Nume: Binca Andreea-Cristina

Grupa: 322CC

Grad de dificultate: basic, doar ce este strict necesar

Timp alocat: aproximativ 4 saptamani de lucrat intermitent

Mediu folosit: Eclipse (JavaSE-14)

Modul de implementare:

Clasa Application:

Am creat aceasta clasa pe principiul Singleton Pattern, pentru a ma folosi de instanta aplicatiei in clasele de test/interfata grafica. Clasa contine o lista de companii (numita companies) si una de useri (users), alaturi de metodele cerute de get, add, remove, implementate simplu.

Clasa Consumer:

Aceasta clasa este una abstracta, ce contine un camp pentru resume (resume) si o lista pentru reteaua sociala (socials). Metodele de add/remove, meanGPA sunt implementate simplu.

Functia getDegreeInFriendship reprezinta o parcurgere in latime a retelei sociale, folosind 3 liste – una ce retine persoanele deja vizitate (visited), pentru a nu parcurge din nou lista lor de prieteni, una pe post de coada pentru persoane (queue), iar cealalta coada pentru gradele de prietenie cu persoana cautata (degree). Algoritmul este un bfs care foloseste queue si degree in paralel. Gradul final este retinut in variabila degrees, ce este returnata. Valoarea 0 reprezinta faptul ca persoana cautata nu se afla in reteaua sociala, 1 ca este prieten direct, 2 prieten al prietenului s.a.m.d.

Functia getGraduationYear cauta in toate elementele de Education cea mai apropiata data de finalizare, care nu este liceu. Valoarea returnata poate fi anul in care persoana a terminat licenta/master, sau 0, atunci cand are doar liceul terminat.

Functia yearsOfExp am adaugat-o pentru a usura comparatia necesara pentru constrangerile in momentul aplicarii la un job. Aceasta functie intoarce numarul total de ani de experienta, aproximati (daca pe langa numarul de ani are si peste 3 luni de experienta, se mai adauga inca un an).

Clasa Resume:

Aceasta clasa este una statica si este interna clasei Consumer, si este construita pe principiul Builder Pattern. Aceasta contine un camp information (info), o lista de education (edu) si o lista de experience (exp). Aceasta contine insasi clasa ResumeBuilder, care este folosita pentru a adauga date in campuri, si care contine functia de build, prin care este construit resume-ul, care arunca o exceptie ResumeIncompleteException (clasa pe care am implementat-o in ResumeIncompleteException.java) atunci cand nu exista informatii sau cel putin un entry in lista de educatie. De asemenea, in clasa Resume am suprascris si functia toString, pentru a afisa frumos continutul unui resume.

Clasa Information:

Aceasta clasa contine toate datele necesare din cerinta, folosind principiul incapsularii: fiecare camp avand propriul getter si setter, iar cele 2 campuri de tip lista pentru limba si nivele acestora folosesc o singura functie addNewLang, astfel incat atunci cand se adauga o noua limba, aceasta sa aiba si nivelul ei. De asemenea, este suprascrisa functia toString, pentru afisare frumoasa.

Clasa Education:

Aceasta clasa contine 2 campuri LocalDate pentru a retine datele de inceput si de sfarsit pentru fiecare ciclu de educatie, campuri de tip String pentru numele institutiei si nivelul de educatie, si un camp double pentru media de absolvire.

In constructorul acestei clase se verifica ca cele 2 date sa fie in ordine cronologica (startdate inaintea lui enddate), altfel arunca o exceptie de tip InvalidDatesException, tip de exceptie pe care l-am implementat in InvalidDatesException.java.

Clasa implementeaza interfata Comparable, asa ca este implementata functia de compareTo, care compara 2 obiecte de tip education asa cum este descris in cerinta. Este suprascrisa si functia de toString pentru afisare frumoasa.

Clasa Experience:

La fel ca si clasa Education, are 2 campuri pentru memorarea datelor, si 3 campuri de String pentru pozitie, companie si departament. Tot la fel in constructor va arunca exceptie de InvalidDatesException daca cele 2 date nu sunt in ordine cronologica. Si aceasta clasa implementeaza Comparable, iar functia compareTo compara 2 obiecte de tip Experience ca in cerinta. Are functia toString pentru afisare frumoasa.

