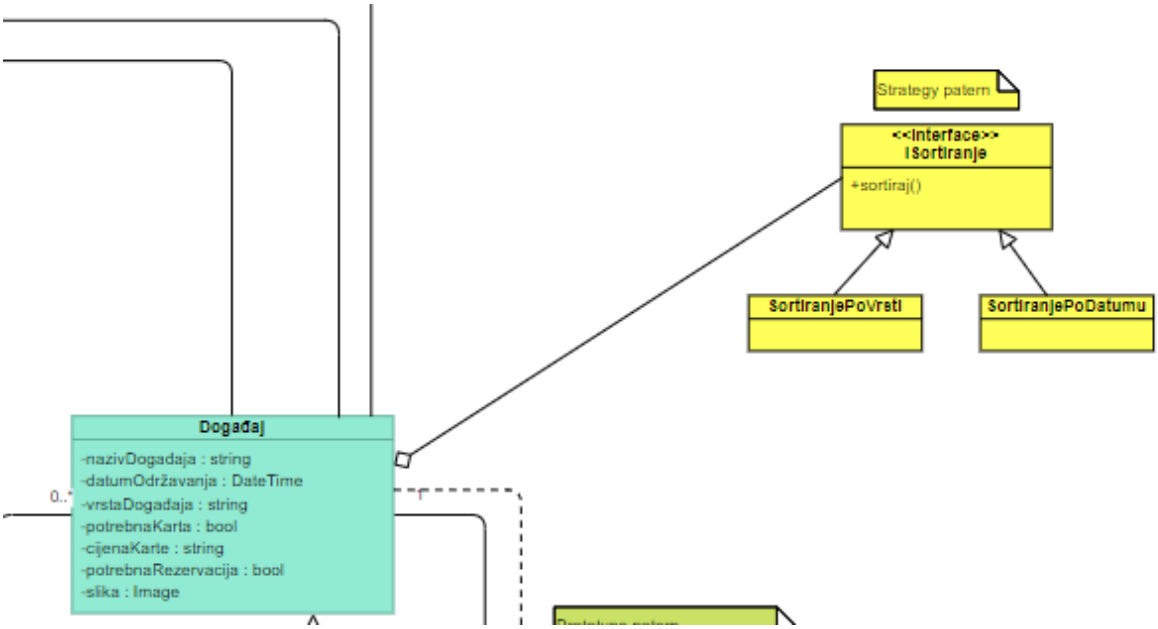
Paterni ponašanja

1. Strategy patern

Strategy patern izdvaja algoritam iz matične klase i uključuje ga u posebne klase. On omogućava klijentu izbor jednog algoritma iz familije algoritama za korištenje. Algoritmi su neovisni od klijenata koji ih koriste.

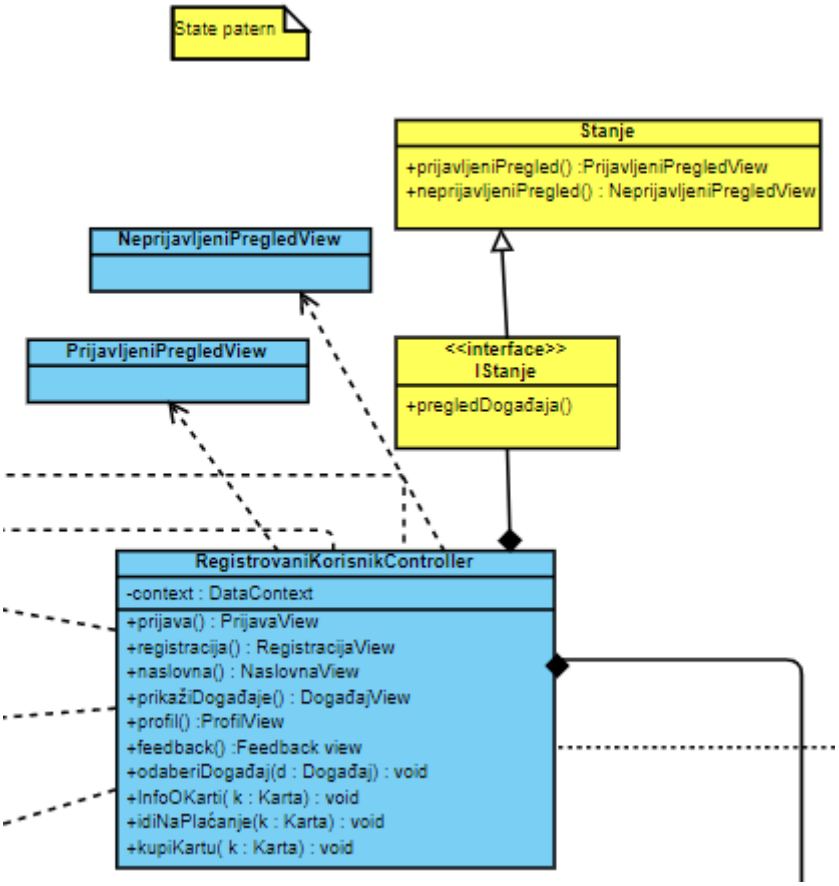
U našem sistemu ovaj patern možemo koristiti kod filtriranja događaja. Korisnik ima mogućnost filtriranja događaja po vrsti ili po datumu. Kako bismo omogućili ovu funkcionalnost koristit ćemo Strategy patern.



