Elektrotehnički fakultet Sarajevo

EBANK

STRUKTURALNI I KREACIJSKI PATERNI

Predmet: Objektno orijentisana analiza i dizajn

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

6. maj 2020. u Sarajevu

STRUKTURALNI PATERNI

1. **Adapter patern**
   Adapter patern služi da se postojeći objekat prilagodi za korištenje na neki novi način u odnosu na postojeći rad, bez mijenjanja same definicije objekta. Na taj način obezbjeđuje se da će se objekti i dalje moći upotrebljavati na način kako su se dosad upotrebljavali, a u isto vrijeme će se omogućiti njihovo prilagođavanje novim uslovima. Ovaj patern nam nije bio potreban u našem dijagramu klasa, pa ga nismo ni iskoristili. Hipotetički bi mogao biti koristan kada bismo pravili funkcionalnost koja bi dozvolila da se iz nekog drugog bankarskog sistema prebaci račun u naš.

2. **Facade patern**
   Fasadni patern služi kako bi se klijentima pojednostavilo korištenje kompleksnih sistema. Klijenti vide samo fasadu, odnosno krajnji izgled objekta, dok je njegova unutrašnja struktura skrivena. Na ovaj način smanjuje se mogućnost pojavljivanja grešaka jer klijenti ne moraju dobro poznavati sistem kako bi ga mogli koristiti Dodali smo klasu FilijaleIBankomatiFasada koja ima dvije metode napraviFilijalu() i napraviBankomat(). Na taj način, korisnik ne mora dobro poznavati sistem kako bi ga mogao koristiti, ali i dalje će moći koristiti klase Bankomat i Filijala.

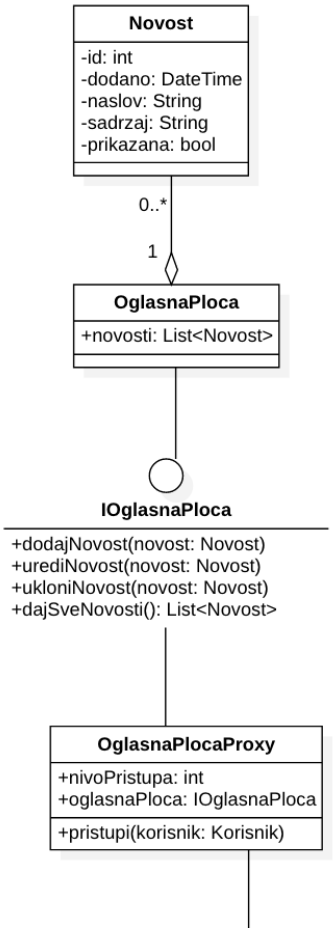
3. **Decorator patern**
   Decorator patern služi za omogućavanja različitih nadogradnji objektima koji svi u osnovi predstavljaju jednu vrstu objekta (odnosno, koji imaju istu osnovu). Umjesto da se definiše veliki broj izvedenih klasa, dovoljno je omogućiti različito dekoriranje objekata (tj. dodavanje različitih detalja), te se na taj način pojednostavljuje i rukovanje objektima klijentima, i samo implementiranje modela objekata. Mi ga nismo iskoristili, jer nije bilo potrebe.

4. **Bridge patern**
   Bridge patern služi kako bi se apstrakcija nekog objekta odvojila od njegove implementacije. Ovaj patern veoma je važan jer omogućava ispunjavanje Open-Closed SOLID principa, odnosno uz poštivanje ovog paterna omogućava se nadogradnja modela klasa u budućnosti te osigurava da se neće morati vršiti određene promjene u postojećim klasama. Ni ovaj patern nismo nigdje iskoristili, jer nemamo dvije klase koje koriste istu metodu na drugaciji nacin. Hipotetički, ovaj patern bismo mogli iskoristiti kada bismo imali klase VISA, MasterCard i Maestro, u svakoj od kojih se nalazi metoda dajRaspolozivaSredstva(). Tu metodu bismo mogli iskoristiti da primijenimo Bridge patern.

5. **Composite patern**
   Composite patern služi za kreiranje hijerarhije objekata. Koristi se kada svi objekti imaju različite implementacije nekih metoda, no potrebno im je svima pristupati na isti način, te se na taj način pojednostavljuje njihova implementacija. Njega smo mogli iskoristiti u slučaju bankomata i filijala. Kada bismo imali zajedničke metode u te dvije klase, ovaj patern bi bio pogodniji od Flyweight paterna.

