Nume: Calu Andrei-Daniel

Grupa: -322 CC-

Grad de dificultate: 7/10 – A fost mai greu sa inteleg limbajul Java (+Pattern-uri, Interfete, Exceptii), care nu e foarte dificil dar nu mi-am dat asa mult interesul pana la tema.

Timp alocat: Am inceput tema in jur de 15 decembrie, lucrand ocazional, insa am lucrat intens in ultima saptamana. (Aprox. 15 zile)

Mod de implementare:

Am inceput prin a crea interfetele si enumerarile. In continuare, am inceput sa creez clasele, lasand clasa IMDB la final, astfel:

Clase:

- Production:

Am creat variabilele + constructor conform cerintei, iar pentru rating-ul final am facut o functie ce calculeaza media aritmetica a notelor prezente in clasele Rating continute de productie, neffind inclus in constructor. In plus, am implementat comparatorul, am creat settere si gettere si am facut o functie updateOverallRating pentru a recalcula rating-ul final la adaugarea unei noi recenzii. Am definit functia abstracta displayInfo() ce va fii implementata in clasele mostenitoare Movie si Sereis.

- Movie:

Exitnde clasa Production si adauga 2 campuri, lungimea (in ore) si anul aparitiei productiei. Am implementat displayInfo() care afiseaza implicitit datele filmului, in afara de rating-uri pentru care am afisat separate campurile clasei Rating. (

+ Settere)

- Episode:

Clasa cu 2 campuri, numele si lungimea episodului, necesara clasei Series, pentru care am realizat un constructor si getters.

- Series:

Mosteneste clasa Production si adauga 3 campuri, anul aparitiei, numarul de sezoane si un LinkedHashMap (deoarece am vrut adaugarea ordonata a sezoanelor) ce contine pentru fiecare sezon(o cheie de tip String) o lista de episode te tip List<Episode>. Am implementat afisarea displayInfo() cu care am avut probleme ulterior deoarece m-a facut sa realizez ca initial, nu am parsat bine din fisierul Json harta de sezoane si episoade. Intr-un final, afisez implicit campurile in afara de Rating-uri si harta de sezoane, in care iterez si afisez campurile separate.

- RequestHolder

O clasa statica ce mentinte o lista separata de Request-uri destinate doar Admin-ilor, o metoda de adaugare si una de eliminare a request-urilor din lista. In plus, o metoda displayRequests() ce afiseaza campurile unui Request in functie de tipul acestuia. (Tipurile ACTOR_ISSUE si MOVIE_ISSUE au un camp additional).

- Request:

Am construit campurile si constructorul conform cerintei: Pentru user-ul menit sa rezolve cererea, am creat un switch ce verifica tipul cererii. Pentru ACTOR_ISSUE si MOVIE_ISSUE caut in lista de conturi, pentru fiecare User, in lista caruia de contributii se regaseste actorul/productia ceruta pentru revizuire, dupa care ii atribui User-ului respectiv cererea.

- Rating:

Clasa cu 3 campuri si un constructor, conform cerintei.

- User:

Am creat campurile si constructorul, insa pe care nu il voi folosi, creand ulterior un **Factory Pattern**. Am definit 4 metode pe care le voi implementa in clasele mostenitoare (User fiind o clasa abstracta) si am creat subclasa Information:

- Information:

  Pentru clasa Information am creat campurile si un constructor (+ o noua clasa Credentials pentru email si parola), insa, pentru instantare am folosit **Builder Pattern.**

De asemenea, am implementat functia update() menita sa trimita notificarile userilor prin **Observer Pattern**.

- Regular:

Mosteneste clasa User. Am implementat metodele abstracte, pentru logout() alegand sa returnez un intreg, 1/0, in functie de care voi actiona in meniu pentru inchiderea aplicatiei sau intoarcerea la meniul de logare. De asemenea, implementeaza interfata RequestManager. Pentru createRequest() am avut grija sa implementez **ObserverPattern** prin care notific userii in functie de tipul cererii. Daca este o cerere menita unui singur contributor, acesta va primii o notificare corespunzatoare, iar, daca este menita unui Admin, toti userii cu acest rol vor primii notificarea. Pentru removeRequest(), am grija sa elimin cererea atat din lista totala de request-uri, cat si din RequestHolder, in functie de caz. Pentru addRating(), am impementat inca o data **ObserverPattern**, notificand atat userii care au mai oferit o recenzie aceleasi productii, cat si Staff-ului care a adaugat-o. De asemenea, prin **ExperienceStrategy**, user-ul va primii experienta pentru adaugarea recenziei. removeRating() doar elimina un rating din lista productiei.