2. State patern

State patern je dinamička verzija Strategy paterna. Objekat mijenja način ponašanja na osnovu trenutnog stanja. Postiže se primjenom podklase unutar hijerarhije klasa.

State patern u našem sistemu možemo koristiti za različit prikaz događaja, zavisno od toga da li je korisnik prijavljen na sistem ili nije. Neprijavljeni korisnik može pregledati događaje, ali ne kupovati kartu ili rezervisati mjesto za njih, za razliku od prijavljenog. Implementirali smo interfejs Istanje koji će prikazivati odgovarajući pogled zavisno od toga da li je korisnik prijavljen na sistem ili nije. U tu svrhu implementirana su i još dva pogleda NeprijavljeniPregledView i PrijavljeniPregledView.



3. Template method patern

Ovaj 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 sistemu nismo naišli na potrebu za implementacijom ovog paterna, ali kada bi se u sistem dodala funkcionalnost pamćenja svakog komentara i ocjene za pojedini događaj moglo bi se dodati neko filtriranje ili sortiranje podataka. U Strategy paternu je već korišteno sortiranje ali ovdje bi ono bilo nešto komplikovanije te bi bilo pogodno da se rastavi na dijelove.

4. Observer patern

Observer patern uspostavlja relaciju između objekata takvu da kad se stanje jednog objekta promijeni svi vezani objekti dobiju informaciju.

Ovaj patern u našem sistemu bi se mogao iskoristiti za funkcionalnost ostvarivanja dovoljnog broja bodova za popust. Nakon što korisnik ostvari pravo na popust o tome može biti obaviješten, kao i o načinu realizacije svog popusta.

5. Iterator patern

Iterator patern omogućava pristup elementima kolekcije sekvencijalno bez poznavanja interne strukture kolekcije.

Ovaj patern bismo mogli iskoristiti ako bi umjesto listi kao atribut imali mapu korisnika i njihovih kupljenih karata i rezervisanih mjesta, te ako bismo npr. željeli naći korisnike koji nemaju niti jednu kupljenu kartu odnosno rezervisano mjesto.

6. Chain of responsibility patern

Patern predstavlja listu objekata, ukoliko objekat ne može da odgovori proslijeđuje zahtjev narednom nizu.

Ovaj patern bi mogli iskoristiti kod rezervacije mjesta za određeni događaj. Ovaj patern bi primijenili kako bi bili sigurni u to da korisnik koji je registrovan na naš sistem vrši rezervaciju mjesta. Handleri bi vršili procesiranje podataka i odlučili o tome da li će zahtjev biti proslijeđen.

7. Mediator patern

Medijator patern enkapsulira protokol za komunikaciju među objektima dozvoljavajući da objekti komuniciraju bez međusobnog poznavanja interne strukture objekta.

Ovaj patern bi mogli iskoristiti kod davanja feedback-a(ocjena ili komentar) za događaj.

Implementirale bi interfejs IMediator koji bi provjeravao da li je registrovani korisnik uopće prisustvovao događaju za koji želi da da ocjenu/komentar.

U postojeću klasu RegistrovaniKorisnik bi dodali atribut koji je tipa IMediator.