6. Proxy patern

Proxy patern služi za dodatno osiguravanje objekata od pogrešne ili zlonamjerne upotrebe. Primjenom ovog paterna omogućava se kontrola pristupa objektima, te se onemogućava manipulacija objektima ukoliko neki uslov nije ispunjen, odnosno ukoliko korisnik nema prava pristupa traženom objektu



Ovaj patern smo iskoristili u više slučajeva, a ovdje ćemo navesti par njih:

- Klasa OglasnaPloca ima četiri metode: dodajNovost, urediNovost, ukloniNovost, dajSveNovosti. Ovim metodama ne bi trebali pristupati svi korisnici. Na osnovu nivoa pristupa možemo osigurati da administrator može dodavati, uređivati, ukloniti i vidjeti sve novosti, dok klijent može samo vidjeti novosti, a ne i koristiti ostale metode. Zato smo ovdje iskoristili Proxy patern tako što smo dodali klasu OglasnaPlocaProxy.
- Korisnik može pozvati metodu uplati nad klasom Transakcije preko klase TransakcijeProxy i tako pokrenuti transakciju. To može izvršiti samo korisnik koji je klijent. Klasom TransakcijeProxy smo obezbijedili da samo korisnik može vršiti plaćanje.
- Bankar može pozvati metodu dajSveKredite() i pokreniKredit(), Administrator može pozvati metodu dajSveKredite(), dok Klijent ne može pozvati niti jednu od ovih metoda. Koristeći Proxy patern onemogućili smo nekim korisnicima da koriste neke metode, dok smo pojedinim omogućili. Time je uspostavljen neki nivo pristupa ovim metodama.

Iskoristili smo Proxy patern u jako puno slucajeva koji su slicni, tako da smatramo da je njihovo navodjenje bespotrebno. Svakako, bit ce prikazano na dijagramu klasa. Jedna od primjena ovog paterna u našem dijagramu klasa prikazana je na slici.

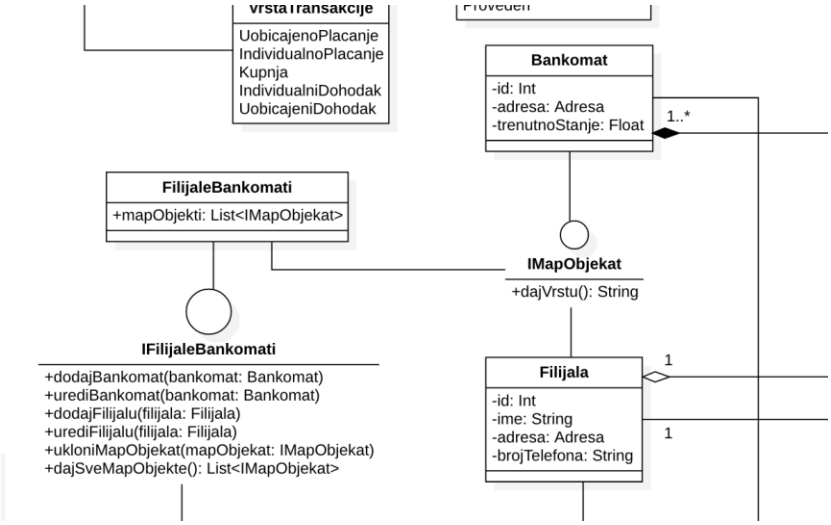
7. Flyweight patern

Flyweight patern koristi se kako bi se onemogućilo bespotrebno stvaranje velikog broja instanci objekata koji svi u suštini predstavljaju jedan objekat. Samo ukoliko postoji potreba za kreiranjem specifičnog objekta sa jedinstvenim karakteristikama (tzv. specifično stanje), vrši se njegova instantacija, dok se u svim ostalim slučajevima koristi postojeća opća instanca objekta (tzv. bezlično stanje). Korištenje ovog paterna veoma je korisno u slučajevima kada je potrebno vršiti uštedu memorije.

