Univerzitet u Sarajevu
Elektrotehnički fakultet u Sarajevu
Odsjek za računarstvo i informatiku

# Paterni ponašanja
# eDostava

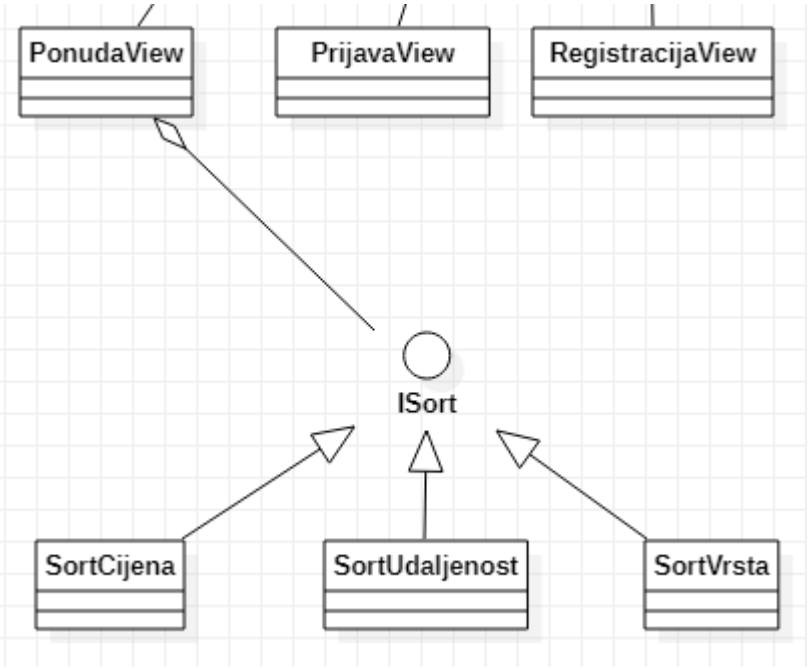
Objektno orijentisana analiza i dizajn

**Naziv grupe**: eDostava
**Članovi:** Ina-Bedra Mujaković
Ema Kalmar
Rijad Zejnilović

# 1. Strategy patern

Uloga Strategy patern-a je da izdvaja algoritam iz matične klase i uključuje ga u posebne klase. Koristi se kada postoje različite primjenljivne strategije za neki problem. On omogućava korisniku izbor jednog algoritma iz familije algoritama za korištenje.

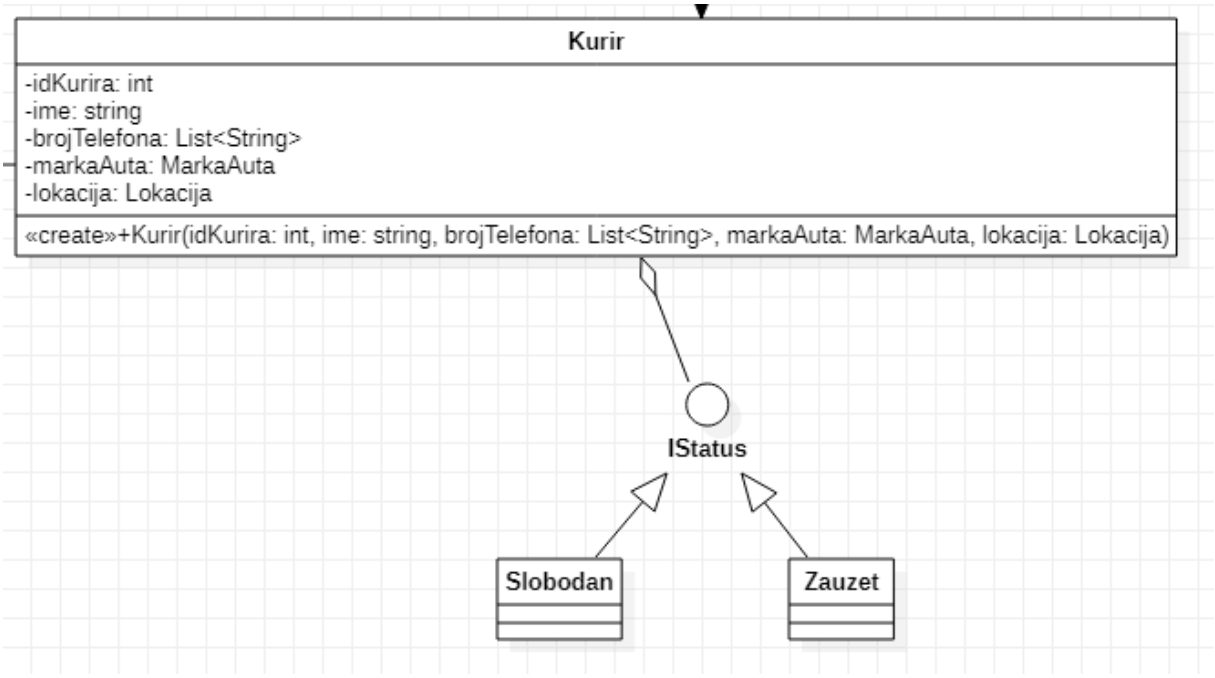
Ovaj patern u našem sistemu možemo iskoristiti kod filtriranja proizvoda. Naime, korisnik kada otvori formu za pretragu dostupnih proizvoda u prodavnici, može vršiti filtriranje proizvoda po vrsti, cijeni i udaljenosti, gdje po vrsti filtiramo proizvode na namirnice, kućanske potrepštine, hemiju i ostalo. U svrhu omogućavanja ova dva tipa filtriranja, iskorišten je upravo ovaj patern.



