PATERNI PONAŠANJA

1) Strategy pattern

Strategy patern služi kako bi se različite implementacije istog algoritma izdvojile u različite klase, te kako bi se omogućila brza i jednostavna promjena implementacije koja se želi koristiti u bilo kojem trenutku. Na ovaj način omogućava se i jednostavno brisanje ili dodavanje novog algoritma koji se može koristiti po želji. Ovaj pattern nismo implementirali. Međutim kada bi željeli implementirati ovaj pattern to bi bilo kod plaćanja karte, tačnije kada bi omogućili drugačije načine plaćanja rezervisanja karte. Dakle dodali bi interfejs **iStrategija**, koji bi imao jednu metodu **placanjeUzivo** , i napravili bi nove klase koje implementiraju ovaj interfejs, npr. klasa za plaćanje karticom, te klasa za plaćanje uživo. Napravili bi posebnu klasu koja služi samo za plaćanje karte, u njoj bi imali atribut **iStrategija**, te metodu za promjenu načina plaćanja.

2) State pattern

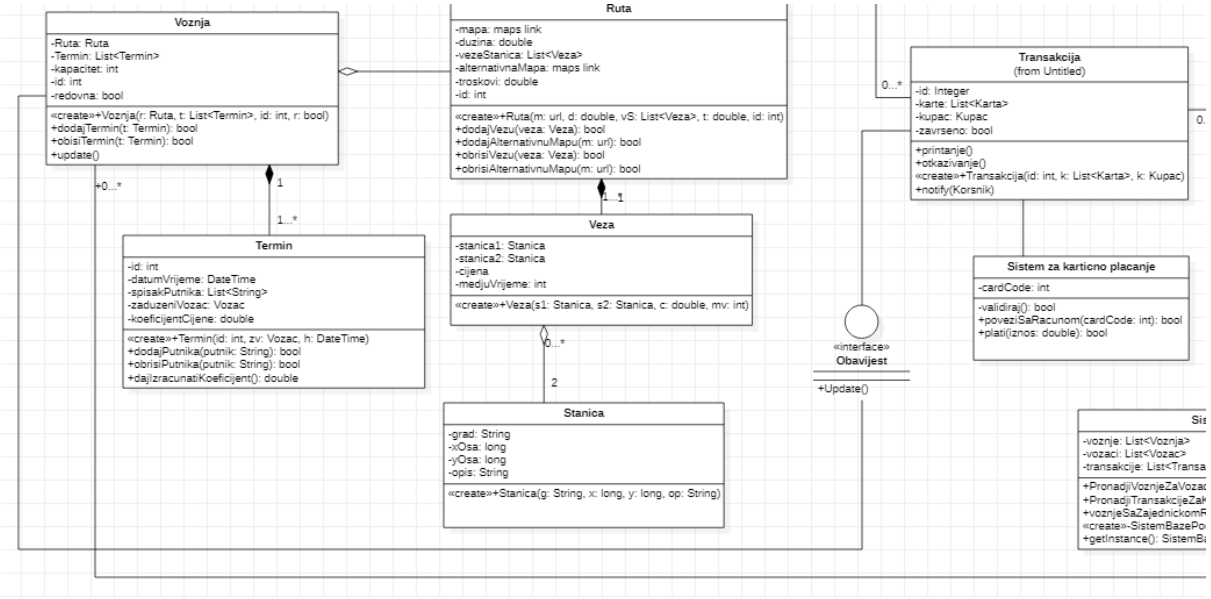
State patern omogućava objektu da mijenja svoja stanja, od kojih zavisi njegovo ponašanje. Sa promjenom stanja objekt se počinje ponašati kao da je promijenio klasu. Stanja se ne mijenjaju po želji klijenta, već automatski, kada se za to steknu uslovi. State pattern nije implementiran, ali bi se mogao implementirati za klasu **Transakcija.** Atribut tipa bool ,Završeno, bi mijenjao svoje stanje u zavisnosti od Sistema za kartično plačanje. Ukoliko bi Završeno bilo tipa true, omogućilo bi se printanje karte kao i zabrana brisanja te instance transakcije. Napravio bi se interfejs iStanje kojeg bi implementirale dvije klase za promjenu stanja sistema.

3) Template Method pattern

Template method patern služi za omogućavanje izmjene ponašanja u jednom ili više dijelova. Najčešće se primjenjuje kada se za neki kompleksni algoritam uvijek trebaju izvršiti isti koraci, no pojedinačne korake moguće je izvršiti na različite načine.

4) Observer pattern

Observer patern služi kako bi se na jednostavan način kreirao mehanizam pretplaćivanja. Pretplatnici dobivaju obavještenja o sadržajima na koje su pretplaćeni, a za slanje obavještenja zadužena je nadležna klasa. Na ovaj način uspostavlja se relacija između klasa kako bi se mogle prilagoditi međusobnim promjenama. BamBus sistem implementira Observer pattern na način da šalje obavijest putem email-a korsniku o uspješno izvršenoj transakciji. Unutar email-a korisnik dobija dokument koji predstavlja njegovu kartu koju može printati. Ovaj pattern je prikazan na sljedećoj slici.



