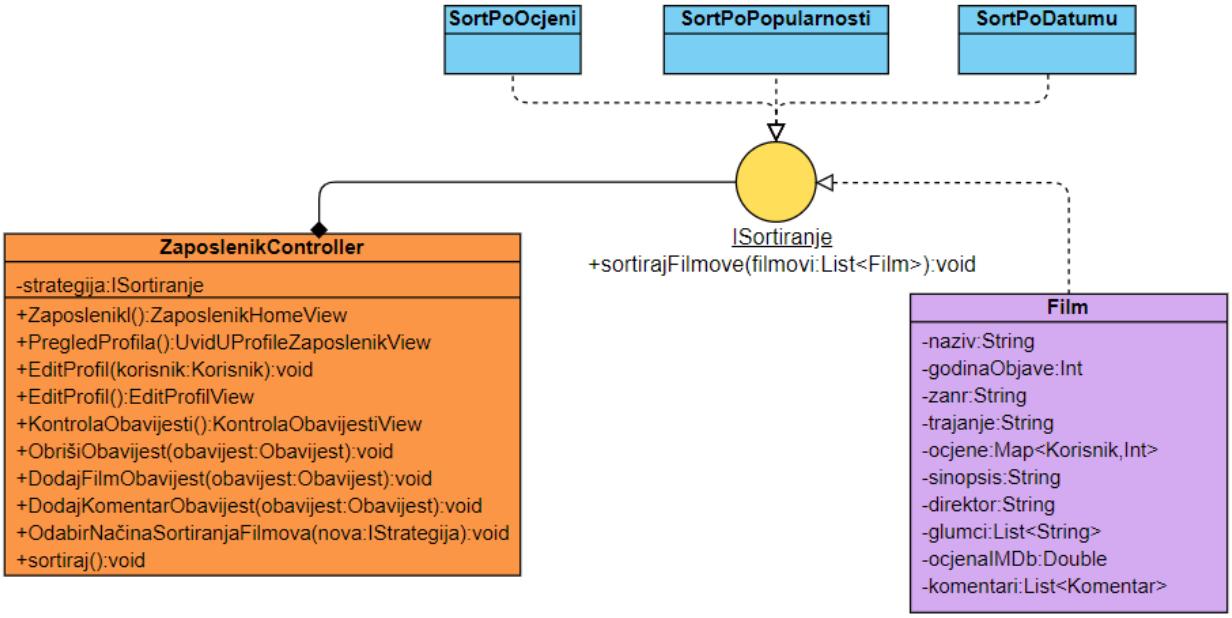
PATERNI PONAŠANJA

1.)Strategy patern

Strategy patern izdvaja algoritam iz matične klase i uključuje ga u posebne klase. Pogodan je kada postoje različiti primijenjivi algoritmi(strategije) za neki problem i omogućava klijentu izbog jedne od strategija za korištenje.
Korištenje ovog paterna nam olakšava brisanje ili dodavanje novih algoritama koji se po želji mogu koristiti.

Ovaj patern smo implementirali na način da smo uveli tri nova algoritma za sortiranje filmova, koje zaposlenik može koristit. On će birati između sortiranja po ocjeni, popularnosti i datumu.

