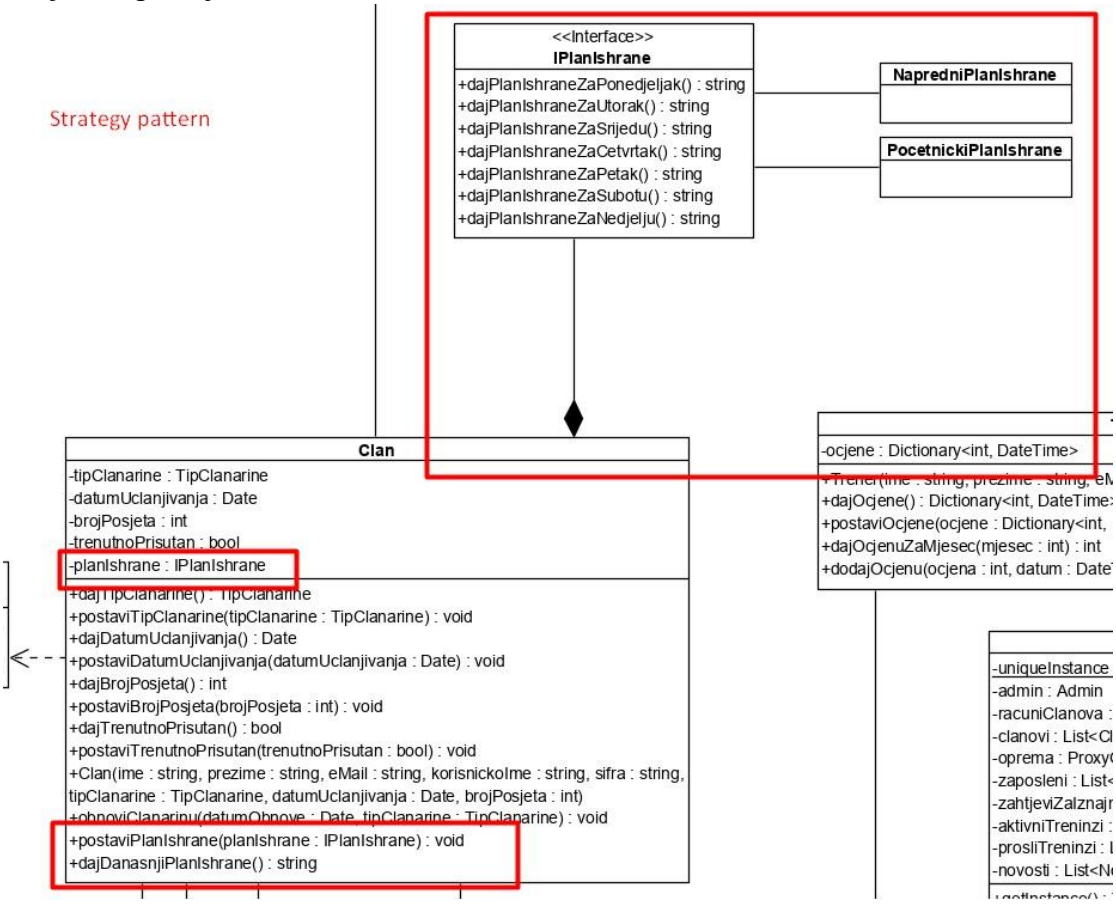
Patterni ponašanja

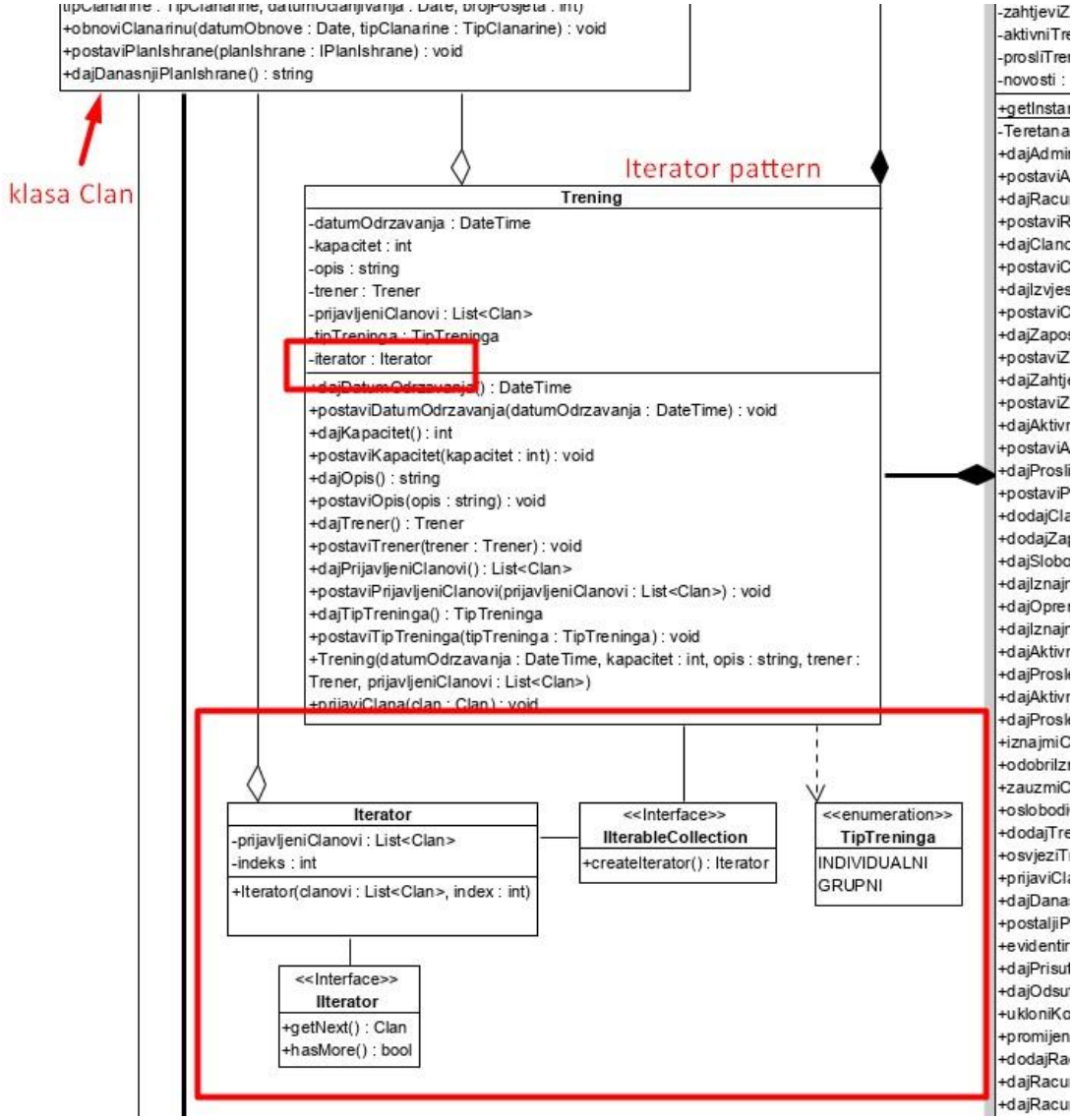
1. Strategy pattern: Ne vidimo primjenu ovog patterna u sklopu trenutnog modela našeg sistema te ćemo iz tog razloga razmotriti hipotetski slučaj. Zamislimo da smo odlučili dati opciju članu da svaki dan može dobiti plan ishrane u obliku određene liste te da član može odabrati koji plan ishrane će pratiti od više njih. To bismo realizovali na sljedeći način: kreirali bismo interfejs IPlanIshrane koji sadrži metode dajPlanIshraneZaPonedjeljak, dajPlanIshraneZaUtorak itd. Klase koje bi implementirale ovaj interfejs mogu biti na primjer klase PocetnickiPlanIshrane i NapredniPlanIshrane, a članu bi se dala opcija da odabere koji plan ishrane njemu odgovara. U klasi Clan bismo čuvali atribut tipa IPlanIshrane kojem bi se pomoću settera dodijelila konkretna klasa, te bi član mogao pristupiti svom dnevnom planu pomoću metode dajDanasnjiPlanIshrane koja bi na osnovu dana pozvala odgovarajuću metodu iz odgovarajuće klase. Dobra strana ovog patterna jeste što on omogućava Open/Closed princip jer vidimo da se novi planovi ishrane mogu dodavati bez ikakvih izmjena u postojećim klasama.