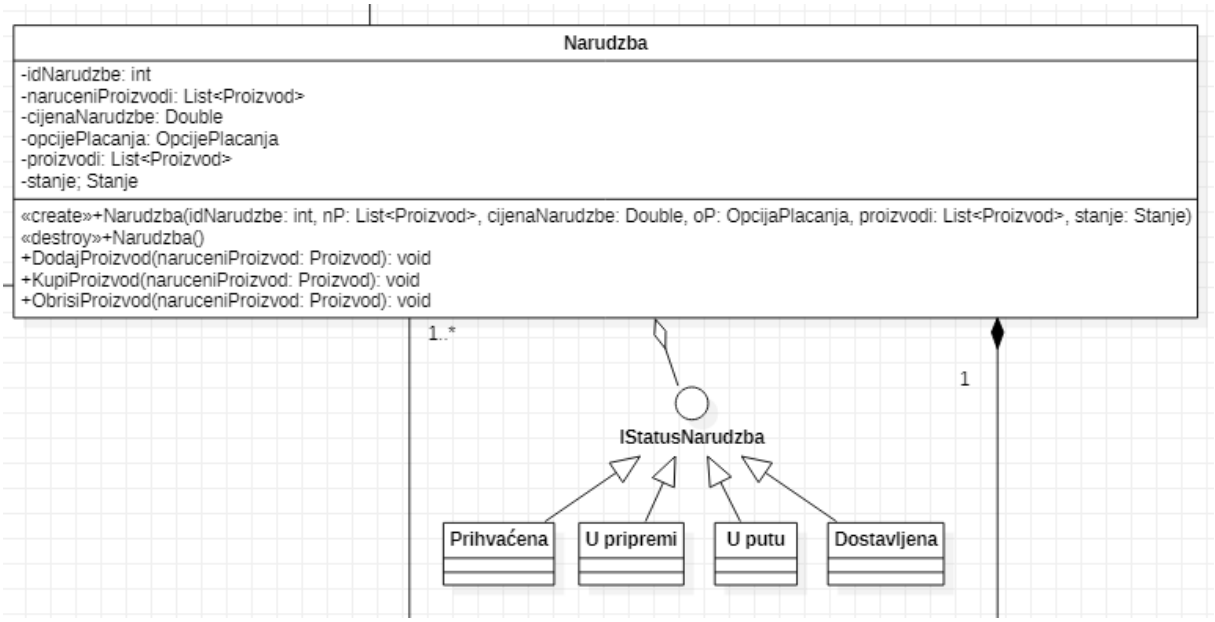
# 2. State patern

State patern omogućava objektu da promijeni svoje ponašanje, od kojih zavisi njegovo ponašanje. Nakon promjene stanja, objekat se počinje ponašati kao da je promijenio klasu. Objekat mijenja način ponašanja na osnovu trenutnog stanja. State patern predstavlja dinamičku verziju Strategy patern-a. Postiže se promjenom podklase unutar hijerarhije klasa. Podržava open-closed princip.

State patern bismo mogli iskoristiti u našem sistemu ako uvedemo da je kurir u stanju “slobodan”, a kad vrši dostavu “zauzet”. Definišemo klasu Dostava koja održava instancu stanja, koja definira u kom se stanju nalazi kurir i interfejs koji je od interesa za klijenta. Odnosno, svaki kurir će imati svoju lokaciju i informaciju na osnovu koje će biti definisano njegovo trenutno stanje(slobodan ili zauzet). Klijent klasa komunicira sa ovom klasom.



Sličan primjer imamo u klasi Narudzba, gdje možemo imati odgovarajući interfejs koji nam daje informaciju o tome u kakvom se trenutno stanju nalazi narudžba.




# 3. Template method patern

Svrha Template method patern-a je da omogući izdvajanje odredenih koraka algoritma u odvojene podklase. Samim tim, struktura algoritma se ne mijenja, mali dijelovi operacija se izdvajaju i ti se dijelovi mogu implementirati različito. Ovaj patern se sastoji od klase Algorithm koja uključuje metodu koja izdvaja dijelove svojih operacija u druge klase. Tu se nalazi interfejs IPrimitive koji definira operacije koja pomenuta metoda izdvaja u druge klase te AnyClass koja implementira interfejs IPrimitive.

Template Method patern bismo mogli iskoristiti u nasem sistemu ako promijenimo implementaciju razlicitog placanja obicnog i vip korisnika. Definisemo klasu Placanje u kojoj se nalazi metoda TemplateMethod(). Ovu klasu povezujemo je sa interfejsom IPlacanje koji ima metode platiOsnovicu() i platiSPopustom(). Ovaj interfejs implementuju klase ObicniKorisnik i VipKorisnik. TemplateMethod() u svom dijelu poziva ove funkcije i u zavisnosti od tipa korisnika platice se drugačija cijena. TemplateMethod() bi se pozivala u klasi Korisnik u kojoj bismo dodali metodu Placanje(). Dakle, razlika između ova dva korisnika je što bi VIP korisnik skupljao određene poene i ostvarivao određene pogodnosti i popuste.

# 4. Observer patern

Uloga Observer patern-a je da uspostavi relaciju izmedu objekata tako kada jedan objekat promijeni stanje drugi zainteresirani objekti se obavještavaju.

U nasem sistemu observer patern je iskoristen u slucaju kada imamo situaciju da je vise korisnika zainteresovano za isti proizvod, te korisnici koji nisu u mogucnosti rezervisati i kupiti taj proizvod trenutno mogu se prijaviti da dobiju obavjestenje kada on ponovo postane dostupan za kupovinu ili rezervaciju.

# 5. Iterator patern

Iterator patern omogućava sekvencijalni pristup elementima kolekcije bez poznavanja kako je ta kolekcija struktuirana.

Ovaj patern bismo mogli iskoristiti ako umjesto listi kao atribut imamo mapu korisnika i njihovih kupljenih proizvoda, te ako bismo na primjer željeli naći korisnike koji nemaju aktivnih narudžbi u tom trenutku. Definisemo interfejs IEnumerable sa metodom getEnumerator() kojeg implementira klasa Kolekcija koja pored metode getEnumerator() moze imati i druge metode koje pružaju vrijednosti kolekcije u nekom drugom redoslijedu ili obliku. U klasu Korisnik dodajemo atribut tipa Map i povezujemo klasu sa interfejsom IEnumerable.

# 6. Chain of responsibility patern

Chain of responsibility patern predstavlja listu objekata, ukoliko objekat ne može da odgovori prosljeduje zahtjev narednom u nizu.

U našem sistemu, ovaj patern bi mogli iskoristiti u situaciji kada korisnik želi izvršiti narudžbu, odnosno podnijeti zahtjev za izvršavanjem. Napravili bi jednu klasu KreiranjeNarudžbe sa metodom kreiraj() koja bi prvo pozvala klasu ZahjtevZaNarudžbu koja bi kreirala jedan zahtjev, a onda bi se nakon odobravanja narudžbe, napravila i instanca klase Narudžba.

# 7. Medijator patern

Medijator patern omogućava reduciranje ovisnosti izmedu objekata. Ovaj patern ograničava direktnu komunikaciju izmedu objekata i forsira ih da kolaboriraju samo preko medijator objekta. Umjesto za direktno povezujemo veliki broj objekata, koristimo medijator, koji je zadužen za njihovu komunikaciju. Kada neki objekat želi poslati poruku drugom objektu, on šalje poruku medijatoru, a medijator zatim prosljeduje tu poruku drugom objektu.

Ovaj patern možemo iskoristiti kod feedbacka, odnosno utisaka o nekim restoranima ili prodavnicama. Aplikacija bi zahtijevala da prethodno imate obavljenu transakciju u restoranu ili prodavnici na koju želite ostaviti feedback. Klasa Korisnik bi imala atribut IfeedbackMedijator. Također, trebamo i novi interfejs IfeedbackMedijator koji bi imao metode sa funkcionalnostima provjere koji korisnik piše feedback i njegov sadržaj.