- Staff:

O clasa mai complexa, parinte pentru Contributor si Admin insa mosteneste User. La adaugarea unei productii sau actor in sistem, user-ul primeste experienta conform **ExperienceStrategy**. La eliminarea unei productii/actor am tratat 2 cazuri: Un contributor poate elimina doar ce a adaugat el, insa un Admin poate elimina orice. La actualizarea unei productii sau a unui actor, am ales ca Staff-ul sa poata reintroduce, pe rand, toate campurile productiei dorite. La rezolvarea unei cereri, prin **ObserverPattern** trimit o notificare catre user-ul ce a creat-o, elimin notificarea user-ului care a rezolvat-o, iar prin **ExperienceStrategy** atribui experienta user-ului ce a creat-o. De asemenea, sterg cererea din sistem.

- Contributor:

Poate crea o cerere, unde am implementat similar **ObserverPattern**, poate sterge o cerere si am implementat rapid metodele de adaugare, stergere in favorite si logout()-ul, similar cu cel anterior.

- Admin

Nu are o metoda noua, intrucat am implementat metoda de adaugare/stergere a unui user in clasa IMDB pentru a o apela direct in meniu.

- Credentials

O clasa cu 2 campuri si un constructor, email-ul si parola utilizatorului.

- Actor

Clasa contine campurile si constructorul conform cerintei, insa pentru productiile in care a jucat am creat o clasa NameType deoarece contine numele si tipul productiei(Movie/Series). Contine gettere, settere si o metoda de afisare displayInfo(), plus metoda de comparare.

- IMDB

Am creat un **Singleton Pattern** prin metoda getInstance() si am initializat in constructorul privat cele 4 mari liste: allAccounts, allActors, allRequests, allProductions, ele fiind declarate public in afara constructorului. Am creat o metoda de parsare a fisierelor Json parseAll(), folosind librarie JsonSimple: Am ales sa parses initial actorii si productiile, pentru care am urmat cam acelasi sablon, deoarece aveam nevoie de acestea pentru a crea preferintele si contributiile unui cont. Totusi, am intampinat probleme la parsarea hartii de sezoane-episoade pentru seriale, deoarece, initial, am incercat sa o realizez intr-un rand. Ulterior am observat ca trebuie mai multa munca, sa creez pe rand episoadele dupa care sa le adaug, alatur de sezon, in harta. La accounts am intampinat din nou probleme, deoarece, initial am parsat preferences si contributions ca String-uri, dupa care m-am intors si am realizat, pentru fiecare, un TreeSet in functie de un comparator pe 4 cazuri: prod-prod, actor-actor, prod-actor, actor-prod. Dupa asta, am parsat o lista de nume pentru productii, si una pentru actori, dupa care am cautat in actorii si productiile parsate anterior pentru a le adauga drept obiecte in SortedSet-uri. In final, am creat User-ul prin **FactoryPattern**. Pentru Request-uri, pe langa parsare, am implementat din nou **ObserverPattern** pentru a trimite notificarile initiale userilor de tip Staff ce trebuie sa rezolve cererile curente. Pentru fiecare parsare am

creat si 4 metode separate, checkActors/Accounts/Requests/Productions() ce afiseaza elementele celor 4 mari liste.

