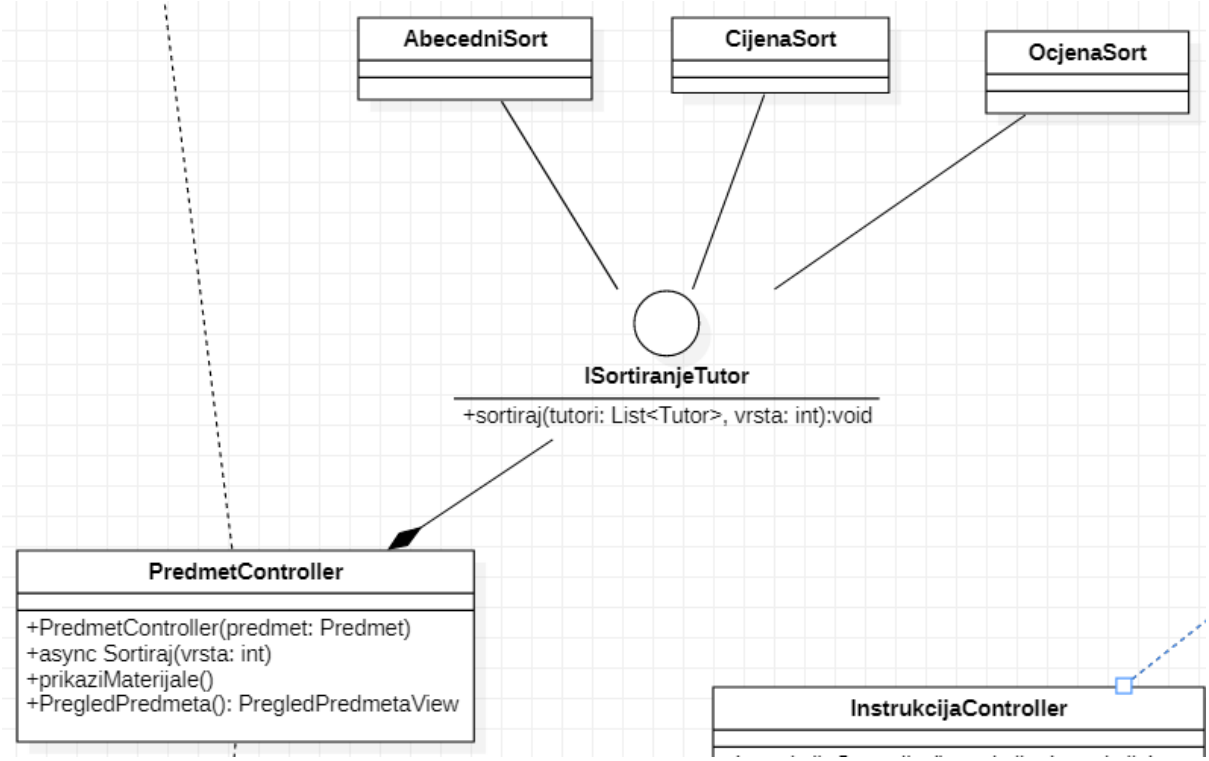
Paterni ponašanja

1. Strategy patern (primijenjen)

Strategy patern izdvaja algoritam iz matične klase i uključuje ga u posebne klase, te kako bi se omogućila brza i jednostavna promjena implementacije koja se želi koristiti u bilo kojem trenutku. Na ovaj način omogućava se i jednostavno brisanje ili dodavanje novog algoritma koji se može koristiti po želji.Podržava open-closed princip.

Koristit ćemo ovaj princip u PredmetControlleru za metodu Sortiraj(vrsta: int).

