PATERNI PONAŠANJA

Strategy patern

Strategy patern izdvaja algoritam iz matične klase i uključuje ga u posebne klase. Pogodan je kada postoje različiti primjenjivi algoritmi (strategije) za neki problem. Strategy patern omogućava klijentu izbor jednog od algoritma iz familije algoritama za korištenje.
Ovaj patern bi mogli iskoristiti pri plaćanju premium paketa. Uveli bismo klasu Placanje koja bi bila naša Context klasa. Sljedeće bi bilo potrebno napraviti interfejs IPlacanje koji bi bio naš IStrategy interfejs. Iz njega bismo naslijedili različite vrste plaćanja, plaćanje jedan mjesec, plaćanje na godišnjem nivou sa prilagođenom manjom cijenom, opcija subscribe(svaki mjesec se automatski skida taj iznos sa računa) itd.

State patern

State Pattern je dinamička verzija Strategy paterna. Objekat mijenja način ponašanja na osnovu trenutnog stanja. Postiže se promjenom podklase unutar hijerarhije klasa.
Ovaj patern bismo mogli iskoristiti ako želimo da korisnicima koji su duže tu pružimo mogućnost putovanja na nekih od premium destinacija kao nagradu što su dugo učlanjeni. Tada bi Destinacija bila naša Context klasa, a imali bismo i interfejs IDostupneDestinacije iz kojeg bi bile naslijeđene vrste destinacija: dostupneSvima, dostupneNakonPolaGodine, dostupneNakonGodinu i slično. Sada bi se nakon određenog perioda automatski uradila izmjena dostupnih destinacija za korisika.

Template method 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.
Ovaj patern bismo mogli iskoristiti kada bismo administratoru željeli omogućiti različite prikaze mačaka. Ovaj patern je ovdje povoljan da ne

bismo duplicirali kod koji je u osnovu isti algoritam za sortiranje samo po različitom kriteriju.

U klasi Macka bismo imali metodu sortiraj koja predstavlja našu templateMethod i ona neće biti override-ana. U Macka bismo dodali i neke metode koje će biti override-ane zavisno na koji način klase naslijeđene iz nje sortiraju(npr metoda porediPoUslovu). Klasu Macka bi naslijedilo nekoliko klasa KompetentnostSort, AlergijeSort,.. koje bi također posjedovale tu metodu porediPoUslovu.
Ovo bi pomoglo adminu da ima pregledniji prikaz mačaka koje čekaju na određivanje kompetentnosti, pregled onih kod kojih je već završeno određivanje, pregled mačaka sa alergijama, zdravstvenim problemima i slično te sortiranim i filtriranim u prikazu shodno tome.