Ovaj patern smo uspjeli primijeniti u našem projektu. Klasa FilijaleIBankomati (klasa koja je generalizacija dvije klase: Filijala i Bankomat) u sebi je imala dvije liste: listu

bankomata i listu filijala, a takodjer je imala i metode dajBankomate(), dajFilijale(), kao i ukloniBankomat(), ukloniFilijalu(). Primijenili smo Flyweight patern tako sto smo dodali interfejs IMapObjekat koji je povezan sa klasama Filijala i Bankomat, kao i sa klasom FilijaleIBankomati. Taj interfejs ima metodu dajVrstu() umjesto dvije metode za filijale i bankomate. Sada, unutar interfejsa IFilijaleIBankomati, sada imamo metode dajSveMapObjekte() i ukloniMapObjekat().

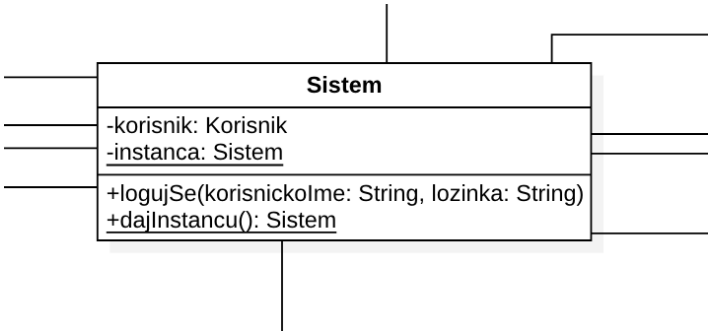
Na sljedecoj slici vidimo primjenu ovog paterna u našem klasnom dijagramu:



KREACIJSKI PATERNI

1. **Singleton patern**
   Singleton patern služi kako bi se neka klasa mogla instancirati samo jednom. Na ovaj način može se omogućiti i tzv. lazy initialization, odnosno instantacija klase tek onda kada se to prvi put traži.

   Ovaj patern smo iskoristili u našoj glavnoj klasi Sistem. U njoj ima privatna statička varijabla instance koja označava jedinu instancu klase. Nju možemo dobiti pomoću javne statičke funkcije dajInstancu().



2. **Prototype patern**
   Prototype patern omogućava smanjenje kompleksnosti kreiranja novog objekta tako što se uvodi operacija kloniranja. Na taj način prave se prototipi objekata koje je moguće replicirati više puta a zatim naknadno promijeniti jednu ili više karakteristika, bez potrebe za kreiranjem novog objekta nanovo od početka. Mi ga nismo iskoristili, jer nije bilo potrebe.

3. **Factory Method patern**
   Factory method patern služi za omogućavanje instanciranje različitih vrsta podklasa koristeći factory metodu koja odlučuje koja će se podklasa instancirati i koja programska logika izvršiti. Na ovaj način osigurava se ispunjavanje O SOLID principa, jer se kod za kreiranje objekata različitih naslijeđenih klasa ne smješta samo u jednu zajedničku metodu, već svaka podklasa ima svoju logiku za instanciranje željenih klasa, a samo instanciranje kontroliše factory metoda koju različite klase implementiraju na različit način. Mi ga nismo iskoristili, jer nije bilo potrebe.

4. **Abstract Factory patern**
   Abstract factory patern služi kako bi se izbjeglo korištenje velikog broja if-else uslova pri kreiranju različitih hijerarhija objekata. Ukoliko postoji više tipova istih objekata te različite klase koriste različite podtipove, te klase postaju fabrike za kreiranje objekata zadanog podtipa bez potrebe za specificiranjem pojedinačnih objekata. Na ovaj način se, korištenjem nasljeđivanja, ukida potreba za postojanjem if-else uslova jer određeni tip fabrike sadrži određene tipove objekata i zna se tačno koju podklasu će instancirati. Mi ga nismo iskoristili, jer nije bilo potrebe.

5. **Builder patern**

Builder patern služi za apstrakciju procesa konstrukcije objekta, kako bi se kao rezultat mogle dobiti različite specifikacije objekta koristeći isti proces konstrukcije. Ovaj patern koristi se kako bi se izbjeglo kreiranje kompleksne hijerarhije klasa te kako bi se izbjegao kompleksni programski kod konstruktora jedne klase koja može imati različite konfiguracije atributa.

Ovaj patern smo iskoristili kod Transakcije. Napravili smo klasu TransakcijaBuilder koja ima sve potrebne metode da napravi novu transakciju od datih informacija, i metodu dajTransakciju() koja vraća Transakciju napravljenu sa tim informacijama.

Ovo je prikazano na sljedećoj slici: