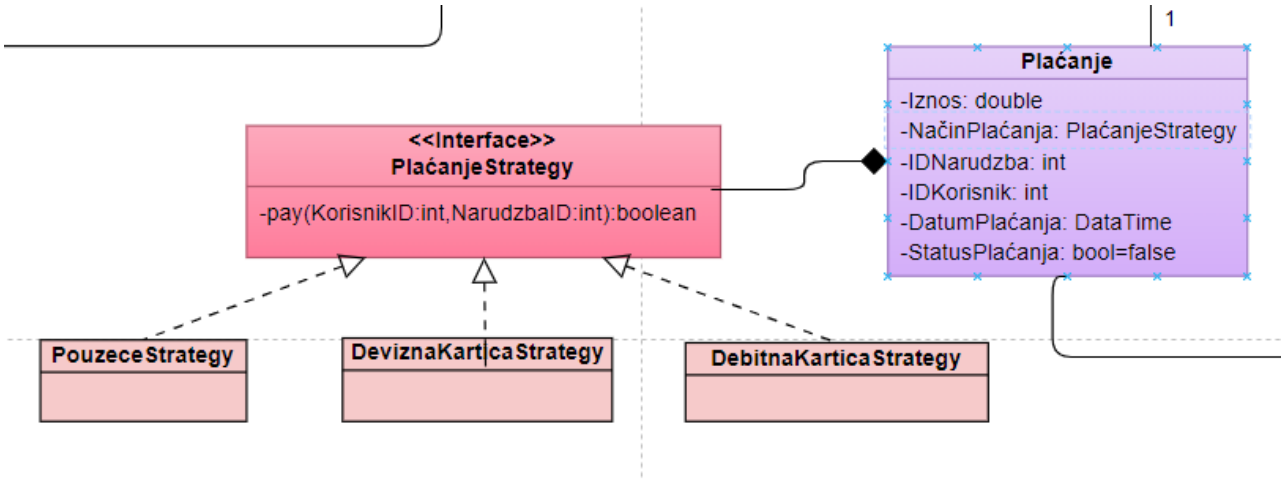
PATERNI PONAŠANJA

-AMA COSMETICS-

1. Strategy patern

Strategy patern je patern ponašanja koji omogućava definisanje različitih strategija ili algoritama za rešavanje određenog problema, pri čemu se strategija može dinamički izabrati i primeniti tokom izvršavanja programa. Ovaj patern omogućava izdvajanje specifičnih algoritama iz glavne logike programa, čime se postiže fleksibilnost i održivost koda.

U našoj aplikaciji se može primeniti za različite strategije načina plaćanja. Jedna strategija može biti plaćanje putem kreditne kartice a druga strategija može biti plaćanje pouzećem. Svaka strategija će biti implementirana kao posebna klasa koja ima metode za inicijalizaciju plaćanja, proveru ispravnosti podataka i izvršavanje plaćanja.

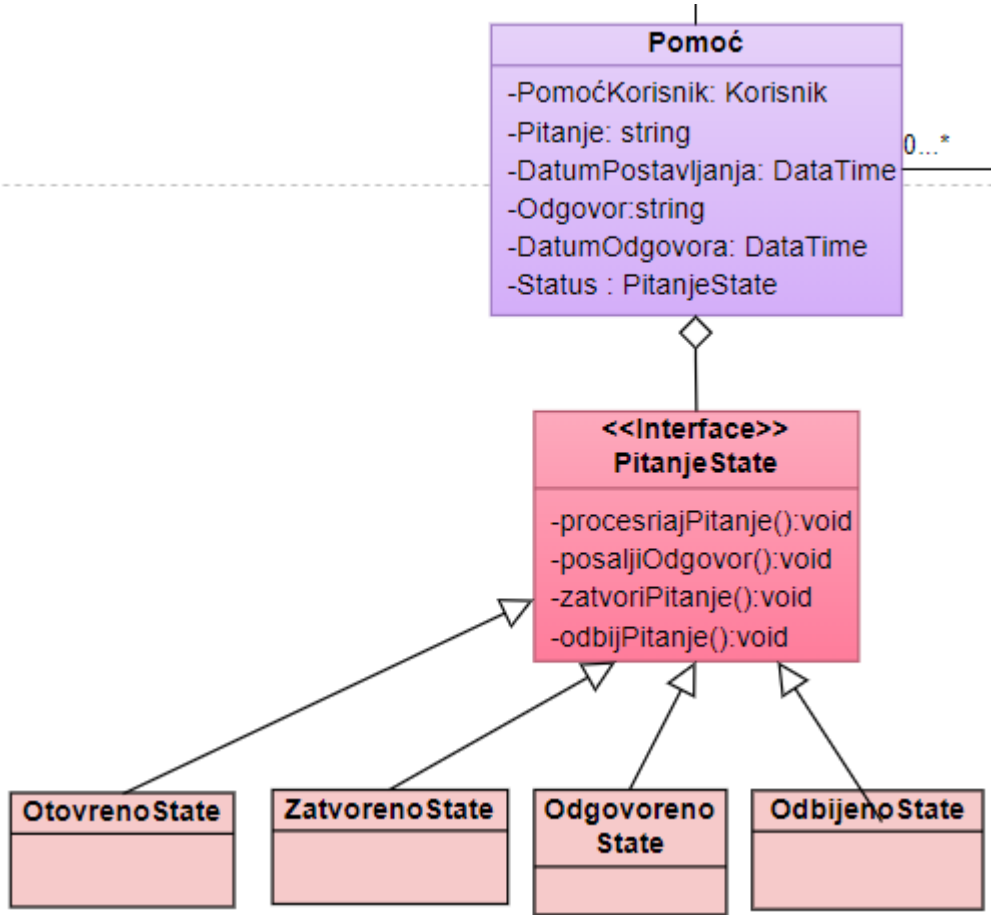


2. State patern

Objekat menja način ponašanja na osnovu trenutnog stanja. Postiže se promenom podklase unutar hijerarhije klasa. State patern se može primeniti na klasu Pomoć. Pomoću ovog paterna možemo implementirati stanja pomoći: Otvoreno, Odgovoreno, Zatvoreno i Odbijeno.

Najpre definišemo interfejs koji predstavlja stanje odgovora, koji može sadržati metode poput "procesirajPitanje", "pošaljiOdgovor", "zatvoriPitanje","odbijPitanje" itd. Ovaj interfejs će biti implementiran od strane svakog konkretnog stanja.

Stanja su izražena putem klasa “OtvorenoStanje”, “ZatvorenoStanje”, “OgovorenoStanje” i “OdbijenoStanje” koje smo implelentirali u našem sistemu.



3. Template method patern

Template Method patern je dizajn patern koji definiše skeletnu strukturu algoritma u osnovnoj apstraktnoj klasi, prepuštajući konkretnim podklasama da implementiraju pojedinačne korake algoritma. Ovaj patern omogućava fleksibilnost koda, jer omogućava promenu ili proširenje pojedinih koraka algoritma, a istovremeno održava zajedničku strukturu.
U našem slučaju on je već implementiran kod apstraktne klase Korisnik koja definiše zajedničke metode za nasleđene klase NeregistrovaniKorisnik, RegistrovaniKorisnik i PremiumKorisnik koje se potom u njima definišu.
Jedan od primera bi mogla biti metoda "KreirajNarudžbinu" koja će u slučaju klase NeregistrovaniKorisnik ponuditi registraciju i obavestiti korisnika da naručivanje nije moguće ukoliko ne poseduje nalog. Za slučaj nasleđene klase RegistrovaniKorisnik će omogućiti kreiranje narudžbe nakon provere dostupnosti proizvoda, dok će PremiumKorisniku kao poklon dodati jedan proizvod sa lise želja.

4. Observer patern

Observer pattern je softverski obrazac dizajna koji se koristi za implementaciju komunikacije između objekata u programu. Ovaj obrazac omogućava objektu, poznatom kao subjekt, da obavesti i automatski ažurira sve svoje zainteresovane objekte, poznate kao posmatrači ili observeri, kada dođe do promene u njegovom stanju.
Observer pattern se često koristi u situacijama kada je potrebno održavati konzistentnost između objekata koji zavise jedni od drugih, ali bez direktnog međusobnog povezivanja. patern se često koristi u online shopovima kako bi omogućio praćenje promena u stanju ili podacima i obaveštavanje relevantnih delova sistema o tim promenama.

U našem primeru se može primeniti na sledeća dva slučaja.
Kada se stanje narudžbine promeni (na primer, kada je narudžbina potvrđena, poslata ili isporučena), subjekt može obavestiti relevantne observere kao što su naručilac i kurirska služba o toj promeni, tako da mogu preduzeti odgovarajuće akcije.
Kada se stanje proizvoda promeni (na primer, ukoliko je proizvod bio rasprodat) može se poslati obaveštenje svim zaiteresovanim kupcima da je proizvod ponovo na stanju.

5. Itereator patern

Iterator patern je softverski patern koji se koristi za pristupanje i iteriranje kroz elemente kolekcije, bez otkrivanja detalja implementacije kolekcije. Ovaj patern omogućava efikasno i jednostavno iteriranje kroz elemente kolekcije bez potrebe za pristupanjem podacima direktno.

Iterator patern se može koristi za pregled proizvoda. Kada korisnik pregleda proizvode u određenoj kategoriji, Iterator patern se može koristiti za iteriranje kroz kolekciju proizvoda te kategorije i prikazivanje jednog po jednog proizvoda korisniku. Korisnik može koristiti funkcionalnosti poput hasNext() da proveri da li ima još proizvoda za prikazivanje, i next() da dohvati sledeći proizvod.
Takođe bi se mogao implementirati na primeru prikaza prethodnih narudžbina. Kada korisnik želi pregledati svoje prethodne narudžbine, Iterator patern se može koristiti za iteriranje kroz kolekciju prethodnih narudžbina i prikazivanje jedne po jedne narudžbine korisniku. Korisnik može koristiti funkcionalnosti poput hasNext() i next() da pregleda prethodne narudžbine.

6. Comand patern

Command patern je softverski patern dizajna koji se koristi za inkapsulaciju zahteva kao objekata, omogućavajući njihovu parametrizaciju, redosled izvršavanja, čuvanje i poništavanje. Ovaj patern omogućava fleksibilno rukovanje zahtevima, izvršavanje operacija i upravljanje undo/redo funkcionalnošću.
Glavna ideja iza Command paterna je da se zahtevi pretvaraju u objekte koji se sastoje od komande (metoda za izvršavanje zahteva) i mogućih dodatnih parametara. Ovi objekti se mogu lako manipulisati, čuvati, preneti ili dodavati u red za izvršavanje.
U našem slučaju Command bi se mogao upotrebiti za poništavanje ili otkazivanje narudžbine. Command patern se može koristiti za implementaciju funkcionalnosti poništavanja ili otkazivanja narudžbine. Komanda će predstavljati zahtev za poništavanje narudžbine, a Receiver će biti odgovoran za izvršavanje operacije poništavanja.