# PATERNI PONAŠANJA

## *STRATEGY PATERN*

Strategy patern koristimo kada imamo više različitih algoritama koje možemo zamijeniti u toku izvršavanja programa. U našem sistemu, ovaj patern bi bio koristan prilikom sortiranja proizvoda na stranici za pretragu. Na primjer, korisnik može izabrati da sortira proizvode po cijeni, nazivu ili datumu dodavanja.
Za implementaciju ovog paterna, napravili bismo interfejs ISortiranjeProizvoda sa metodom sortiraj(List<Proizvod>). Zatim bismo implementirali različite klase kao što su SortirajPoCijeni, SortirajPoNazivu, SortirajPoDatumu, koje implementiraju ovu metodu. Na ovaj način omogućavamo lako proširenje i fleksibilnost kod promjene algoritma sortiranja bez izmjene osnovnog koda.

## *OBSERVER PATERN*

Observer patern omogućava da objekti budu obaviješteni o promjenama u drugim objektima. U našoj aplikaciji, ovaj patern bi se koristio za slanje notifikacija korisnicima – npr. kada proizvod koji su dodali na listu želja postane dostupan ili na popustu, ili kada bi korisnik kojeg prate objavio neki novi artikal.

Implementacija uključuje interfejs IPosmatrač sa metodom obavijesti(Poruka p), dok klasa Artikal može imati listu posmatrača koje obavještava kada se promijeni dostupnost ili cijena.

## *STATE PATERN*

State patern primjenjujemo kada objekat treba da mijenja ponašanje zavisno od svog stanja. U našem slučaju, korisnik može imati različite uloge – Korisnik, Kuriska služba, Administrator – i svaki od njih ima različita prava.
Napravili bismo apstraktnu klasu UlogaKorisnika sa metodom obradiNarudzbu(), a konkretne klase KorisnikUloga, KurirskaSluzbaUloga, AdminUloga implementiraju različita ponašanja.

## *ITERATOR PATERN*

Iterator patern omogućava pristup elementima kolekcije bez izlaganja unutrašnje strukture. U aplikaciji ePazar, možemo ga koristiti za prikaz proizvoda u kategorijama. Na primjer, klasa Kategorija može implementirati metodu dajIterator(), koja vraća objekat tipa ProizvodIterator, a koji omogućava sekvencijalni prolazak kroz proizvode.

## *TEMPLATE METHOD PATERN*

Template Method patern se koristi kada imamo algoritam koji sadrži fiksne korake, ali neki od njih treba da budu redefinisani u podklasama.
Ovaj patern bismo koristili kod obračuna narudžbe, gdje osnovni algoritam ostaje isti (npr. obračun artikala, dostave ili slično), ali se određeni dijelovi razlikuju zavisno od vrste korisnika ili tipa dostave.
Apstraktna klasa ObracunNarudzbe bi sadržavala metodu obradiNarudzbu() koja poziva korake obradiArtikal(), izracunajDostavu() itd., a podklase bi implementirale ove korake drugačije.

## *CHAIN OF RESPONSIBILITY PATERN*

Ovaj patern omogućava lančano obrađivanje zahtjeva – svaki objekat u lancu može obraditi zahtjev ili ga proslijediti dalje.
Primjena u ePazaru može biti preko validacije narudžbe. Tokom narudžbe, korisnik unosi adresu, podatke o plaćanju, dostavi itd. Svaka od ovih provjera može biti dio lanca odgovornosti.
Napravimo apstraktnu klasu ValidacijaNarudzbe sa metodom validiraj(Narudzba n), gdje svaka konkretna klasa (ValidacijaAdrese, ValidacijaPlacanja, ValidacijaDostave) vrši svoj dio provjere i, ako je sve u redu, prosljeđuje dalje.
Na ovaj način obezbjeđujemo modularnu i proširivu validaciju bez „if-else“ konsturkcije.

## *MEDIATOR PATERN*

Mediator patern koristi se za centralizaciju komunikacije između objekata. U aplikaciji ePazar, može se koristiti za upravljanje interakcijama između modula Korisnik, Narudžba i Pracenje.
U implementaciji bismo kreirali interfejs IMediator i klasu ProcesNarudzbeMediator koja koordinira interakcije – npr. nakon što korisnik potvrdi narudžbu, mediator obavještava

sistem za praćenje, pa modul za narudžbu.  Na ovaj način se smanjuje povezanost između klasa i olakšava održavanje sistema.

## *VISITOR PATERN*

Visitor patern omogućava definisanje novih operacija nad objektima bez mijenjanja njihovih klasa. Ovaj patern bismo mogli iskoristiti u situaciji kada želimo izvršiti različite statističke analize nad korisnicima i proizvodima – npr. broj narudžbi po korisniku, ukupan prihod po kategoriji proizvoda, prosječna cijena proizvoda itd.
Implementirali bismo interfejs IVisitor sa metodama posjeti(Korisnik k) i posjeti(Artikal a). Klase Korisnik i Artikal bi imale metodu prihvati(IVisitor v). Svaki konkretni visitor bi definisao svoju specifičnu obradu (npr. StatistikaNarudzbi, StatistikaPrihoda).
Na ovaj način možemo dodavati nove operacije bez mijenjanja osnovne strukture klasa.

## *INTERPRETER PATERN*

Interpreter patern se koristi kada imamo jezik za specifičnu domenu i želimo da analiziramo ili izvršavamo instrukcije u tom jeziku.
U našem slučaju, ovaj patern možemo koristiti ako omogućimo naprednu pretragu pomoću vlastitog mini-jezika. Na primjer, korisnik može unijeti upit poput:
cijena < 100 AND kategorija = "elektronika"
Ovaj string bi bio interpretiran u odgovarajuće filtere.
Potrebno je kreirati apstraktnu klasu Izraz sa metodom interpretiraj(Kontekst k), te konkretne klase kao što su IzrazCijenaManjeOd, IzrazKategorijaJednaka, IzrazI itd.
Ovaj pristup omogućava fleksibilnu i moćnu pretragu za napredne korisnike ili administraciju.

## *MEMENTO PATERN*

Memento patern omogućava spremanje i kasniji povrat prethodnog stanja objekta bez narušavanja enkapsulacije.
Ovaj patern možemo koristiti za funkcionalnost „poništi promjenu“ (undo) kod uređivanja korisničkog profila ili narudžbe. Na primjer, korisnik može urediti adresu isporuke, a sistem sačuva prethodnu verziju u memento objektu.
Imamo klasu Profil, koja može stvoriti objekt tipa ProfilMemento, a klasa Caretaker čuva sve verzije. Kasnije, korisnik može odlučiti da vrati staru adresu pozivom vratiStanje().

## *COMMAND PATERN*

Command patern omogućava da svaki zahtjev tretiramo kao poseban objekat. Na taj način, zahtjeve možemo lako proslijediti, sačuvati, staviti u red čekanja ili omogućiti funkcionalnosti poput undo/redo.
U našoj aplikaciji, ovaj patern možemo koristiti za obradu korisničkih akcija u virtuelnoj korpi. Na primjer, korisnik dodaje ili uklanja proizvod, a svaka ta akcija se može predstaviti kao objekat komande (DodajProizvodCommand, UkloniProizvodCommand).
Kreirali bismo interfejs ICommand sa metodama izvrši() i poništi(). Svaka konkretna komanda implementira ove metode. Time bismo omogućili da korisnik može opozvati prethodnu akciju (undo), što dodatno poboljšava korisničko iskustvo.