5) Iterator pattern

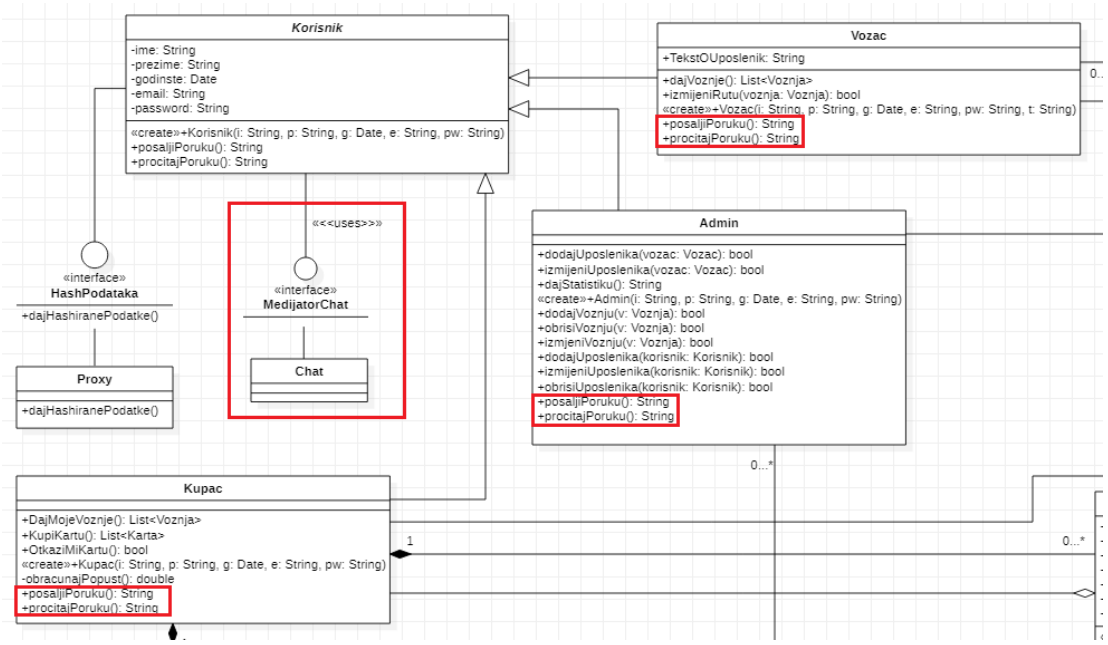
Iterator patern namijenjen je kako bi se omogućio prolazak kroz listu elemenata bez da je neophodno poznavati implementacijske detalje strukture u kojoj se čuvaju elementi liste. Izvedba liste može biti u obliku stabla, jednostruke liste, niza i sl., no klijentu se omogućava da na jednostavan način dolazi do željenih elemenata. Osim toga, ovaj patern preporučljivo je iskoristiti kada se za iteriranje koristi kompleksna logika koja ovisi o više kriterija. Ovaj patern nije implementiran. Iterator pattern bi se mogao implementirati na klasi vožnja. Napravio bi se intefejs iVoznja koji bi regulisao iteraciju za svaku instancu vožnje.

6) Chain of rensponsibility

Dati pattern je moguće realizirati ukoliko pravimo posebnu klasu za popust koja bi trebala više drugih faktora za realizaciju.Posmatrajmo situaciju ukoliko imamo pomoćnu klasu koja je zadužena da klasi popust javi da je kupac napravio recimo desetu vožnju čime mu se omogućava popust na neko sljedeće putovanje.Tada još ako je i student i taj faktor bi se uzeo obzir na sljedeću cijenu zajedno sa varijantom popusta nakon 10-te vožnje.Dakle različite vrste popusta i statusa kupca sastavljaju ustvari novu kartu koja važi određeni period.

7)Medijator pattern

Patern koji je primjenjen na komunikaciji između Vozača i Admina(kojeg možemo posmatrati kao kordinatora) i Kupca putem svojstvenog chata za potrebe obavljanja njihovog dijela posla. Kreiran je interface MediatorChat kao i konkretna klasa Chat.


Korisnik
-ime: String
-prezime: String
-godinste: Date
-email: String
-password: String
«create»+Korisnik(i: String, p: String, g: Date, e: String, pw: String)
+posaljiPoruku(): String
+procitajPoruku(): String
Vozac
+TekstOUposlenik: String
+dajVoznje(): List<Voznja>
+izmijeniRutu(voznja: Voznja): bool
«create»+Vozac(i: String, p: String, g: Date, e: String, pw: String, t: String)
+posaljiPoruku(): String
+procitajPoruku(): String
«<<uses>>»
«interface»
HashPodataka
+dajHashiranePodatke()
«interface»
MedijatorChat
Chat
Admin
+dodajUposlenika(vozac: Vozac): bool
+izmijeniUposlenika(vozac: Vozac): bool
+dajStatistiku(): String
«create»+Admin(i: String, p: String, g: Date, e: String, pw: String)
+dodajVoznju(v: Voznja): bool
+obrisiVoznju(v: Voznja): bool
+izmjeniVoznju(v: Voznja): bool
+dodajUposlenika(korisnik: Korisnik): bool
+izmijeniUposlenika(korisnik: Korisnik): bool
+obrisiUposlenika(korisnik: Korisnik): bool
+posaljiPoruku(): String
+procitajPoruku(): String
Proxy
+dajHashiranePodatke()
Kupac
+DajMojeVoznje(): List<Voznja>
+KupiKartu(): List<Karta>
+OtkaziMiKartu(): bool
«create»+Kupac(i: String, p: String, g: Date, e: String, pw: String)
-obracunajPopust(): double
+posaljiPoruku(): String
+procitajPoruku(): String
1
0...*
0...*