Patterni ponašanja

Patterni ponašanja koje smo odlučile implementirati u našem projektu su:

Strategy pattern - design pattern koji se koristi kada želimo omogućiti promjenu ponašanja objekta tokom izvođenja programa. Ovaj pattern omogućuje definiranje različitih strategija (algoritama) koje objekt može koristiti, a odabir strategije se može mijenjati dinamički. U našem projektu, Strategy pattern bi mogao biti primijenjen kod plaćanja narudžbe. U različitim dijelovima našeg koda gdje se vrši kreiranje i manipulacija narudžbama, možemo koristiti Strategy pattern tako da omogućimo odabir različitih strategija plaćanja. Na primjer, prilikom kreiranja narudžbe, klijent može odabrati plaćanje putem PayPal-a, Credit Card-a ili gotovinom. Kreiranjem interfacea PaymentStrategy definiramo metode poput makePayment i refund za različite načine plaćanja. Na primjer, možemo imati implementacije PayPalStrategy, CreditCardStrategy, CashStrategy itd. Order klasa bi sadržavala metode poput setPaymentStrategy koja postavlja željenu strategiju plaćanja i makePayment koja će pozvati metodu makePayment na odabranoj strategiji. Implementacija Strategy patterna omogućava fleksibilnost u odabiru strategije plaćanja, omogućujući jednostavno dodavanje novih načina plaćanja u budućnosti. Također, odvaja logiku plaćanja od same klase Order, što olakšava održavanje i poboljšava čitljivost koda.

Observer pattern - često se koristi za implementaciju komunikacije između objekata u kojem jedan objekt obavještava druge objekte o promjeni svog stanja, što bi nama moglo biti korisno u sistemu, ali smo se ipak odlučile da ga ne implementiramo. U kontekstu našeg sistema, observer pattern bi se mogao primjeniti na više načina, te ćemo sada opisati neke od potencijalnih slučajeva kada bi nam koristio. Kada konobar obradi narudžbu, možemo implementirati observer pattern da obavijesti druge relevantne objekte o tome. Na primjer, ako bi imali objekt "Šank" koji će se registrovati kao observer i biti obaviješten svaki put kada konobar obradi narudžbu. Šank može zatim pokrenuti pripremu narudžbe. Ako vlasnik želi pratiti stanje kafića, možemo koristiti observer pattern kako bi obavijestili vlasnika o važnim događajima. Na primjer, ako imamo objekt "Vlasnik" koji će biti observer on će biti obaviješten kada se zalihe nekog važnog sastojka približe minimumu ili kada broj gostiju pređe određeni prag.

Mediator pattern – možemo koristiti za olakšavanje komunikacije između različitih objekata putem posrednika, umjesto da direktno komuniciraju između sebe. Primjer gdje bi mogli iskoristiti pattern je naveden u nastavku. Kada bi imali više konobara u kafiću, možemo koristiti mediator pattern kako bi omogućili komunikaciju između njih. Mediator objekt može poslužiti kao centralna tačka za razmjenu informacija između konobara. Na primjer, kada jedan konobar završi s obradom narudžbe, može poslati obavijest mediatoru, a zatim mediator može proslijediti tu informaciju drugim konobarima koji možda imaju veze s tom narudžbom (na primjer, ako je potrebna pomoć pri dostavi).

Iterator pattern - ovaj pattern se koristi za pristupanje elemenata kolekcije bez otkrivanja unutrašnje strukture te kolekcije. Primjer primjene patterna Iterator u našem projektu može biti prolaženje kroz narudžbe i prikazivanje njihovih detalja na korisničkom interfejsu. Umjesto direktnog pristupa elementima kolekcije naloga, koristili bismo iterator da bismo dobili svaku narudžbu jednu po jednu i prikazali njene detalje na interfejsu. Implementacija Iterator patterna pruža jednostavnost, fleksibilnost i intuitivan pristup elementima kolekcije. Također olakšava dodavanje novih metoda i operacija koje se mogu izvoditi na narudžbama u budućnosti, bez promjene interne strukture

kolekcije. U projektu mogao bi biti primijenjen na sljedeći način: Ako bismo klasu "OrderCollection" implementirali kao kolekciju naloga, unutar te klase bismo definirali i implementirali "Iterator" interfejs, koji bi sadržavao metode poput "getNext", "hasMore" i "reset" za pristup elementima kolekcije. Također, unutar klase "OrderCollection" implementirali bismo "OrderIterator" koji bi bio odgovoran za iteraciju kroz elemente kolekcije naloga. Klijenti mogu dobiti instancu objekta "OrderIterator" pozivanjem metode "getIterator" unutar klase "OrderCollection" i tako mogu koristiti iterator za petlju kroz narudžbe.