Paterni ponašanja

Strategy patern

Strategy patern izdvaja algoritam iz matične klase i uključuje ga u posebne klase. Omogućava fleksibilan izbor algoritama bez izmjene klijentskog koda. Klijent bira odgovarajuću strategiju, a algoritmi su enkapsulirani i implementirani kroz zajednički interfejs. Ovaj patern zamjenjuje nasljeđivanje kompozicijom, što čini kod fleksibilnijim i lakšim za proširivanje.

Funkcionalnost "Prijedlog i prijava na treninge" koristi algoritam koji preporučuje treninge na osnovu različitih parametara kao što su nivo jahača, prethodno jahani konji, treneri i termini. Trenutno je ovaj algoritam hardkodovan u jednoj klasi, što otežava dodavanje novih strategija preporučivanja.

Primjena:

- Kreiranje interfejsa `TrainingRecommendationStrategy`
- Implementacija različitih strategija: `BeginnerRecommendationStrategy`, `AdvancedRecommendationStrategy`, `PreferenceBasedStrategy`
- Omogućava lako dodavanje novih algoritma preporučivanja bez mijenjanja postojećeg koda
- Različiti tipovi članova mogu koristiti različite strategije preporučivanja

State patern

State patern omogućava objektima da mijenjaju ponašanje u zavisnosti od svog unutrašnjeg stanja. Svako stanje je predstavljeno kao posebna klasa, a klasa konteksta delegira ponašanje trenutnom stanju. Ovo pojednostavljuje kod i čini ga lakšim za održavanje, posebno kada objekat može imati kompleksne promjene ponašanja u zavisnosti od stanja.

Članarina ima različita stanja koja utiču na dostupne funkcionalnosti - aktivna, neaktivna. Trenutno se ova logika rješava kroz if-else provjere što može dovesti do kompleksnog i teško održljivog koda.

Primjena:

- Kreiranje stanja: `ActiveMembershipState`, `InactiveMembershipState`, `TrialMembershipState`
- Svako stanje definiše koje funkcionalnosti su dostupne članu
- Automatski prelazak između stanja kada se članarina produžuje ili istekne
- Pojednostavljuje upravljanje pristupom funkcionalnostima na osnovu stanja članarine

Template Method patern

Template Method patern definiše kostur algoritma u nadklasi, a prepušta podklasama da implementiraju specifične korake. Pruža mogućnost ponovne upotrebe koda i prilagodbu dijelova algoritma. Ovaj patern je povezan sa nasljeđivanjem, te nadklasa kontroliše tok izvršavanja i poziva metode podklasa kada je potrebno.

Kreiranje treninga i dodavanje trailova dijele sličnu strukturu - oboje zahtijevaju validaciju, odabir konja, postavljanje datuma i vremena. Trenutno se ova logika duplicira.

Primjena:

- Kreiranje nadklase `ActivityCreationTemplate` sa metodom `createActivity()`
- Podklase `TrainingCreation` i `TrailCreation` implementiraju specifične korake
- Omogućava ponovnu upotrebu zajedničke logike za kreiranje aktivnosti
- Lakše dodavanje novih tipova aktivnosti u budućnosti

Observer patern

Observer patern uspostavlja jednosmjernu zavisnost između objekata tako da promjena stanja u jednom objektu automatski obavještava sve njegove posmatrače (observers). Koristan je za implementaciju događajnog sistema gdje jedan objekat (subject) može imati više posmatrača koji se automatski ažuriraju kada se stanje promijeni. Ovaj patern omogućava skalabilnost i odvojenost subjekta i posmatrača.

Kada se promijeni nivo člana, potrebno je obavijestiti različite dijelove sistema - ažurirati preporučene treninge, poslati email obavještenje, ažurirati dashboard. Trenutno se ove akcije pozivaju direktno iz koda za mijenjanje nivoa.

Primjena:

- Kreiranje `MemberLevelSubject` koji obavještava posmatrače o promjenama
- Implementacija observer-a: `TrainingRecommendationObserver`, `EmailNotificationObserver`, `DashboardUpdateObserver`
- Omogućava dodavanje novih funkcionalnosti koje reaguju na promjenu nivoa bez mijenjanja postojećeg koda

Iterator patern

Iterator patern omogućava sekvencijalni pristup elementima kolekcije bez otkrivanja njene interne strukture. Podržava više istovremenih iteratora nad istom kolekcijom. Klijent koristi zajednički interfejs, bez potrebe da zna konkretni tip kolekcije. Ovaj patern čini kod nezavisan od konkretne implementacije kolekcije i omogućava jednostavno iteriranje kroz različite tipove struktura podataka.

Pregled različitih kolekcija (konji, treneri, članovi, treninzi) zahtijeva sekvencijalni pristup elementima. Trenutno se koriste običajne petlje što čini kod zavisan o konkretnom tipu kolekcije.

Primjena:

- Kreiranje iteratora za različite kolekcije: `HorseIterator`, `TrainerIterator`, `MemberIterator`
- Omogućava jedinstveni pristup iteriranju bez obzira na tip kolekcije
- Podržava više istovremenih iteratora nad istom kolekcijom
- Olakšava dodavanje filtriranja i sortiranja

Command patern

Command patern enkapsulira zahtjev kao objekat, čime omogućava parametarsko pokretanje operacija, kao i mogućnosti undo/redo, red čekanja i zabilježavanje. Ovaj patern razdvaja pošiljaoca od izvršioca akcije i koristi se gdje je potrebno odvojeno upravljanje operacijama. Komande mogu biti pohranjene, proslijeđene kao parametri, i izvršene u različito vrijeme.

Admin funkcionalnosti (upravljanje konjima, trenerima, trailovima) zahtijevaju mogućnost undo/redo operacija. Također, operacije poput brisanja članova ili konja trebaju biti reverzibilne.

Primjena:

- Enkapsulacija svake admin operacije kao komandu: `AddHorseCommand`, `DeleteMemberCommand`, `UpdateTrainerCommand`
- Implementacija `CommandInvoker` klase za izvršavanje komandi
- Mogućnost čuvanja historije komandi za undo/redo funkcionalnost