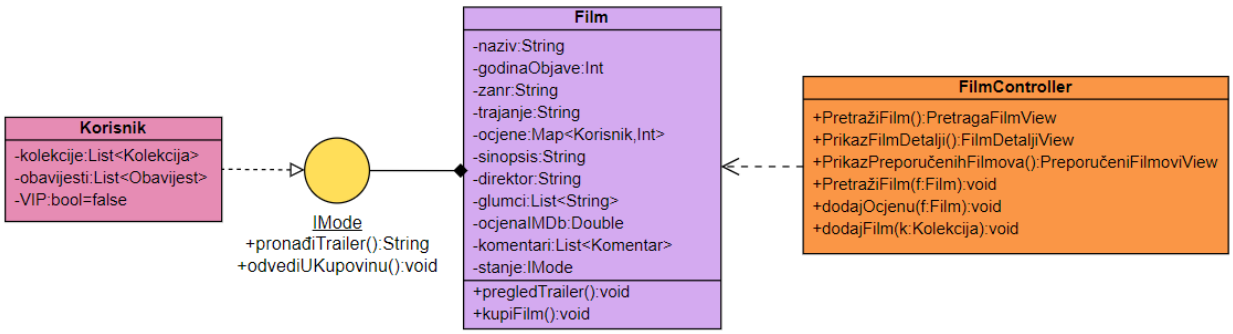


2.) State patern

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.

Ovaj patern bismo mogli iskoristiti u sistemu ukoliko uvedemo klase VIP i NonVip, zajedno sa interfejsom IStanje. Ideja za ovaj patern će biti prekomplikovana za

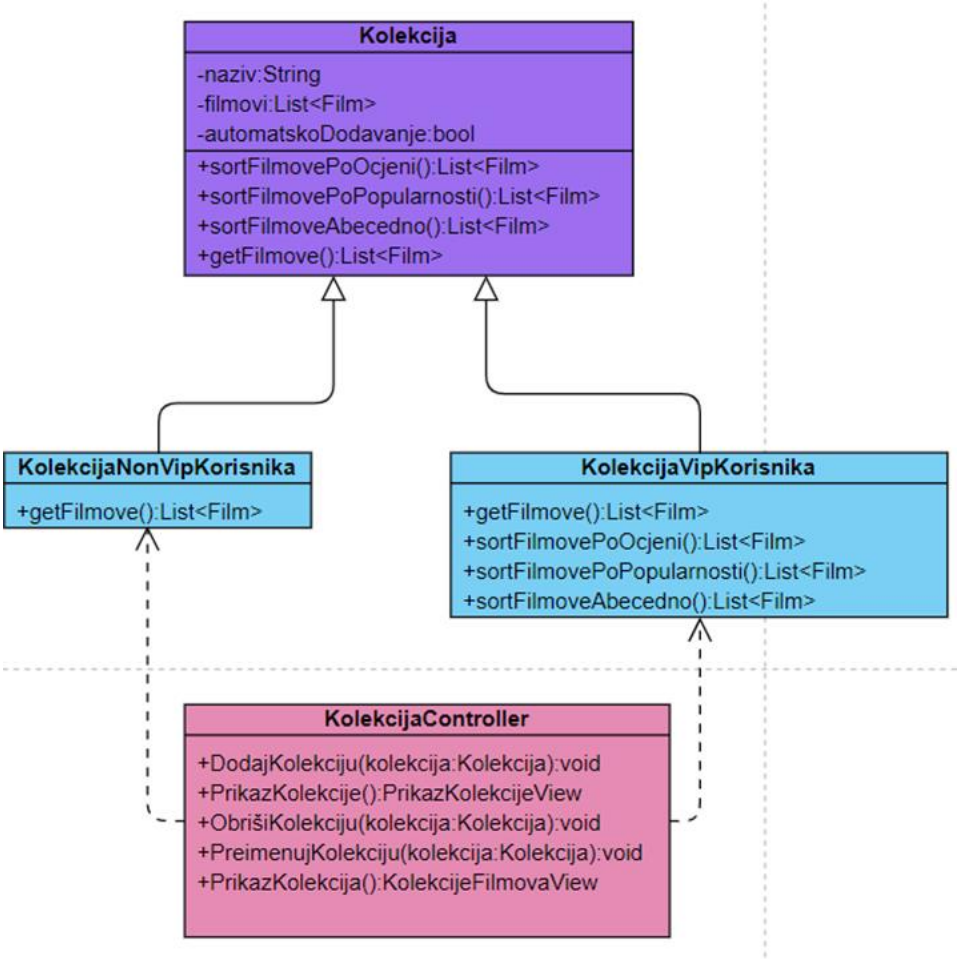
implementaciju, pa ga nećemo implementirati, već ćemo samo dati ideju. Ona je sljedeća:
Vip korisnici, prilikom otvaranja detalja o filmu, imaju mogućnosti pregleda trailer-a i kupine filma. “Obični” korisnici to ne mogu. Kako bi oni mogli oprobati ove funkcionalnosti, bez plaćanja da budu Vip, njima je omogućeno jednom godišnje u aprilu da postanu Vip besplatno. U klasi Film nalazi se atribut stanje zajedno sa dvije metode pregledTrailer i kupiFilm. Unutar ovih metoda vrši se provjera da li je trenutni mjesec april. Ukoliko jeste, ove dvije funkcionalnosti će biti omogućene za sve korisnike (instancirat će se objekat tipa Vip). Međutim, ukoliko nije mjesec april, instancirat će se NonVip klasa, ukoliko je u pitanju “običan” korisnik, a Vip ukoliko nije.



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 bismo mogli iskoristiti u sistemu na način da uvedemo dvije vrste kolekcija, u zavisnosti od toga da li je korisnik prijavljen kao VIP ili ne. Ukoliko je VIP, tada ima mogućnost sortiranja filmova u kolekcijama na željeni način, dok su filmovi kod “običnog” korisnika nesortirani.

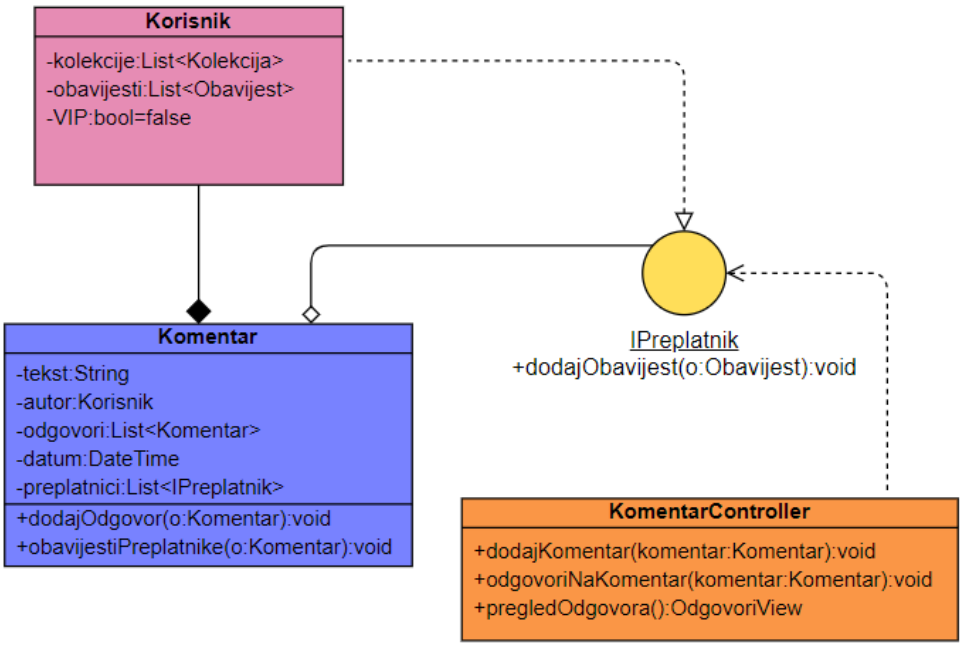


4.)Observer

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

Jedna od funkcionalnosti našeg sistema je mogućnost slanja obavijesti korisnicima. Komentar obavijest se šalje u onom trenutku kada korisnik dobije odgovor (eng. reply) na svoj komenatar. Svi korisnici će imati opciju biranja da li žele da primaju ove notifikacije. Oni koji to žele, posmatraju se kao “preplatnici” i implementiraju metodu interfejsa IPreplatnik.

