Korištenje Proxy obrasca na autorizaciji komentara pomaže u održavanju sigurnosti i kontrole pristupa, omogućava fleksibilnost u dodavanju novih provjera i funkcionalnosti te olakšava implementaciju i održavanje sustava za upravljanje komentarima.

Implementirali bi ga na sljedeći način: