PATERNI PONAŠANJA

1. Strategy patern

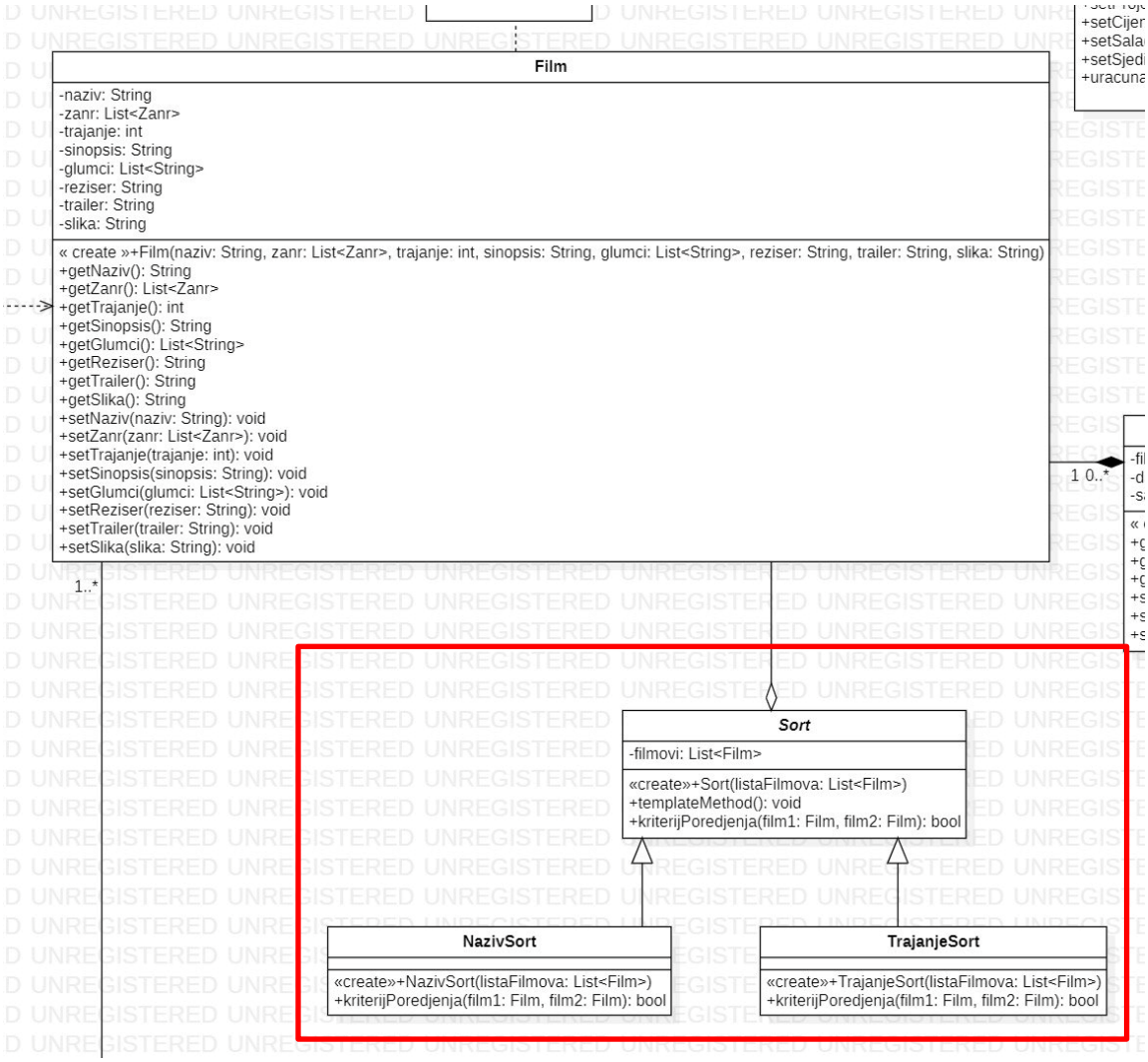
Strategy patern služi kako bi se različite implementacije istog algoritma izdvojile u različite klase, te kako bi se omogućila brza i jednostavna promjena implementacije koja se želi koristiti u bilo kojem trenutku. Na ovaj način omogućava se i jednostavno brisanje ili dodavanje novog algoritma koji se može koristiti po želji. Ovaj pattern nismo implementirali. Međutim kada bi željeli implementirati ovaj pattern to bi bilo kod plaćanja karte, tačnije kada bi omogućili drugačije načine plaćanja rezervisanja karte. Dakle dodali bi interfejs IStrategy, koji bi imao jednu metodu *placanjeKarte,* i napravili bi nove klase koje implementiraju ovaj interfejs, npr. klasa za plaćanje karticom, te klasa za plaćanje u kešu. Također bi morali napraviti posebnu klasu koja služi samo za plaćanje karte, u njoj bi imali atribut IStrategy, te metodu za promjenu načina plaćanja.

