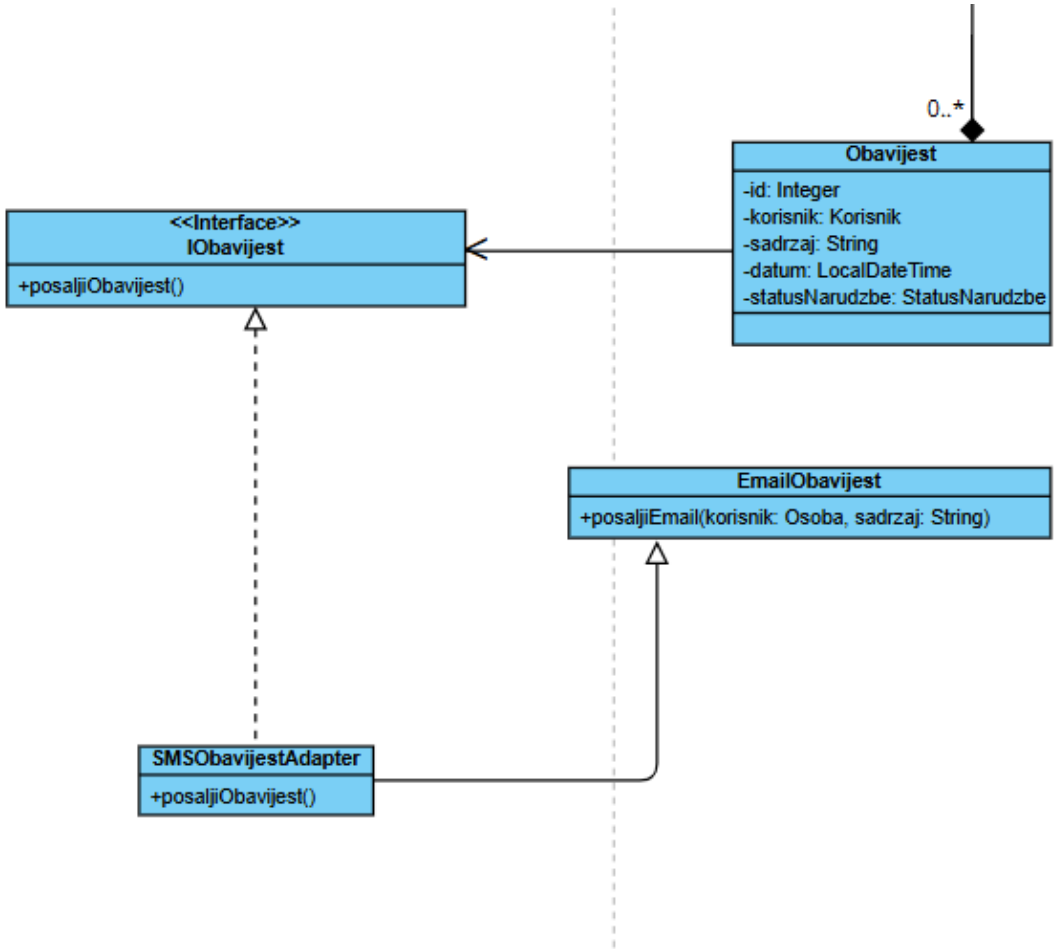
Strukturalni dizajn paterni

Adapter patern

Adapter patern nam omogućava širu upotrebu već postojećih klasa, tj u situacijama kada ne želimo mijenjati postojeću klasu a potreban nam je drugačiji interfejs već postojeće klase. On kreira novu adapter klasu koja služi kao posrednik između originalne klase i željenog interfejsa. Na taj način dobivamo željenu funkcionalnost bez izmjena na originalnoj klasi i bez ugrožavanja integriteta cijele aplikacije.

Adapter patern u našem sistemu možemo iskoristiti za slanje obavijesti koristeći različite sisteme za slanje obavijesti (email, SMS..). Dodati ćemo interfejs IObavijest koji definiše zajednički način slanja obavijesti, a zatim se implementiraju adapter klase poput EmailObavijest i SMSObavijestAdapter koje prilagođavaju konkretne sisteme za slanje poruka ovom interfejsu. Na ovaj način, sistem postaje fleksibilan, lako ga je proširiti novim načinima slanja obavijesti bez izmjene postojećeg koda.



