PATERNI PONAŠANJA

1) **Strategy patern**

Uloga **strategy paterna** jeste da izdvaja algoritam iz matične klase i uključuje ga u posebne klase. Pogodan je kada postoje različiti primjenjivi algoritmi (strategije) za neki problem.
Strategy patern omogućava klijentu izbor jednog od algoritma iz familije algoritama za korištenje. Algoritmi su neovisni od klijenata koji ih koriste.
Podržava *open-closed* princip.

U nasem slucaju, **strategy patern** bi se mogao iskoristiti prilikom odabira nacina sortiranja izbora uslužne jedinice (restorana,supermarketa ili fast-fooda),gdje bi algoritam djelovao na osnovu strategije, odnosno na osnovu izbora korisnika.

2) **State patern**

State Pattern je dinamička verzija Strategy paterna.
Objekat mijenja način ponašanja na osnovu trenutnog stanja.
Postiže se promjenom podklase unutar hijerarhije klasa.

Status dostupnosti restorana može biti modeliran pomoću State obrasca, s različitim stanjima kao što su "Otvoren", "Zatvoren" ili "Na pauzi". Ovo stanje može se dinamički mijenjati ovisno o radnom vremenu restorana ili drugim faktorima, a aplikacija može prilagoditi prikaz restorana ovisno o trenutnom stanju.

State patern možemo koristiti i u opisu statusa narudžbe korisniku.

3) **TemplateMethod patern**

Omogućava **izdvajanje određenih koraka algoritma u odvojene podklase**.
Struktura algoritma se ne mijenja - mali dijelovi operacija se izdvajaju i ti se dijelovi mogu implementirati različito.

U nasem slucaju, templateMethod patern bi se mogao iskoristiti prilikom kupovine , gdje u zavisnosti da li je korisnik prijavljen ili ne, prilikom kupovine narudžbe uračunavamo popust.
Tok narudžbe i druge mogućnosti će ostati jednake, jedina razlika će se ogledati u popustu, zavisno od podklase korisnika.

Osnovna klasa korisnika ima implementirane *default* verzije metoda, a *override* je za popust (koji u općem slučaju vraća 0) napravljen samo u klasi koja omogućava popust pri kupovini.

4) **Observer patern**

Uloga **Observer paterna** je da **uspostavi relaciju između objekata tako kada jedan objekat promijeni stanje drugi zainteresirani objekti se obavještavaju**.

Kada korisnik napravi narudžbu, sistem može koristiti Observer obrazac kako bi obavijestio korisnika o promjenama statusa njihove narudžbe. Na primjer, korisnik može biti obaviješten putem e-maila,

SMS-a ili notifikacije u aplikaciji kada se status narudžbe promijeni (npr. "Priprema", "Dostava", "Isporučeno").

5) **Iterator patern**

Iterator patern omogućava sekvencijalni pristup elementima kolekcije **bez poznavanja kako je kolekcija strukturirana**.

U nasem slucaju, **iterator patern** bi mogli iskoristiti prilikom korisnikove „posjete“ web aplikaciji. Korisnik bi mogao da bira nacin listanja artikala, bilo kroz *„shuffle“* mode (slucajni redoslijed artikala), *„cijena“* (ide kroz artikle prema nekom algoritmu koji uzima u obzir cijenu) itd.

6) **Mediator patern**

Mediator patern enkapsulira protokol za komunikaciju medju objektima dozvoljavajuci da objekti komuniciraju bez medjusobnog poznavanja interne strukture objekta.

U nasem slucaju, mediator patern bi mogli iskoristiti ukoliko bi upravljali narudžbama. Mediator može poslužiti kao centralizirani posrednik za upravljanje narudžbama. Kada korisnik izvrši narudžbu, različiti dijelovi sistema kao što su administrator, kuriri i sistem naplate mogu komunicirati putem mediatora radi obrade narudžbe.

7)

Memento patern omogucava da spasimo i vratimo prijašnje stanje objekta bez otkrivanja detalja njegove implementacije.

U našem slucaju, **memento patern** bi mogli iskoristiti prilikom kupovine karte, gdje bi korisnik bio u mogucnosti da se pritiskom na *back* vrati na prethodno odabranu količinu ili vrstu jela ili neke druge promjene u narudžbi.