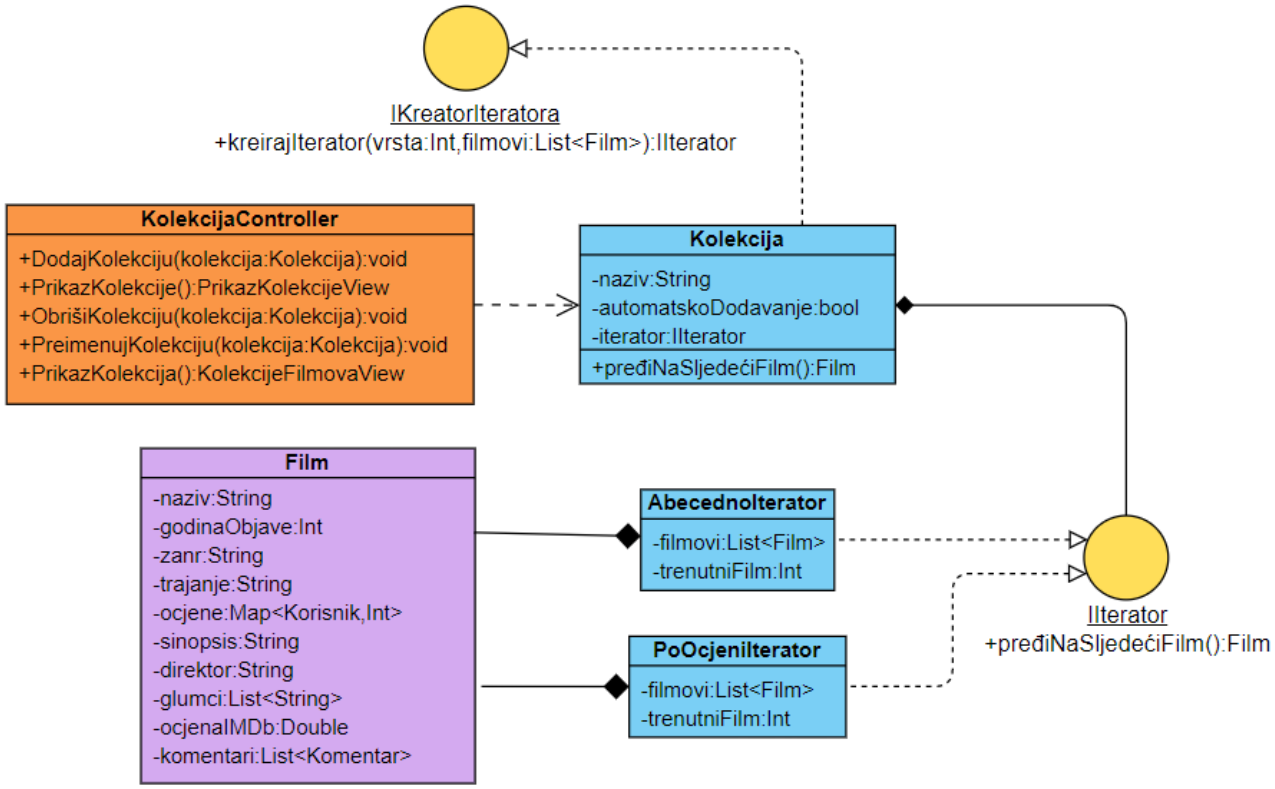
Svaki komentar će imati listu preplatnika, koja obuhvata autora komentara na koji se odgovara i sve druge korisnike koji su također ostavili reply na posmatrani komentar.



5.) Iterator

Ovaj patern omogućava sekvencijalni pristup elementima kolekcije bez poznavanja kako je kolekcija struktuirana.