Facade patern

Osnovna namjena Facade paterna je da osigura više pogleda (interfejsa) visokog nivoa na podsisteme čija implementacija je skrivena od korisnika. Facade patern se koristi kada sistem ima više podsistema pri čemu su apstrakcije i implementacije podsistema usko povezane.

Analizom našeg sistema, možemo uočiti složenost u procesu upravljanja narudžbama. Klasa Narudžba direktno komunicira s više klasa, uključujući Osoba, StavkaNarudžbe, Proizvod, te sistemima za obradu plaćanja poput KarticaPlaćanja, kao i s enum klasama koje definišu vrste plaćanja i preuzimanja.Ovakva implementacija čini kod zavisnim od implementacije klasa u pozadini sistema za narudžbu, čineći sistem težim za razumijevanje, korištenje i održavanje. Svaka promjena unutar logike naručivanja ili plaćanja potencijalno zahtijeva izmjene na više mjesta u kodu. Uvođenje Facade paterna na ovom mjestu, kroz kreiranje jedinstvenog interfejsa (npr. NarudzbaFacade), bi značajno pojednostavilo korištenje ovog podsistema. Facade bi sakrio internu kompleksnost, pružajući klijentima jednostavan i jasan interfejs za uobičajene operacije poput kreiranja narudžbe, obrade plaćanja ili provjere statusa, čime bi se smanjila povezanost, poboljšala čitljivost koda i olakšale buduće modifikacije sistema.

NarudzbaFacade predstavlja pojednostavljen interfejs koji omogućava korisnicima da kreiraju narudžbu bez potrebe da direktno upravljaju kompleksnim objektima kao što su Narudzba, StavkaNarudzbe i KarticaPlacanja. Umjesto da korisnik ručno kreira narudžbu, dodaje stavke, postavlja način plaćanja i računa ukupan iznos, NarudzbaFacade sve te korake obavlja interno kroz jednu metodu kreirajNarudzbu. Ova metoda prima podatke o korisniku, proizvodima i načinu plaćanja, automatski kreira narudžbu, dodaje sve stavke, validira karticu ako je potrebno i računa ukupan iznos narudžbe. Na taj način se složena logika pozadinskog sistema sakriva iza jednostavnog i intuitivnog interfejsa, što olakšava korištenje i održavanje koda.

Decorator patern

Osnovna namjena Decorator paterna je da omogući dinamičko dodavanje novih elemenata i ponašanja (funkcionalnosti) postojećim objektima. Objekat pri tome ne zna da je urađena dekoracija što je veoma korisno za iskoristljivost i ponovnu upotrebu komponenti softverskog sistema.

U sistemu za kafić, Decorator pattern se koristi za dinamičko dodavanje dodataka proizvodima bez potrebe da se kreira nova podklasa za svaku kombinaciju. Pošto je Proizvod apstraktna klasa iz koje nasljeđuju Hrana i Pice, možemo napraviti apstraktnu dekorator klasu ProizvodDecorator koja nasljeđuje Proizvod i sadrži referencu na drugi objekat tipa Proizvod. Na taj način, svaki konkretni dekorator – poput SlagDecorator ili

CokoladniPreljevDecorator – dodaje novu funkcionalnost (npr. povećanje cijene i promjenu opisa) bez izmjene osnovnog proizvoda. Tako korisnik može naručiti osnovnu kafu i fleksibilno dodavati dodatke kao što su šlag, preljev pri čemu se dekoratori jednostavno slažu jedan na drugi. Ovo omogućava veću fleksibilnost i održivost sistema, jer ne moramo unaprijed definisati sve moguće kombinacije proizvoda i dodataka.

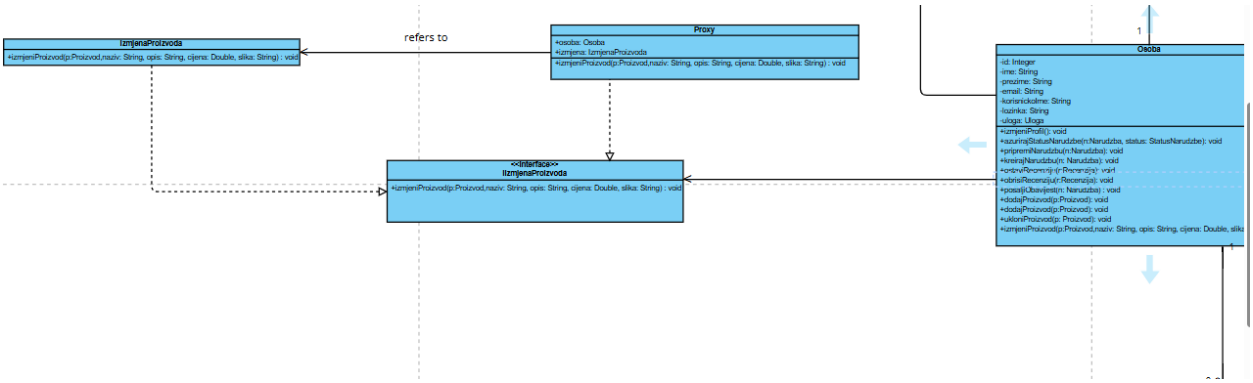
Bridge patern

Bridge pattern omogućava da se iste operacije primjenjuju nad različitim podklasama. Također se koristi kod izbjegavanja kreiranja novih metoda za postojeće funkcionalnosti. Ono što je karakteristično za ovaj pattern je to da te operacije imaju jedan zajednički korak, nakon čega su različite.

U sistemu za K&K kafić, Bridge pattern se koristi za **odvajanje apstrakcije plaćanja od konkretne implementacije načina plaćanja**, čime se omogućava njihovo nezavisno proširivanje. Umjesto da klasa Narudžba direktno koristi konkretne vrste plaćanja kao što su gotovina ili kartica, uvodi se apstraktna klasa Plaćanje, koja sadrži referencu na interfejs PlacanjeImplementacija. Ovaj interfejs definiše operaciju plati(iznos), a konkretne klase poput GotovinaPlacanje i već postojeće KarticnoPlacanje implementiraju to ponašanje. Klasa KarticnoPlacanje, koja već postoji u modelu i sadrži sve podatke i metode za validaciju kartice, sada se koristi kao konkretna implementacija interfejsa PlacanjeImplementacija. Apstraktna klasa Plaćanje može se slobodno proširivati bez potrebe da se mijenjaju konkretni načini plaćanja, i obrnuto. Time se izbjegava stvaranje velikog broja nasljednih klasa. Ova primjena Bridge patterna omogućava veću fleksibilnost, bolju organizaciju koda i olakšava održavanje sistema u skladu s principima objektno-orijentisanog dizajna, poput **Open/Closed principa** i **Single Responsibility principa**.

Proxy patern

U aplikaciji za kafić, Proxy pattern se koristi za kontrolu pristupa osjetljivim administratorskim funkcijama, poput dodavanja, uređivanja i brisanja proizvoda. Umjesto da korisnici direktno pristupaju implementaciji tih funkcionalnosti, koristi se proxy objekat koji stoji između korisnika i stvarne logike. Proxy provjerava ulogu i ovlasti korisnika prije nego što dopusti izvršavanje operacije. Na primjer, samo korisnici s ulogom Admin mogu pozvati metode za izmjenu proizvoda, dok će pokušaji drugih korisnika biti odbijeni. Na ovaj način, aplikacija dobija sigurnosni sloj koji štiti od neovlaštenog pristupa.



Compostite patern

U sistemu kafića, korisnicima je omogućen pregled različitih proizvoda poput hrane i pića, a radi poboljšanja korisničkog iskustva javlja se potreba za naprednim filtriranjem proizvoda prema više kriterija istovremeno, kao što su vrsta proizvoda, cijena ili veličina. Da bismo omogućili fleksibilno i proširivo filtriranje, koristi se Composite patern koji omogućava da se pojedinačni i složeni filteri tretiraju na isti način. Svaki filter implementira zajednički interfejs, a složeni filter sadrži više podfiltera i primjenjuje ih nad listom proizvoda kako bi prikazao samo one koji zadovoljavaju sve kriterije. Na ovaj način omogućava se jednostavna kombinacija filtera, lakša nadogradnja sistema i održivost koda bez potrebe za pisanjem posebne logike za svaku kombinaciju uslova filtriranja.