2. State pattern: Prethodni primjer smo mogli zamisliti i kao state pattern da smo umjesto toga da član bira plan ishrane, koji bi u ovom slučaju predstavljao stanje, rekli da svaki član započinje sa PocetnickiPlanIshrane, nakon 2 mjeseca prelazi na AmaterskiPlanIshrane te nakon još tri mjeseca prelazi na NapredniPlanIshrane. Ove izmjene se odvijaju automatski u ovisnosti od vremena koje je prošlo.
3. Template Method pattern: Za realizaciju ovog patterna smo zamislili apstraktnu klasu ObracunPlate koja bi posjedovala metode dajOsnovicu i dajDodatak te metodu dajPlatu. Prve dvije metode bi bile apstraktne te bi se time forsirala svaka klasa koja

naslijedi ovu klasu da napiše svoju implementaciju tih metoda. Ove dvije metode nemaju parametrra i povratni tip im je double, a vraćaju redom osnovicu i dodatak na platu. Metoda dajPlatu ima svoju implementaciju, a to je samo da vraća zbir rezultata prethodne dvije metode. Ova klasa bi zatim bila naslijeđena od strane klasa Trener, Recepcioner i Admin, odnosno od strane klasa koje predstavljaju uposlenike teretane, te bi svaka ta klasa implementirala metode dajOsnovicu i dajDodatak na odgovarajući način, npr. Metoda dajOsnovicu može za svaku od ovih klasa vraćati neku konstantnu vrijednost koja se razlikuje za različite klase, dok bi se metoda dajDodatak određivala dinamički na osnovu nekih podataka za odgovarajući mjesec.