Clasa User:

Aceasta clasa extinde Consumer si implementeaza interfata de Observer din Observer Pattern. Contine o lista pentru companiile de care este interesat userul (interested), si un String pentru notificari (notifs, parte a pattern-ului).

Functia convert intoare un nou employee cu resume-ul si reteaua sociala a userului.

Functia getTotalScore intoarce valoarea data de inmultirea anilor de experienta (aflati prin functia creata de mine mai sus) si adunarea cu media.

Functia addInterestedComp am creat-o pentru a adauga companii in lista interested, dar si pentru a adauga userul in lista de observeri a companiilor respective.

Functia update face pattern din Observer Pattern si actualizeaza stringul de notificari atunci cand apar noi notificari.

Clasa Employee:

Acesta clasa extinde Consumer si are inca 2 campuri pentru numele companiei la care lucreaza (compname) si un camp pentru salariu (salary). Clasa contine doar 2 constructori, cel de-al doilea fiind folosit la convertirea dintr-un user in employee. Deoarece nu am folosit principiul incapsularii, am ales sa nu mai adaug functii pentru a schimba valorile din compname si salary, avand in vedere ca pot fi folosite direct.

Clasa Recruiter:

Aceasta clasa extinde Employee si contine in plus un camp pentru rating.

Functia evaluate este cea prin care recruiter-ul evalueaza un user pentru un job la care acesta a aplicat. La fiecare evaluare facuta, ratingul recruiterului creste, iar evaluarea se face in modul urmator: se calculeaza un scor bazat pe rating si scorul userului, apoi se creeaza un request cu campurile – job-ul respectivul user, recruiterul curent, scor – care se aduga in lista de request-uri a managerului din compania respectiva. Functia intoarce de asemenea scorul obtinut.

Clasa Manager:

Aceasta clasa extinde Employee si contine in plus lista de request-uri.

Functia process este cea prin care se fac angajari. Ea proceseaza toate request-uri pentru un job, folosind o lista in care se retin request-urile pentru jobul dat ca paramentru (applicants), si o lista ce retine scorurile respectivelor request-uri (scores), dar si o variabila ce retine numarul de locuri disponibile (noPositions). Cele doua liste se parcurg in paralel, pana cand se termina lista de applicants sau pana cand nu mai sunt locuri libere. De fiecare data se alege aplicantul cu scorul cel mai mare, apoi daca mai exista in lista de useri, adica daca nu a fost deja angajat, se converteste in employee, apoi se adauga in lista de employee din departamentul potrivit, dupa care este scos din lista de applicants si score, dar si din lista de request-uri a managerului, dar si din lista de observeri a companiilor de care era interesat, deoarece fiind angajat, acelea dispar din interese. De asemnea, se micsoreaza numarul pozitiilor libere. Daca cumva a fost angajat intre timp, acesta va fi scos din lista de applicants, scores si request-uri, nemaicontand. Daca dupa terminarea locurilor au ramas request-uri, acestea vor fi sterse, iar daca au ramas locuri libere, acestea se vor face automat 0, la sfarsit jobul fiind inchis, iar observerii fiind notificati.

Clasa Job:

Aceasta clasa contine mai multe campuri String pentru numele jobului, companiei si departamentului, o variabila boolean care retine daca job-ul e deschis sau nu (open), 3 campuri Constraint pentru constrangerile jobului – anul de absolvire (graduationyear), numarul de ani de experienta (experienceyears) si pentru constrangerea de medie (meancons). Mai contine si numarul de locuri disponibile pentru aplicare, alaturi de salariu.

Functiile de set/add/remove sunt implementate ca in cerinta.

Functia apply este cea prin care un user aplica la un job. Aplicarea poate fi facuta atunci cand jobul e deschis, iar user intalneste constrangerile, apoi se selecteaza un recruiter folosind functia getRecruiter, dupa care se apeleaza functia de evaluare din acel recruiter.

Verificarea constrangerilor este facuta cu functia meetsRequirements, care foloseste niste variabile auxiliare, pentru a trata si cazurile in care unul dintre parametri este null (de exemplu, daca jobul respectiv nu necesita diploma de licenta, limita de sus – upperlimit – a constrangerii de an de absolvire este null, iar pentru a fi posibila o comparatie consider ca in loc de null avem anul 2020). Functia intoarce true doar daca userul se incadreaza in toate constrangerile, altfel returneaza false.

Clasa Constraint:

Aceasta clasa contine campurile upperlimit si lowerlimit. Ea este parametrizabila, deci poate fi folosita pentru toate tipurile de constrangeri.

Clasa Company:

Acesta clasa implementeaza interfata de Subject a pattern-ului Observer si contine numele companiei si a managerului, o lista de departamente (departments), una de recruiteri (recruiters), si una de observeri (observers).

Functiile de add/remove/move/get sunt implementate simplu, ca in cerinta.

Functia getRecruiter este cea care alege Recruiter-ul potrivit user-ului dat ca parametru. Aceasta itereaza prin lista de recruiteri a companiei, alegandu-l pe cel mai indepartat, folosind functia getDegreeInFriendship pentru a afla gradul de prietenie. Daca nu gaseste recruiter, il alege pe cel cu rating-ul cel mai mare.

Functiile addObserver, removeObserver si notifyAllObservers fac parte din implementarea pattern-ului Observer.

Clasa Department:

Aceasta clasa este una abstracta si contine o lista de angajati (employees) si una pentru job-urile disponibile (availablejobs). Functiile add/remove/get sunt implementate ca in cerinta, cu aditia faptului ca in functia care adauga un nou job observerii acelei companii for fi notificati, iar functia abstracta este implementata in fiecare departament.

Clasele IT, Management, Marketing, Finance:

Aceste clase implementeaza functia getTotalSalaryBudget in functie de taxele din fiecare departament.

Pentru a instantia aceste clase am folosit Factory Pattern, implementat in clasa DeptFactory, care returneaza o noua instanta de departament in functie de un id dat.

Clasa Test:

Aceasta clasa este formata din 2 parti:

Functia read(JSONObject obj) prin care se face citirea campurilor unui resume dat ca parametru de tip obiect json. Din acesta se preiau informatiile eticheta cu eticheta, construindu-se un resume, ce este returnat de functie.

Functia startApplication() prin intermediul careia se foloseste aplicatia. Am creat de mana cele 2 companii, adaugand departamentele, cu toate datele lor. Apoi am parcurs fisierul de intrare json folosind functiile din Parser (pe care l-am gasit la adresa: https://github.com/stleary/JSON-java).

Pe rand, pentru fiecare eticheta (employees, recruiters, users, managers) se citesc toate datele si se construieste fiecare tip de data. Apoi am construit reteaua sociala de mana, asa cum era prezentata in cerinta.

Exista si functia main care apeleaza startApplication pentru a rula algoritmul fara partea grafica.

Clasa Grafica:

Aceasta clasa este formata din 5 metode si main.

In main este creat frame-ul si este deschisa aplicatia, apelandu-se functia de startApplication si metoda MainPage care deschide pagina principala a componentei grafice.

MainPage contine 3 butoane din care pot fi alese urmatoarele pagini:

AdminPage – in pagina de admin apare o ierarhie ce contine intreaga aplicatie, afisand companiile, departamentele, angajatii si job-urile disponibile. Atunci cand se da click de un departament se deblocheaza butonul „Calculate” care calculeaza bugetul total al acelui departament.

ManagerPage – pagina de manager afiseaza mai intai o pagina SearchPage, unde se introduce numele unui manager pentru a intra. Aici sunt afisate request-urile disponibile, iar mangerul le poate da „Accept” sau „Deny”. Accept-ul este un fel de process, doar ca nu se mai face pentru toata lista odata, ci pentru fiecare request in parte.

ProfilePage – se afiseaza pagina de cautare la fel ca la manager, doar ca aici poate intra oricine. Odata introdus numele, se afiseaza resume-ul acelei persoane.

Toate paginile contin si buton de inapoi, ce intoarce la paginile anterioare.