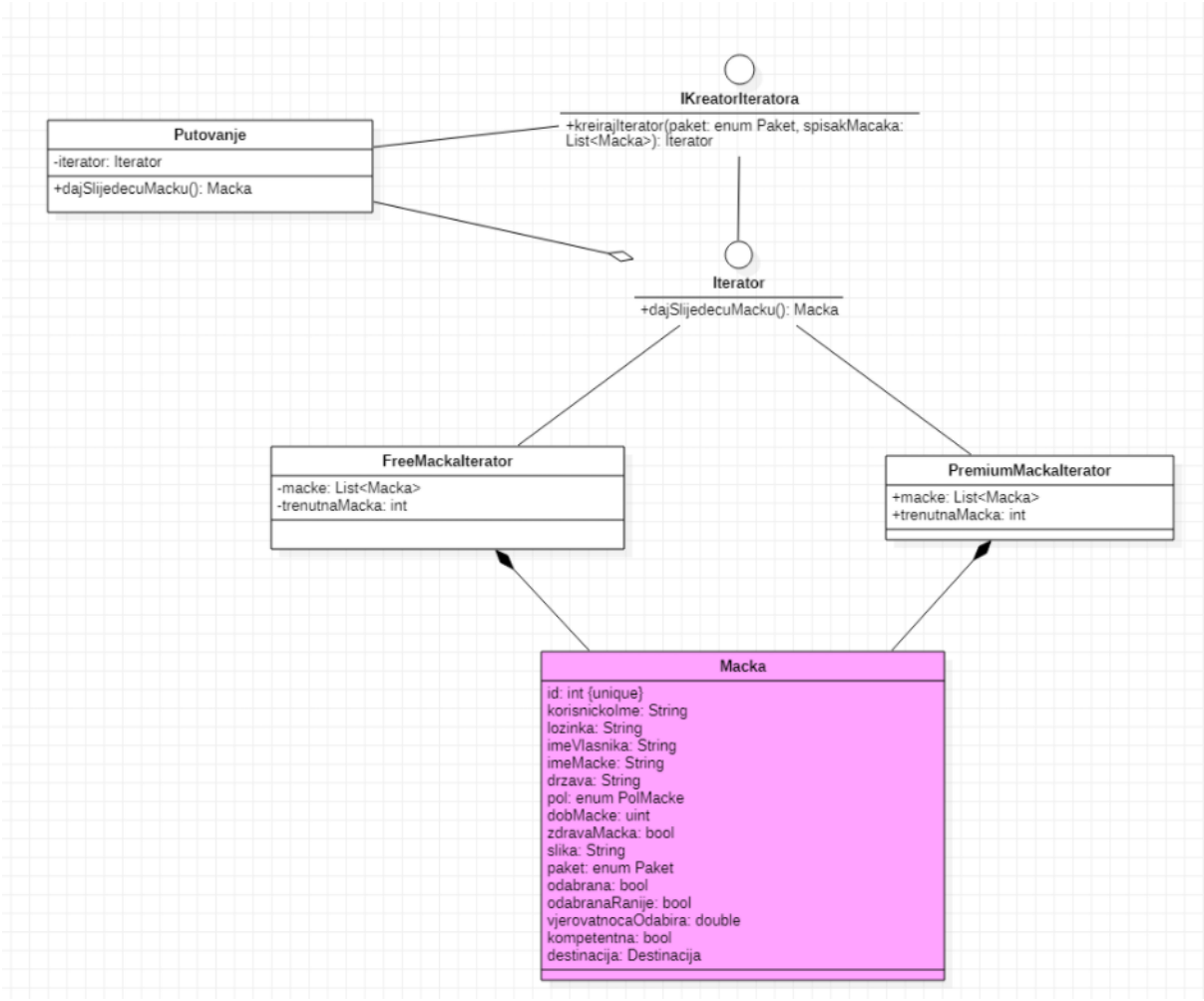
Observer patern

Uloga Observer paterna je da uspostavi relaciju između objekata tako kada jedan objekat promijeni stanje drugi zainteresirani objekti se obavještavaju.
Ovaj patern bismo mogli primijeniti ako bismo željeli da korisnici budu obaviješteni kada se doda neka nova destinacija. Ovdje bi nam nova Destinacija predstavljala Subject. Trebao bi nam i interfejs IObserver te naslijeđena klasa Observer koja bi predstavljala našu mačku tj njen profil koji bi bio obaviješten o dodanoj destinaciji putem poruke.

Iterator patern

Iterator patern omogućava sekvencijalni pristup elementima kolekcije bez poznavanja kako je kolekcija strukturirana.

Pošto će se mačke koje idu na putovanje koristiti često(novosti, historija letova,..) možemo iskoristiti listu objekata Mačka u klasi Putovanje za iterator patern. MačkeZaPutovanjeIterator će imati kao privatni atribut listu mačaka odabranih za to putovanje. Kroz tu listu će se iterirati. Bit će nam potreban i interfejs koji će ona implementirati IEnumerable koji ima metodu getEnumeration().



Chain of responsibility patern

Chain of responsibility patern namijenjen je kako bi se jedan kompleksni proces obrade razdvojio na način da više objekata na različite načine procesiraju primljene podatke.
Mi bismo ovaj patern mogli iskoristiti ako bismo željeli u aplikaciju omogućiti priliku da dvije mačke idu zajedno u svemir(medeni mjesec za mačke). Da bi se odobrilo ovo putovanje potrebno je da ga potvrde obje mačke i admin. Jedna od maca će poslati zahtjev drugoj za putovanje pa će nam trebati klasa ZahtjevPotvrda, interfejs IHandler, klase koje naslijeđuju ZahtjevPotvrda koje će obrađivati zahtjev.

Medijator patern

Mediator patern enkapsulira protokol za komunikaciju među objektima dozvoljavajući da objekti komuniciraju bez međusobnog poznavanje interne strukture objekta.
Ovaj patern bismo mogli iskoristiti kod utisaka putovanja. Naša aplikacija bi zahtijevala da utisak može ostaviti samo mačka koja je već putovala. Klasa Mačka bi imala atribut tipa IUtisakMedijator. Trebao bi nam novi interfejs IUtisakMedijator. On bi imao metode sa funkcionalnostima provjere koja mačka piše utisak i koji je sadržaj utiska.