Projekat Cinestvar – zadnji zadatak pred implementaciju (paterni ponašanja)

U nastavku se nalazi potencijalna i aktuelna primjena skoro svih dosad nabrojanih šablona ponašanja obrađenih ili spomenutih na času. Izuzetak je interpreter, koji nije dobio detaljan pregled ni na prezentaciji i u urađenim zadacima, a pošto se tiče tumačenja gramatike, ne mislimo da ima prikladnu primjenu unutar naše aplikacije.

Primjene:

1. Strategy – klasa dobije kontekst i onda bira između više algoritama koje će koristiti, algoritam A, algoritam B itd. Konkretan primjer u predavanjima je sa algoritmima sortiranja, pa smo i mi iskoristili nešto slično: za klasu koja sadrži filmsku ponudu, ukoliko imamo pretraživanje po filteru tj žanru morat ćemo sortirati filmove po tom kriteriju. Ali ukoliko nismo selektirali nijedan žanr, svi filmovi će biti prikazani. U oba slučaja će se poredati abecedno, međutim možemo odvojiti ta sortiranja kao dva različita algoritma.
2. State – objekat mijenja način ponašanja na osnovu trenutnog stanja: ovaj obrazac bi se mogao iskoristiti za ponašanje aplikacije zavisno od stanja korisnika – filmska ponuda bi se mogla ažurirati sa dodatnim opcijama i mogućnostima zavisno od toga da li je korisnik registriran ili ne, s tim će se činiti kao da je filmska ponuda bez registracije sasvim druga klasa u odnosu na filmsku ponudu sa registracijom (jer potonja u suštini jedina sadrži sve funkcionalnosti aplikacije).
3. Template Method – ovaj patern je teže izvesti jer zahtijeva malo proširenje aplikacije, ali ideja se sastoji u ovome: temp. meth. se ponaša kao Strategy, samo umjesto izbora između algoritama bira raspored kojim se koraci odvijaju. Sa ovim paternom bismo mogli razviti robusnije i preciznije pretraživanje sa filterom, gdje bismo npr. imali proces  
   izaberite žanr || izaberite verziju (2d, 3d) || izaberite dan projekcije i sl. gdje bi sve ove opcije bile ponuđene unutar svojih malih odjeljaka, i u glavnoj klasi ponude bismo imali uslugu sortiranja čije bi korake na mjestima override-ale izvedene klase za sortiranje po žanru, verziji i danu
4. Chain of Responsibility – ovaj patern delegira događaje na koje ne može reagovati drugoj klasi, druga trećoj itd. Ovo bi se moglo implementirati između klasa koje upravljaju rezervacijom, log in ekranom i sign up ekranom. Prilikom rezervacije filma imamo handler interfejs koji je spreman da klijentu usluži film, ukoliko je klijent registrirani korisnik. Ukoliko nije basehandler klasa preusmjerava na kontroler i pogled za log in, a ukoliko klijent nema ni napravljen račun onda se ponovno preusmjerava na kontroler i pogled za sign up.
5. Command je teško za implementirati jer nemamo puno elemenata koji su slični arhitekturom ali obnašaju različite funkcije, najsličnija primjena bi bila na ekranu za dodavanje filma kod Radnika, da ima mali toolbar kao aplikacije za održavanje web stranica npr. Typo3. Na takvim aplikacijama ima posebna kartica za tekst, posebna kartica za dodavanje slike, posebna kartica za dodavanje termina . . . Onda bismo mogli uvesti interfejs Kartica koji se nastavlja na klase KarticaSlika, KarticaTekst, KarticaTermin, gdje se individualno obrađuju podaci na različite načine.
6. Iterator ćemo iskoristiti za popis radnika i korisnika na ekranu za Admin profil, jer ih moramo izlistati za popis banovanih i regularnih korisnika i sl.
7. Mediator zapetljava komunikaciju između dvije klase ali omogućava zaštitu elemenata jedne klase u odnosu na drugu. S obzirom na stanje naših klasa, nemamo direktnu potrebu za primjenom, ali ukoliko bismo željeli zaštititi informacije korisnika od Admina, a da mu i dalje damo opciju da pregleda profile, između tih klasa se može implementirati medijator koji daje informacije Admin klasi, ali koji ne da Admin klasi da ih mijenja. Dodatno, ukoliko se u budućnosti promijeni kostur korisničkih klasa, način pregledanja profila se neće pokvariti.
8. Observer bi se mogao iskoristiti ovako: pošto svi korisnici pri stvaranju profila dodaju svoju email adresu, možemo napraviti opciju “Želim ažuriranja o novim ponudama” ili neki sličan tekst. Onda se ti korisnici proslijede Publisher klasi koja ih doda u pretplatnike, i pri svakom ažuriranju filmske ponude novim filmom stigne im obavijest na mail. Ova funkcionalnost nije upotrijebljena zbog upotrebe vanjske komunikacije preko maila što bi bilo previše kompleksno za dodavati ovako kasno.
9. Visitor nam nije potreban jer nemamo više vrsta filmova koji bi se mogli posjetiti, ali ukoliko želimo, možemo napraviti poseban interfejs i prikaz za filmove koji su u “posebnoj hit ponudi” (npr. ukoliko nas distributer dodatno finansijski motivira da reklamiramo film), pa onda imamo visitore za normalni film (samo prikaži film), HitAkciju (pusti neki zvuk pri otvaranju prozora), HitHorror (promijeni boju pozadine u crnu), HitRomantičnuKomediju(nacrtaj srce na nekoj nenametljivoj poziciji na ekranu). Kako bi svi ovi efekti nestali čim se zatvori film i vrati na regularnu ponudu, visitor bi bio idealan.
10. Interpreter - /
11. Memento – možemo ga implementirati sa interfejsom SnapshotHandler i klasom Snapshot s kojim bi radnik mijenjao informacije o filmu, tako da svaki put kad otvori film da uređuje, napravi se snapshot otvorenog filma, tako da se mogu poništiti postojeće izmjene, ili ukoliko radnik pritisne dugme “Spasi izmjene”, da to postane novi snapshot za taj film.