Elektrotehnički fakultet Sarajevo

EBANK

PATERNI PONAŠANJA

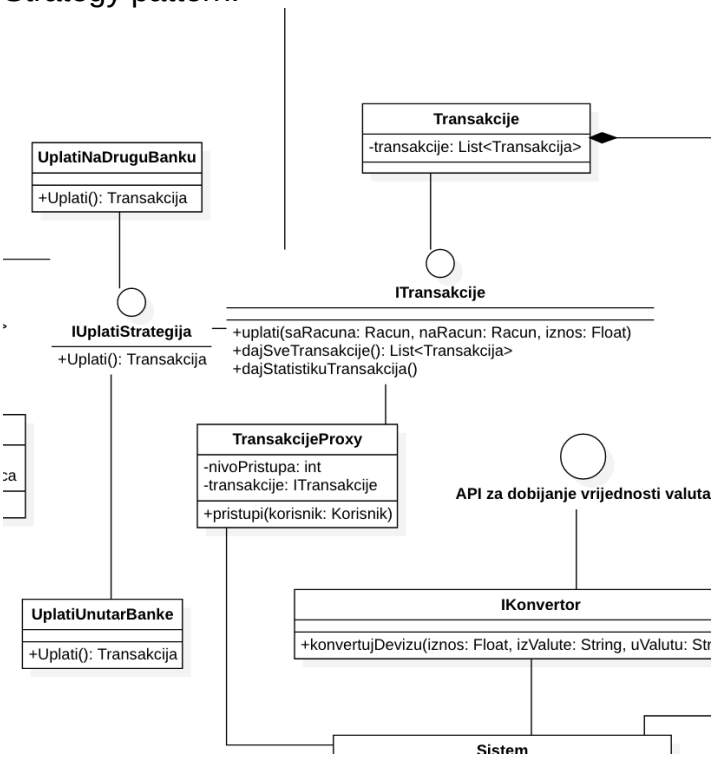
Predmet: Objektno orijentisana analiza i dizajn

Grupa: **Bankari**
Enver Suljić
Berina Suljić
Amina Šiljak

10. maj 2020. u Sarajevu

1. Strategy pattern

Strategy patern izdvaja algoritam iz matične klase i uključuje ga u posebne klase. Pogodan je kada postoje različiti primjenjivi algoritmi (strategije) za neki problem. Ovaj patern smo primijenili kod metode uplati u interfejsu ITransakcije tako što smo dodali interfejs IUplatiStrategija koji implementiraju klase UplatiNaDruguBanku i UplatiUnutarBanke. Ovo smo primijenili jer metoda uplati se treba implementirati različito u zavisnosti da li je klijent odabrao uplatu na račun druge banke ili uplatu na račun EBANK.
Ovakvu funkcionalnost smo ranije bili zanemarili u našem dijagramu klasa, iako smo je imali na dijagramima aktivnosti i dijagramu slučajeva upotrebe. Čitajući o strategy paternu smo shvatili da nam dio funkcionalnosti nedostaje, te smo iz klase Račun naslijedili klasu RačunEBANK koja označava račun unutar naše banke. Zatim smo ovaj patern primijenili da bi metodu uplati izvršavali drugačije u zavisnosti od načina uplate.
Na sljedećoj slici se nalazi prikaz sa klasnog dijagrama gdje smo primijenili Strategy pattern:



2. State pattern

State Pattern je dinamička verzija Strategy paterna. Objekat mijenja način ponašanja na osnovu trenutnog stanja. Ovaj patern nismo iskoristili jer nemamo algoritam koji bi se izvrsavao na različite načine u odnosu na neko stanje. U našem projektu npr. Imamo 3 stanja kredita: u zahtjevu, aktivan i završen. Kada bi imali neku metodu koja se izvršava drugačije u zavisnosti od stanja kredita, mogli bi iskoristiti ovaj patern. Ili, kada bi imali atribut hitno u transakciji, mogli bismo vršiti uplate različito u zavisnosti od toga da li je uplata hitna ili ne.

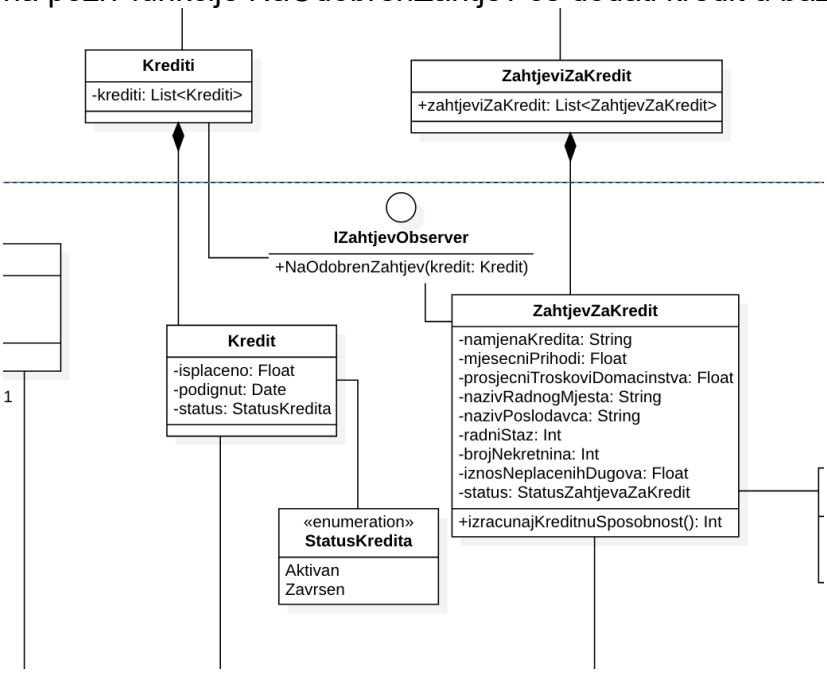
3. TemplateMethod patern

Omogućava izdvajanje određenih koraka algoritma u odvojene podklase. Struktura algoritma se ne mijenja - mali dijelovi operacija se izdvajaju i ti se

dijelovi mogu implementirati različito. U našem slučaju, kada bismo imali funkcionalnost da sortiramo listu transakcija po nekom kriteriju, ovaj patern bismo mogli iskoristiti za različite kriterije. Tako bismo za sortiranje po vremenu mogli koristiti jednu klasu, za sortiranje po iznosu drugu, itd.

4. Observer patern

Uloga Observer paterna je da uspostavi relaciju između objekata tako kada jedan objekat promijeni stanje drugi zainteresirani objekti se obavještavaju. U našem projektu, observer patern smo iskoristili kada neki od zahtjeva za kredit biva odobren. Klasa Krediti će implementirati interfejs IZahtjevObserver, na poziv funkcije NaOdobrenZahtjev će dodati kredit u bazu.



Također ovaj patern smo iskoristili kada stanje transakcije pređe iz UToku u Provedena. Pomoću njega će se sa jednog računa skinuti iznos transakcije, a na drugi će se isti iznos položiti.

5. Iterator patern

Iterator patern omogućava sekvencijalni pristup elementima kolekcije bez poznavanja kako je kolekcija strukturirana. Implementirat ćemo ovaj patern tako što ćemo interfejs IEnumerable implementirati u naše interfejse IOglasaPloča, ITransakcije, IFilijaleBankomati, IRacuni, iKrediti, IZahtjeviZaKredit, IBankari i IKlijenti. Pošto njega implementiraju i proxy i DAO klase, u proxy-u će getEnumerator() samo proslijediti vrijednost one iz DAO, a ona iz DAO će vratiti iterator liste stavki koju dobije iz baze.