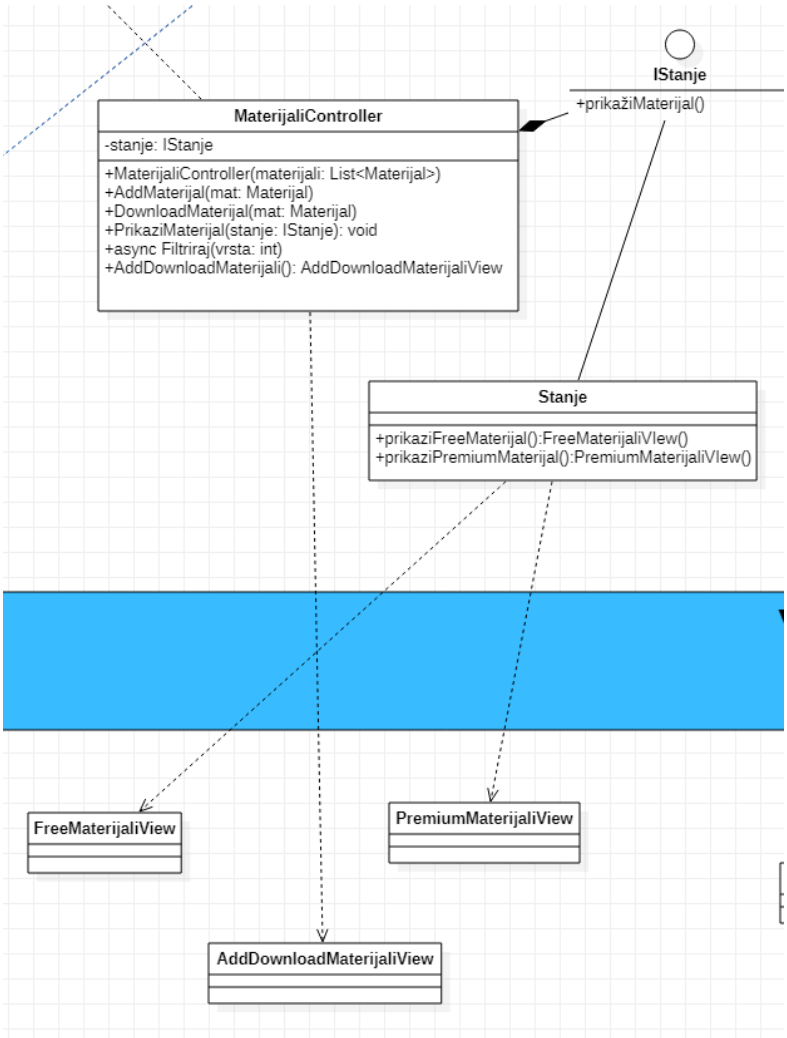


Na ovaj način vrši se refaktoring postojećeg dijagrama klase kako bi se dobio novi dijagram u kojem je ispoštovan strategy patern.
Sada je za dodavanje novog algoritma (sortiranja) dovoljno dodati novu klasu koja će naslijediti interfejs ISortiranjeTutor, a u postojećim klasama ne treba praviti nikakve izmjene.

2. State patern (primijenjen)

State patern omogućava objektu da mijenja svoja stanja, od kojih zavisi njegovo ponašanje. Sa promjenom stanja objekat se počinje ponašati kao da je promijenio klasu. Stanja se ne mijenjaju po želji klijenta, već automatski, kada se za to steknu uslovi.
State Pattern je dinamička verzija Strategy paterna.
Postiže se promjenom podklase unutar hijerarhije klasa.
Podržava open-closed princip.

U MaterijaliController imali smo dvije metode, prikaziFree() i prikaziPremium(), umjesto toga sada imamo jednu metodu prikaziMaterijal(stanje: IStanje).
To bi izveli preko interfejsa IStanje koji bi u zavisnosti od parametra prikazivao FreeMaterijaliView ili PremiumMaterijaliView



3. Template Method patern

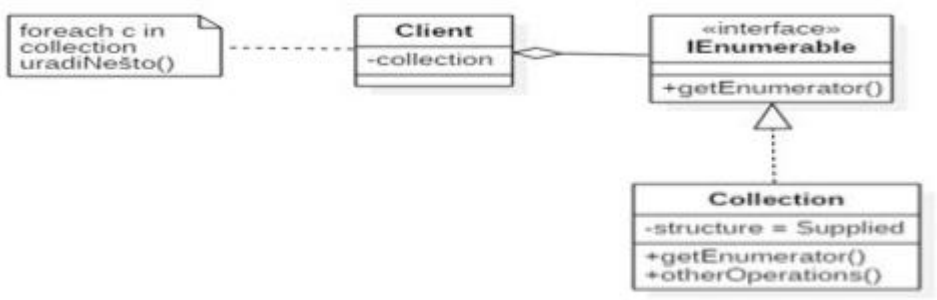
   Template method patern služi za omogućavanje izmjene ponašanja u jednom ili više dijelova. Najčešće se primjenjuje kada se za neki kompleksni algoritam uvijek trebaju izvršiti isti koraci, ali pojedinačne korake moguće je izvršiti na različite načine.