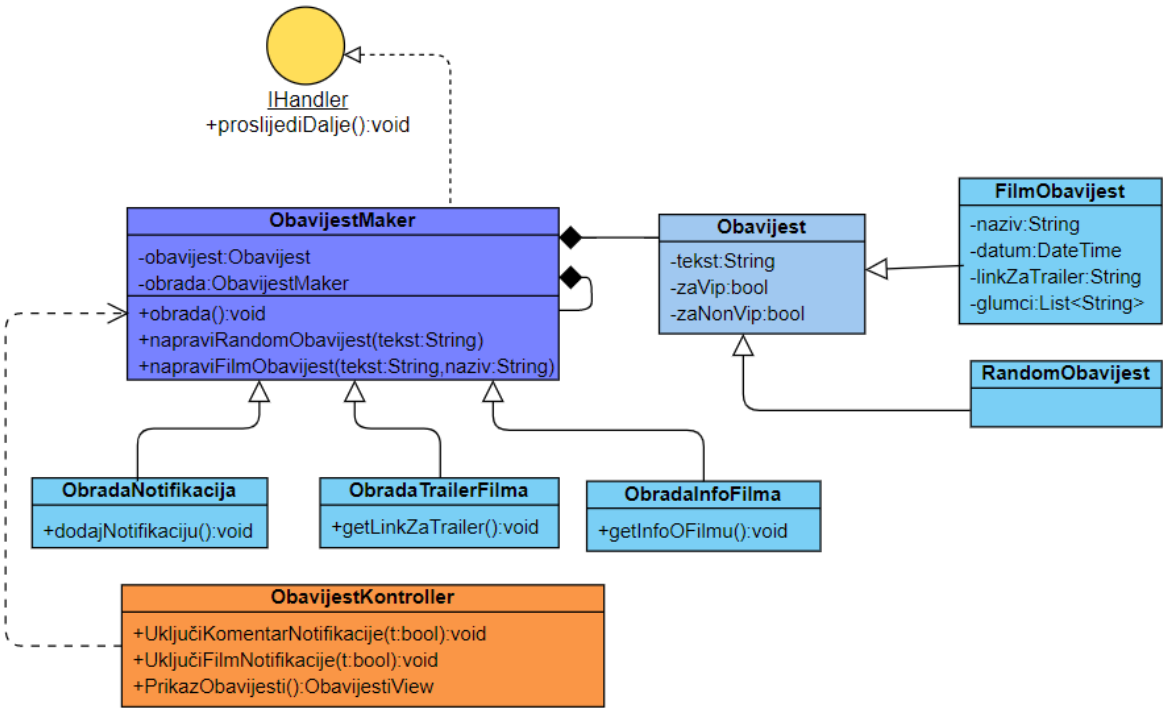
Ovaj patern nismo implementirali u našem sistemu, ali jedan od načina kako bismo to uradili je sljedeći:
Možemo uvesti opciju prolaska kroz filmove kolekcije, na osnovu abecednog poretka ili na osnovu ocjena filmova. Za ovo su nam potrebne dvije klase sa filmovima (AbecedniIterator, PoOcjeniIterator) koje implementiraju metodu pređiNaSljedećiFilm():Film.
Klasa kolekcija je struktura preko koje ćemo pristupati filmovima. Ona nasljeđuje interfejs IKreatorIterator sa metodom za kreiranje odgovarajućeg iteratora.



6.) Chain of responsibility patern

Chain of responsibility patern namijenjen je tome kako bi se jedan kompleksni proces obrade razdvojio na način da više objekata na različite načine procesiraju primljene podatke.