4. Observer pattern: Ukoliko bismo htjeli da realizujemo to da svaki član dobije poruku o održavanju treninga od strane trenera kod kojeg je već nekada prije bio na treninzima, to bismo uradili na sljedeći način: prvo bismo kreirali interfejs IObserver sa jednom metodom primiPoruku koja ima jedan parametar tipa ISubject, a kao rezultat ne vraća ništa. ISubject je također jedan interfejs i on sadrži metode obavijesti bez parametara koja također ne vraća ništa, zatim metode dodajObservera i ukloniObservera koje imaju jedan parametar tipa IObserver. Klasa Trener bi implementirala interfejs ISubject te bih posjedovala jedan dodatni atribut koji bi predstavljao listu objekata koji implementiraju interfejs IObserver. Metodom dodajObservera i ukloniObservera bi respektivno dodavali i brisali objekat iz liste koji je primljen kao parametar. Unutar metode obavijesti bi se za svaki objekat iz liste pozvala metoda primiPoruku u kojoj se šalje referenca na trenera iz kojeg se metoda poziva (this). Interfejs IObserver implementira klasa Clan, a kao implementaciju metode primiPoruku će se kreirati određena vrsta poruke odnosno obavijesti da trener (koji je primljen kao parametar) održava novi trening te će se kreirana poruka dodati u listu poruka (stringova) sadržanu u klasi Clan.
5. Iterator pattern: Ovaj pattern možemo iskoristiti za prolazak kroz listu članova sadržanu unutar klase Trening. U tu svrhu, potrebno je kreirati klasu Iterator koji će implementirati interface IIterator. Ova klasa će kao atribute imate listu tipa Član i indeks trenutnog elementa. Pored toga, ova klasa će implementirati metode getNext() i hasMore() interface-a Iiterator. Metoda hasMore() će porediti vrijednost trenutnog indeksa sa veličinom kolekcije koju sadrži kao atribut, te ako je trenutni indeks manji od veličine kolekcije vraća vrijednost true, dok u suprotnom vraća false. Metoda getNext() uvećava vrijednost trenutnog indeksa, te kao rezultat vraćati objekat tipa Član koji predstavlja sljedeći element u listi. Klasa Trening mora implementirati interface IterableCollection, i njenu metodu createIterator() unutar koja kao rezultat vraća novokreirani objekat tipa Iterator (kreiran pozivom metode new Iterator(prijavljeniClanovi, 0)). Potrebno je i da klasa Trening ima atribut tipa Iterator unutar kojeg će čuvati rezultat metode createIterator().


+obnoviClanarinu(datumObnove : Date, tipClanarine : TipClanarine) : void
+postaviPlanIshrane(planIshrane : IPlanIshrane) : void
+dajDanasnjiPlanIshrane() : string
klasa Clan
Iterator pattern
Trening
-datumOdrzavanja : DateTime
-kapacitet : int
-opis : string
-trener : Trener
-prijavljeniClanovi : List<Clan>
-tipTreninga : TipTreninga
-iterator : Iterator
+dajDatumOdrzavanja() : DateTime
+postaviDatumOdrzavanja(datumOdrzavanja : DateTime) : void
+dajKapacitet() : int
+postaviKapacitet(kapacitet : int) : void
+dajOpis() : string
+postaviOpis(opis : string) : void
+dajTrener() : Trener
+postaviTrener(trener : Trener) : void
+dajPrijavljeniClanovi() : List<Clan>
+postaviPrijavljeniClanovi(prijavljeniClanovi : List<Clan>) : void
+dajTipTreninga() : TipTreninga
+postaviTipTreninga(tipTreninga : TipTreninga) : void
+Trening(datumOdrzavanja : DateTime, kapacitet : int, opis : string, trener : Trener, prijavljeniClanovi : List<Clan>)
+prijaviClana(clan : Clan) : void
Iterator
-prijavljeniClanovi : List<Clan>
-indeks : int
+Iterator(clanovi : List<Clan>, index : int)
<<Interface>>
IIterableCollection
+createIterator() : Iterator
<<enumeration>>
TipTreninga
INDIVIDUALNI
GRUPNI
<<Interface>>
IIterator
+getNext() : Clan
+hasMore() : bool