   Ovaj patern nećemo koristiti ali kada bi koristili moglo bi se staviti u sistem na sljedeći način.
   Kako imamo klasu BankovniRačun koja ima svoje atribute brojRacuna,stanjeRacuna,CSC i datumIsteka, oni bi se mogli proširiti na kreiranje posebne vrste računa, recimo bankovni račun za pravno lice koji u tom slučaju ne bi imao datumIsteka te poseban broj računa bi bio kreiran i dodatna zaštita osim CSC. Tada za kreiranje takvog računa mogli koristiti neke od metoda BankovniRačun ali bi imali neke dodatne uslove i dodatne metode.

4. Iterator patern

   Iterator patern omogućava sekvencijalni pristup elementima kolekcije bez poznavanja kako je kolekcija struktuirana. Njega vrlo lako možemo iskoristiti u našem sistemu budući da smo na mnogo mjesta predvidjeli upotrebu liste kao atributa klase.

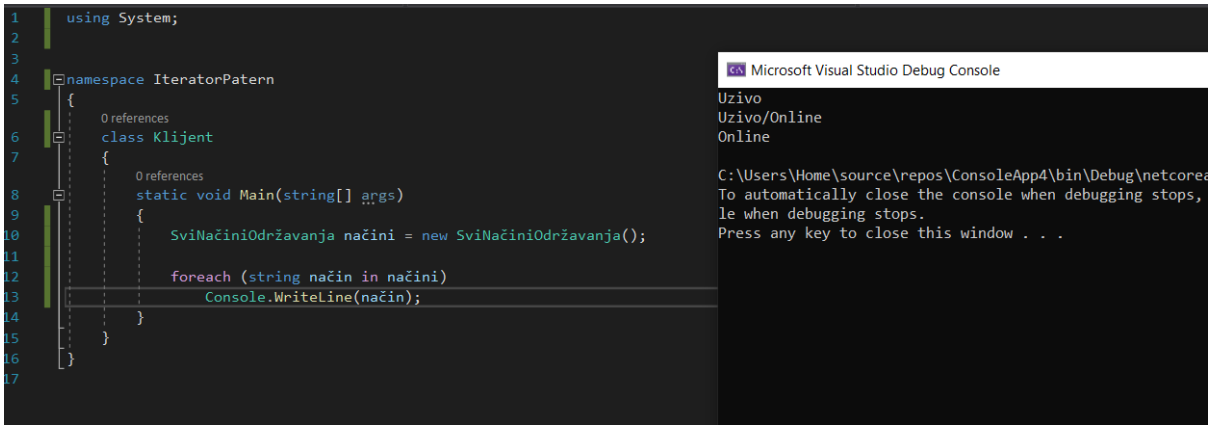
   Budući da je struktura ovog paterna veoma jednostavna, mi ćemo primjer upotrebe prikazati kroz kod u C#.



Struktura Iterator paterna

Potrebno je da klasa Collection implementira interfejs IEnumerable, odnosno akcenat je nad metodi getEnumerator(). GetEnumerator se automatski uključuje prilikom foreach iskaza.

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Text;

namespace IteratorPatern
{



        //Definisanje kolekcije i implementacija IEnumerable interfejsa
        2 references
        class SviNačiniOdržavanja : IEnumerable
        {
            string[] sviNačini = { "Uživo", "Uživo/Online", "Online" };
            1 reference
            public IEnumerator GetEnumerator()
            {
                foreach (string način in sviNačini)
                    yield return način;