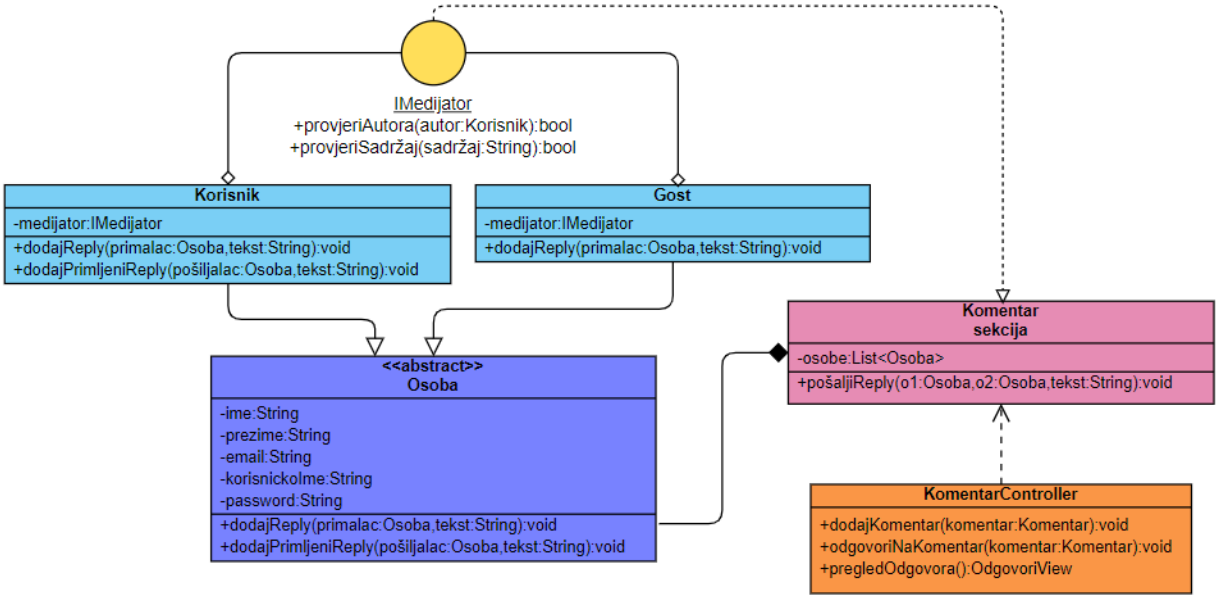
Recimo da smo u našem sistemu zaposleniku dali mogućnosti kreiranja obavijesti. U sistemu imamo tri vrste obavijesti, a to su “film” (samo za Vip), “komentar”(za sve korisnike) i “random” (za sve korisnike).
Zaposlenik može jedino kreirati “random” i “film” obavijesti. Za “random” dovoljno je da unese tekst, a notifikacije će se same postaviti (zaVip=true i zaNonVip=true). Prilikom kreiranja “film” obavijesti, unosi tekst i naziv filma, dok se link za trailer i dodatne nedostajuće informacije pronalaze na internetu i automatski postavljaju. Nakon toga pozvat će se obradaNotifikacija koja će postaviti zaVip=true && zaNonVip=false.



7.) Medijator patern

Mediator patern 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.

Za svrhu implementacije ovog paterna zamislimo da imamo još klasu Gost. On može odgovarati na komentare, ali ih ne može primati, dok korisnik može oboje. Da bi napravili ovu funkcionalnost, potreban nam je medijator koji će vršiti provjere autora (da li ta osoba ima pravo na ostavljanje odgovora na komentar) i provjere sadržaja (da li komentar ima neprimjeren sadržaj).
Naš sistem bi imao dodatnu klasu “Komentar sekcija”, preko koje šalje napisane odgovore i interfejs IMedijator sa metodama provjeriAutora i provjeriSadržaj.


IMedijator
+provjeriAutora(autor:Korisnik):bool
+provjeriSadržaj(sadržaj:String):bool
Korisnik
-medijator:IMedijator
+dodajReply(primalac:Osoba,tekst:String):void
+dodajPrimljeniReply(pošiljalac:Osoba,tekst:String):void
Gost
-medijator:IMedijator
+dodajReply(primalac:Osoba,tekst:String):void
Komentar sekcija
-osobe:List<Osoba>
+pošaljiReply(o1:Osoba,o2:Osoba,tekst:String):void
<<abstract>>
Osoba
-ime:String
-prezime:String
-email:String
-korisnickoIme:String
-password:String
+dodajReply(primalac:Osoba,tekst:String):void
+dodajPrimljeniReply(pošiljalac:Osoba,tekst:String):void
KomentarController
+dodajKomentar(komentar:Komentar):void
+odgovoriNaKomentar(komentar:Komentar):void
+pregledOdgovora():OdgovoriView