Functia run(): Initial, am apelat metoda de parsare parseAll(), dupa care am inceput sa creez meniul: Am folosit 5 while-uri, dintre care 3 sunt apelate, in functie de tipul user-ului:

- unul pentru intreg programul, ce depinde de o variabila running (val 1/0) ce devine 1 cand utilizatorul decide sa inchida programul in metoda de logout(). Daca acesta decide sa se logheze in alt cont, variabile ramane 0 iar programul se reia de la caseta de logare.
- unul pentru logare, ce depinde de un boolean found, ce devine true cand email-ul si parola sunt gasite. Daca acesta ramane false, logarea se reia.
- cate unul pentru fiecare tip de utilizator, ce depinde de o variabila menu, care devine 1 in momentul logout-ului, oprind afisarea meniului.

In functie de tipul utilizatorului, afisez cate un meniu corespunzator. Pentru fiecare caz, am creat o metoda separata pe care o apelez, dupa care se asteapta un ‘Enter’ de la tastatura pentru a reafisa meniul. (Unele au fost create mai general, pentru a putea fi folosite atat de Regular, cat si Contributor, Admin) (Pentru afisarea productiilor si a actorilor am creat deja checkProductions, checkActors).

Metodele ajutatoare:

- displayNotif() -> Afiseaza notificarile utilzatorului (!Daca acesta are minim una)
- searchActorMovieSeries()-> In functie de caz, cauta obiectul in lista corespunzatoare deja existenta. In cazul in care nu exista, afiseaza un mesaj corespunzator.
- addDeleteFavorite()-> Adauga sau sterge la favorite actorul/productia dorinta, dupa care afiseaza noua lista de favorite. Daca nu este gasit obiectul, se va afisa un mesaj corespunzator.
- addDeleteRequest()-> La adaugare, se alege tipul cererii dupa care se introduc campurile necesare crearii, ulterior creand cererea prin metoda createRequest() din Regular/Contributor. In mod similar, la stergere se alege tipul, se introduc campurile dupa care se elimina cererea prin removeRequest();
- addDeleteRating()-> La adaugare, se introduce pe rand cele 3 campuri dupa care se adauga rating-ul prin metoda addRating(), insa, pentru rating-ul propriu-zis (definit de mine oneRating) am pus 2 conditii: sa fie numere

intregi intre 1 si 10 si sa fie numere. Altfel, sunt afisate mesaje corespunzatoare prin exceptii. La stergere, se introduce titplu si comment-ul dupa care rating-ul este cautat s eliminat prin metoda removeRating();

- addDeleteProductionActor()-> Probabil cea mai lunga metoda. Pentru adaugare, se alege daca este o productie sau un actor. In cazul unei producii, se introduc campurile commune filmelor si serialelor, din clasa Production. Pentru campurile ce contin liste, las utilizatorul sa introduca pe rand, iar, in final sa tasteze "end" cand a terminat, dupa care procedeaza la fel pentru urmatoarele. Dupa campurile commune, alege dintre film sau serial. Pentru film, mai trebuie introdusi 2 intregi, iar, pentru serial, pe langa 2 inregi, trebuie sa introduca harta de sezoane-episoade. Am procedat similar, pentru un sezon se introduce o lista de episoade, oprind-o prin "end", iar sezoanele vor fii oprite asemenea. La adaugarea unui actor se introduc cele 3 campuri pe rand, al doilea fiind tot o lista in care creez clase NameType, iar, pentru stergere, doar caut titlul/numele si elimin actorul prin metoda din clasa Staff removeProductionSystem()/removeActorSystem();
- viewSolveRequestContributor()-> Initial, daca are, afisez request-urile acestui user, dupa care ii ofer optiunea de a introduce descrierea cererii dorite. Acesta poate alege daca o sterge sau o rezolva. La rezolvare, se apeleaza metoda solveRequest() din staff prin care se ofera experienta si manage-uieste notificarile, eliminand totodata cererea, iar, la stergere, doar se elimina cererea si se sterge notificarea user-ului menit sa o rezolve.
- viewSolveRequestAdmin()-> Similar cu metoda anterioara, insa, initial, se afiseaza cererile din RequestHolder, dupa care, la rezolvarea sau stergerea unei cereri DELETE_ACCOUNT sau OTHER, se elimina notificarea tuturor adminilor.
- updateProductionActor()-> Pentru aceasta metoda, se alege daca obiectul este o productie sau un actor, se cauta in lista in functie de titlu, dupa care se apeleaza metoda din Staff updateProduction() sau updateActor() care functioneaza similar cu metoda de adaugare, insa, in loc sa creeze un nou obiect, seteaza campurile obiectului deja existent, pe rand, de la tastatura.
- addDeleteUserSystem()-> La adaugarea unui user, am citit initial campurile din Information, la care am avut grija sa folosesc **InformationIncompleteException** pentru a nu crea un astfel de camp fara emal, parola sau nume. Am instantat Information prin **Builder Pattern** si am continuat sa citestc urmatoarele campuri. Pentru notifications am initializat un new ArrayList(), neavand notificari in momentul crearii contului. De