2. State patern

State patern omogućava objektu da mijenja svoja stanja, od kojih zavisi njegovo ponašanje. Sa promjenom stanja objekt se počinje ponašati kao da je promijenio klasu. Stanja se ne mijenjaju po želji klijenta, već automatski, kada se za to steknu uslovi. Nismo implementirali ovaj pattern. Imamo priliku za implementaciju ovog patterna kod rezervisanja karte, tj kada bi dodali interfejs IStanje, koji bi imao metode *rezervisiKartu*, i *otkaziRezervaciju*, te bi dodali IStanje interfejs kao atribut klase Karta. Također bi dodali i dvije klase koje bi implementirale interfejs IStanje, *Rezervisana* i *NijeRezervisana,* ove klase bi opisivale moguća stanja za pojedinu kartu. Funkcionisalo bi na način da kada je karta rezervisana, u klasi *Rezervisana* bi metodom *otkaziRezervaciju* mijenjali stanje karte u *NijeRezervisana*. Na isti način bi metodom *rezervisiKartu* u klasi *NijeRezervisana,* mijenjali stanje karte u *Rezervisana.*

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, no pojedinačne korake moguće je izvršiti na različite načine. Ovaj patern ćemo implementirati. Služi nam za sortiranje filmova po različitim parametrima. Moramo dodati apstraktnu klasu Sortiranje sa listom filmova kao privatnim atributom, ona će imati metodu *templateMethod()*, te ćemo imati i metode koje će biti implementirane u izvedenim klasama tipa *kriterijPoredjenja().* Imat ćemo izvedene klase pod nazivima NazivSort i TrajanjeSort koje će imati svoju metodu *kriterijPoredjenja().*



4. Observer patern

Observer patern služi kako bi se na jednostavan način kreirao mehanizam pretplaćivanja. Pretplatnici dobivaju obavještenja o sadržajima na koje su pretplaćeni, a za slanje obavještenja zadužena je nadležna klasa. Na ovaj način uspostavlja se relacija između klasa kako bi se mogle prilagoditi međusobnim promjenama. Ovaj patern nismo implementirali, međutim prilika za implementaciju ovog paterna bi bila kada bi imali nekog VIP korisnika, koji bi se mjesečno pretplaćivao na naš sajt. Pretplaćivanjem bi dobio obavijesti o svakom novom filmu koji je zakazan za prikazivanje. Za implementaciju nam je potreban interfejs *IObserver*, te klasa *VIPKorisnik* koja implementira taj interfejs. Interfejs će imati metodu *Update()* koja šalje obavijesti svim pretplaćenim tj VIP korisnicima. Dalje nam u klasi *CineMagic* treba lista VIP korisnika, zatim nam trebaju metode za dodavanje članova u tu listu i brisanje članova iz liste, te metoda *Notify()*. Također nam je potreban i privatni event *obavijestiVIPKorisnike()* u klasi CineMagic.

5. Iterator patern

Iterator patern namijenjen je kako bi se omogućio prolazak kroz listu elemenata bez da je neophodno poznavati implementacijske detalje strukture u kojoj se čuvaju elementi liste. Izvedba liste može biti u obliku stabla, jednostruke liste, niza i sl., no klijentu se omogućava da na jednostavan način dolazi do željenih elemenata. Osim toga, ovaj patern preporučljivo je iskoristiti kada se za iteriranje koristi kompleksna logika koja ovisi o više kriterija. Ovaj patern smo odlučili da implementiramo. Odlučili samo napraviti da pristupamo elementima kolekcije sala u klasi CineMagic preko iteratora. Da bi ovo postigli morali bi napraviti interfejs *IKreatorIterator*, koja kreira iterator, tj posjeduje metodu *createIterator(),* zatim bismo kreirali interfejs *Iterator* koji posjeduje dvije metode *hasNext()* i *getNext().* Morali bismo i kreirati klasu *SalaIterator,* koja kao atribute ima listu sala i atribut koji govori o trenutnom indeksu, ova klasa bi implementirala interfejs *Iterator.* Također bi u klasu *CineMagic* morali dodati još jedan atribut tipa *Iterator*, dok nam lista sala u klasi *CineMagic* ne bi bila potrebna.