            }
        }

}
```

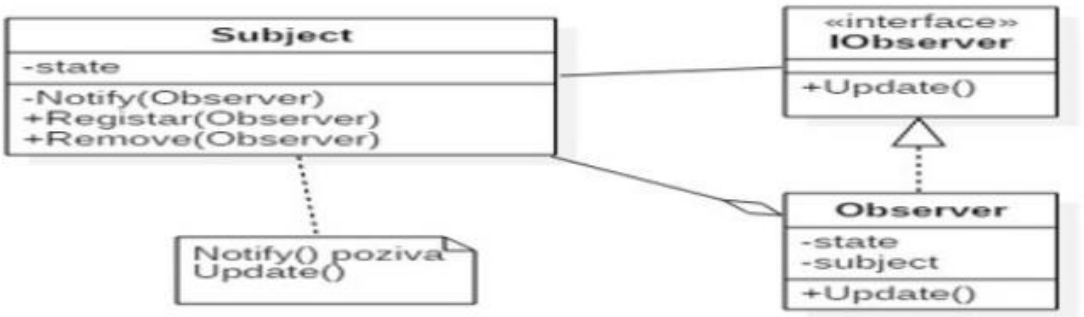


Vidimo da je na ovaj način ispunjena struktura Iterator paterna. Njega u sistem nećemo uključivati jer već imamo postignutu sličnu funkcionalnost na drugačiji način.

5. Observer patern

Uloga Observer paterna je da uspostavi relaciju između objekata tako da kada jedan objekat promijeni stanje, drugi zainteresirani objekti se obavještavaju.

Ovaj patern nećemo implementirati u sistem, ali opisat ćemo moguću upotrebu ovog paterna. Studenti posjećuju Tutore i predmete na koje su dati tutori prijavljeni. Sistem možemo proširiti na sljedeći način: Ukoliko je student zainteresiran za datog tutora i predmet koji on drži, on se može prijaviti da mu se pošalje obavijest (npr. putem email-a ili broja telefona) svaki put kada tutor promijeni cijenu na datom predmetu.



Struktura Observer paterna

U konkretnom slučaju, uloge bi bile sljedeće:

Subject - Tutor koji modificira cijene na svojim predmetima

IObserver - obavijest putem e-maila ili poruke

Observer - Student kojeg interesira promjena cijene

Update – Prijem obavijesti putem e-maila ili poruke

Notify – e-mail ili poruka od tutora svim studentima koji si zainteresirani i koji su se prijavili da dobiju obavijest

State - cijena

6. Chain of Responsibility pattern

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.

Ukoliko se sistem proširi zahtjevom da se svakom klijentu, odnosno studentu, prilikom izbora tutora omogući najoptimalniji izbor tutora po kriterijima (predmeta, cijene, lokacije, recenzija, prosjeka ocjena, načina održavanja i slično). Studentu će biti omogućen izbor prioriteta za ove kriterije. Kako bi se navedena obrada efikasno obavila, a dizajn i implementacija funkcija razumljiva potrebno je pratiti Chain of responsibility pattern.

Kreirat ćemo interfejs IUpravljač s metodom proslijedi() koja će slati zahtjeve narednoj klasi po redu za obradu po lancu koji je definiran prioritetom studenta. Ovaj interfejs nasljeđuje klasa TutorFilter. Ona ima atribute tutori (lista potencijalnih tutora) i bazniFilter koji predstavlja trenutni objekat u lancu koji je zadužen za vršenje obrade nad objektom tutor. Dodati nove klase obrađivača FiltrirajCijenu, FiltrirajLokacije i FiltrirajOcjene i sl., koje će naslijediti klasu TutorFilter te koje će posjedovati vlastite metode za vršenje različitih filtriranja liste tutora na osnovu datih informacija. One će implementirati virtuelnu metodu obradi na način da vrše pozivanje onih dijelova obrade za koje su zaduženi.

7. Mediator pattern

   Mediator pattern namijenjen je za smanjenje broja veza između objekata. Umjesto direktnog međusobnog povezivanja velikog broja objekata, objekti se povezuju sa međuobjektom medijatorom, koji je zadužen za njihovu komunikaciju. Kada neki objekt želi poslati poruku drugom objektu, on šalje poruku medijatoru, a medijator prosljeđuje tu poruku namijenjenom objektu ukoliko je isto dozvoljeno.
   Naš sistem trenutno nema potrebe za ovim paternom. Potencijalna situacija gdje bi se mogao koristiti je ukoliko bi se sistem proširio korisničkim zahtjevom da se rad i kvaliteta tutora može ocijeniti (like i dislike, 0 do 5 zvijezdica ili slično) ili/i omogući pisanje recenzija i komentara na rad tutora. U tom slučaju bilo bi neophodno izdvojiti logiku komunikacije studenta i tutora u poseban interface putem Mediator patterna. Također neophodno je da komentarisanje i ocjenjivanje bude omogućeno samo studentima koji su imali instrukcije kod tog tutora.

   Da bi sve gore navedeno postigli kreirali bi interface IRecenzijaMedijator s javnim metodama provjeriAutoraRecenzije (provjerava da li recenziju piše student koji na to ima pravo) i provjeriSadržajRecenzije (provjerava da li tekst recenzije sadrži neprimjeren ili irelevantan sadržaj). Klase koje mogu ostavljati recenzije će imati atribut tipa IRecenzijaMedijator.