GRIGORE Cosmin-Mitică

322CC

Tema POO

Dificultate: medie

Durată rezolvare: 50h sau mai mult, nu stiu exact, am avut si pauze.

Implementare: La functiile de tipul “add”, “remove”, “contains” m-am folosit in principiu de functiile implementate deja in clasele ArrayList/HashSet/TreeSet. Am folosit in majoritatea cazurilor ArrayList-uri tipizate in functie de caz. La metodele de genul getJobs (din Department), getEmployees, getCompanies am returnat direct obiectul cerut. La metoda getDegreeInFriendship din clasa Consumer am facut un bfs modificat, care contine un vector de parinti si un vector de distante , iar distanta o aflu pe principiul dist[i] = dist[parent[i]] + 1. La metoda getGraduationYear am iterat pana la obiectul Education care continea ciclul de licenta si am returnat data de sfarsit. In metoda meanGPA fac media tuturor ciclurilor de studiu care au data de final nenula. Pentru a retine mai multe obiecte de tip Education/Experience ordonate folosesc Colectia TreeSet. Pentru datele de inceput si de sfarsit din Education/Experience folosesc LocalDate, pentru a-mi fi usor sa compar datele. Cand creez un ciclu Education/Experience, verific daca data de final e mai mare decat data de inceput, altfel arunc exceptia InvalidDatesException. Metoda convert din User imi creeaza un Employee cu aceleasi informatii ca ale user-ului. In metoda getTotalScore din user, pentru a afla anii de experienta, creez in clasa Experience, o metoda months_of_experience care imi returneaza luniile de experienta dintr-un ciclu folosindu-ma de clasa Period din Java, returnata de metoda until din LocalDate, iar apoi impart la 12 si rotunjesc in functie de rest. In metoda evaluate, fac o cerere pe care o completez, iar apoi iau manager-ul companiei caruia ii trimit cererea completata si cresc rating-ul.

In metoda process din Manager fac un Treeset in care sortez descrescator dupa scor cererile(fac un Comparator pentru asta), iar apoi verific daca nu a fost angajat si daca respecta constrangerile, il angajez. In caz ca nu trece de una din conditii, ii trimit o notificare. Dupa terminarea cererilor, trimit notificare tuturor candidatiilor care au ramas neangajati si inchid job-ul. In metoda apply din Job, iau Recruiter-ul potrivit, evaluez User-ul si apoi il adaug in lista de candidati si observers. In metoda getRecruiter din Company ma folosesc de un vector de distante in care tin distanta de la fiecare Recruiter la User si il aleg pe cel mai indepartat, iar daca sunt mai multi, pe cel cu rating-ul maxim. In metoda getTotalSalaryBudget, pentru fiecare Departament, aplic o formula in functie de taxe.

GUI: Am facut un Meniu din care se poate intra in cele 3 moduri (Admin, Manager, Profile), accesul fiind realizat prin butoane. In modul Admin am create o fereastra care contine 2 JList-uri, una pentru useri si una pentru companii. La selectarea unei companii, butonul showDepartaments devine accesibil si deshide o alta fereastra care contine un JList cu departamentele, 3 butoane care devin accesibile cand selectam un department si putem afla bugetul

de salarii, create prin 2 label-uri, un JList cu angajatii si un JList cu job-urile din department (si cele deschise si ale angajatilor). Am folost modele default pentru a putea pastra modificarile.

Daca selectam modul Manager, se deschide o fereastra cu un obiecte de tip JList cu manageri din aplicatie, iar prin selectarea unuia, butonul Manager Page devine accesibil si ne trimite catre o alta fereastra care contine un obiect de tip JList cu cererile de anjagare si 2 butoane (accept si reject) care devin accesibile doar prin selectia unui element din lista de cereri. Prin apasarea butonului Accept, se intampla un algoritm asemanator ca cel din metoda process, doar ca pentru o singura cerere si fara criterii, iar in caz de reject, acesta este scos din lista de candidati, iar cererea este stearsa.

Daca selectam modul Profile ne deschide o fereastra in care avem un button, un JTextField in care putem adauga prenumele si numele User-ului pe care dorim sa il cautam si un JProgressBar care ne spune daca s-a gasit sau nu user-ul. Daca s-a gasit se deschide o fereastra noua in care se afla informatiile User-ului.

Observatii:

La clasele in care am facut unii membrii private, am facut Setter/Getter.

Cand se deschide o fereastra noua, cea veche devine invizibila, iar pentru a o face iar vizibila, adaug pentru fereastra noua un WindowListener particular, prin care fosta fereastra devine vizibila.

Pentru Parsarea fisierelor JSON am folosit org.json.simple.parser.JSONParser.

Pentru creare obiectelor de tip JList am folosit in general modele default pentru a putea pastra modificarile

Am creat constructori acolo un se impunea.

La Observer Pattern am mai adugat o metoda noua notifyObserver(Observer observer, Notification notification) pentru a notifica un singur Observer. Mi s-a parut mai ok ca atunci cand un user este respins sa il notific doar pe el.

Am folosit poze iar pentru path am folosit Absolute Path asa ca s-ar putea sa nu functioneze daca se schimba Path-ul si puteti modifica path-ul daca apare eroare.

Am facut 2 clase de testare. Clasa Test testeaza aplicatia, cu angajarea automata a userilor, iar Clasa StartApplication porneste Interfata Grafica