asemenea, preferences si contributions au fost doar initializat ca TreeSet ce depind de un comparator, precum la parsare. Am alcatuit user-ul prin **Factory Pattern** si l-am adaugat in lista mare de conturi. La stergere, caut user-ul in functie de username in lista, iar, daca este gasit, va fii sters. Totusi, trebuie sa respecete conditia de a nu fi un admin, deoarece acest tip de conturi nu poate fi eliminat.

Exceptions:

- InvalidCommandException-> Trateaza introducerea unei date necorespunzatoare la citirea unui camp. Am folosit-o pe parcursul crearii meniului si metodelor necesare.
- InformationIncompleteException-> Am creat aceasta clasa pentru a trata exceptia: In momentul in care, la crearea unui user de catre un Admin, unul din campurile email/password/name ramane gol, adaugarea este intrerupta si se revine la meniul de optiuni.

Patterns:

- Singleton Pattern-> L-am folosit la instantarea clasei principale, IMDB. Am declarant o variabila statica de tip IMDB, instance, care, initial este instantata prin constructor (ce initializeaza cele 4 mari liste), dupa care, daca aceasta deja a fost instantata, o sa fie returnata de metoda pentru a o reutiliza.
- Builder Pattern-> A fost folosit in cadrul construirii unui obiect de tip Information. Am creat o clasa InformationBuilder cu 6 metode de set pentru cele 6 campuri, plus o metoda care construieste obiectul de tip Information, public User.Information build(); Au fost folosite atat la parsare, cat si la crearea unui nou User.
- Factory Pattern-> Am creat o clasa UserFactory ce contine un constructor static User(…){}, care, in functie de tipul de utilizator primit in argumente, instanteaza un Regular/Contributor/Admin, utilizand campurile specifice lor. Am folosit UserFactory atat la parsarea conturilor, cat si la crearea unui nou User.
- Observer Pattern-> Avem 2 intefete: Observer, cu metoda update(), implementata in User pentru a adauga notificarile la momentul potrivit; Subject, cu 3 metode pentru gestionarea pattern-ului, adaugare, eliminare si notificare a Observer-ului. Am creat clasa NotificationManager ce implementeaza metodele din Subject. Am folosit si pentru aceasta un

SingletonPattern, pentru a folosi mereu acelasi obiect NotificationManager, deoarece nu vreau sa pierd Observerii deja adaugati in acesta. In continuare, am folosit Observer Pattern in situatiile mentionate anterior, pentru adaugare, notificare si stergere a observatorilor.

- Strategy Pattern-> Am creat interfata ExperienceStrategy cu o metoda calculateExperience(), implementata in 3 clase, corespunzatoare celor 3 cazuri pentru care un user poate primi experienta: ProductionAddedStrategy: Un staff primeste 5 exp la adaugarea unei productii sau a unui actor in sistem (la adaugarea unui actor am folosit aceeasi clasa pentru a calcula experienta); RequestStrategy: La rezolvarea unui request de catre un solvingUser, requestUser-ul primeste 5 exp; ReviewStrategy: La adaugarea unei recenzii pentru o productie, user-ul primeste 1 exp. Dupa acestea, am creat clasa UserStrategy, instantat printr-un constructor ce seteaza tipul de experienta prin una din cele 3 clase create, dupa care apeleaza metoda de calcul a acelei clase, returnand experienta curenta a user-ului + experienta primita in urma actiunii. Aceasta a fost utilizata in cazurile descries in cerinta, dupa cum am mentionat pe parcursul documentului.