Ma numesc Petrica Ilie-Alex, grupa 321CC. Consider ca tema a avut un grad de dificultate mediu, dar a fost placuta intrucat am aplicat multe concepte de POO studiate la curs / laborator iar intr-un final am obtinut o aplicatie utilizabila multumita interfetei. Implementarea temei a durat aproximativ 2 saptamani, dar cu tot cu bonus, sunt multumit de ce am obtinut si consider ca a fost o tema foarte frumoasa.

La taskul1 toate clasele sunt implementate conform cerintei, am folosit constructori acolo unde am crezut eu ca este folositor, de asemenea gettere si settere. O sa mentionez de la acest task doar metodele care mi s-au parut mie mai importante si anume : metoda apply din clasa job, metoda compareTo din education(similara cu cea din experience), metoda process din clasa manager, metoda getRecruiter din clasa company, metoda getDegreeFriendship. Metoda getRecruiter din clasa company retine un flag in recruiter flag, pe care il vom folosii sa stim daca userul se afla pe „arborele de prieteni" al acestuia sau nu (adica au gradul 0 sau diferit de 0, caz in care luam gradul maxim). Practic pentru fiecare recruiter din lista aflu gradul de prietenie si il retin in variabila max. Daca recruiter flag este falee inseamna ca nu am recruiter cu grad de prietenie 0 pentru user, deci parcurg iar lista si cand dau de valoarea max am gasit recruiterul pe care ii adaug in lista de available recruiters(pe care o voi sorta ulterior dupa scor). In cazul in care recruiter flag este true inseamna ca am un recruiter cu grad de prietenie 0(cel mai ideal), deci parcurg vectorul si ii adaug in lista de available recruiters pe cei de acest gen. In final intorc recruiterul de la indexul 0 (recruiterul cel mai indepartat fata de user si cu cel mai mare scor). Metoda getDegreeinFriendship se comporta ca un bfs cu mentiunea ca retin intr-un hashmap care are key consumerul si value valoarea nivelului pe care ma aflu in arbore. Cand am gasit userul cautat in arborele de prietenie intorc value, daca nu apare in arbore se intoarce 0. Metoda process este cea mai complexa dar este viabila. Initial in new_noPositions retin job.noPositions pentru a retine numarul initial de joburi la intrarea procesarii(cate locuri disponibile am pe job, pe care il decrementez dupa fiecare user angajat). In lista goodRequests imi voi pune requesturile bune(cele pt jobul pe care il procesam din totalul requesturilor managerului). Le sortam folosind Collections.sort si clasa anonima comparator pentru a le sorta descrescator dupa score. Se intra in for de la 0 : job.noPositions – 1 ce isi propune sa angajeze atatia angajati cate goodrequesturi am. Pentru asta, folosim breakul in caz de nu mai avem requesturi bune(deci nu mai avem pecine sa angajam). Stergem requestul de la pasul i din lista de requesturi a managerului cat si din cea cu goodRequests, dupa care verificam daca nu cumva userul nostru nu s-a angajat intre timp la alt job/companie. Daca este inca user, il convertim la employee, cautam departamentul si jobul si il adaugam, setam salariul si compania de care apartine, decrementam new_noPositions(numarul real de pozitii deschise ramase), cat si adaugam un experience (cu data de start data curenta si de end null). Ulterior daca angajam trebuie sa eliminam requesturile de la alte companii cat si cele din compania la care aplica pentru alte joburi + eliminarea din observer. In cazul in care nu este gasit un user in lista de useri din Application se face job.noPositions ++ pentru a se marii iteratiile forului. Dupa iesirea din for se seteaza job.noPositions = new_noPositions pentru urmatorele procesari ale acestui job. Daca este 0, se sterg toate requesturile, se inchide si sterge jobul(moment in care se notifica observatorii ca jobul a fost inchis). Metoda compareTo din Education stabileste intai ce fel de comparatii se fac, daca amandoua au null(obiect apelant si obiect comparat) deci se compara crescator dupa data sau daca doar cea cu care se compara are enddate null caz in care le consider egale(pentru a nu se face permutari). In cazul in care ambele obiecte au null(obiect comparat si obiect apelant) le schimb intre ele(pentru a mentine ordinea crescatoare). In celelalte cazuri se compara daca au end_Date egal caz in care se compara crescator dupa start_date sau cazul clasic in care se compara descrescator dupa end_date. Metoda apply adauga utilizatorul care aplica la lista de

observer a companiei(daca nu exista deja), dupa care ii cauta recruiterul si se trimite requestul la manager daca indeplineste conditiile(se trimite requestul ce foloseste si evaluate). In cazul in care nu indeplineste conditiile minime este notificat imediat si considerat respins la jobul respectiv.

In cadrul taskului2 am curpins toata partea de testare. In primul rand vreau sa mentionez ca am folosit o metoda getDate care modifica data de intrare intr-o data acceptata de formatul LocalDate, pe care o folosesc in continutul aplicatiei. Apoi am creat companiile pe care le primesc dintr-un fisier text. Ulterior vom folosi doar fisiere json atat pentru consumers cat si pentru jobs si friends. In partea de testare doar imi adaug din json toate datele corespunzator aplicatiei cat si formez graful de prietenii dintre consumers.

In cadrul taskului3 am implementat toate DesigPatternurile cerute folosindu-ma de informatiile din breviarele laboratoarelor cat si de breviar in sine drept exercitiu. Am implementat Lazy Singleton, Observer Pattern, cu Subject Company si Observers Users, BuilderPattern pentru Resume si FactoryPattern pentru instantierea diferitelor tipuri de departamente.

In cadrul taskului4 am decis sa ma joc cu creativitatea mea. Mentionez ca toate clasele ce implementeaza se muleaza pe acelasi schelet, implementand diferite labeluri, textfielduri, diferite layouturi dupa nevoi, multa paneluri, butoane, comboBoxuri si scrollpaneuri. Astfel, esti intampinat de un meniu de autentificare ce iti cere usernameul si parola si iti da maxim 3 incercari dupa care aplicatia se inchide. Pentru logare se pot folosi doar admin-admin, user-user, manager-manager, implementate printr-un hashmap key value, deci se poate modifica usor posibilitatea accesarii aplicatiei. In cazul in care ai o logare valida intrii in pagina de meniu care are 5 butoane, pentru AdminPage, ManagerPage,ProfilePage, CompanyPage, NotificationPage. AdminPage arata intuitiv in partea de jos utilizatorii disponibili in aplicatie, iar in partea de sus campuri pentru salariu ale unui departament, ce se valideaza in momentul in care apesi pe departamentul corespunzator in Jtree. In partea central – Estica se gasesc detalii despre compania Amazon(angajati joburi) iar in partea centra-Vestica aceleasi detalii pentru compania Amazon. In ManagerPage initial trebuie sa selectezi compania pentru a putea vedea cine este managerul si a putea face ulterior modificari in compania ta(acceptand / respingand) utilizatori. Profile Page este o pagina intuitiva cu multe fielduri si comboboxuri in care introduci numele si prenumele(celelalte fielduri nu pot fi editate) si primesti toate informatiile despre acesta. CompanyPage este o pagina elaborata in care iconita reprezentativa companiei se schimba in functie de cea selectata. In aceasta apar toate detaliile aferente companiei si poti adauga noi recruiteri(se muta instant si lise atribuie rating 5, li se termina jobul curent si incep un nou job ca recruiter) si poti muta un emloyee dintr-un departament in altul. Intr-un final, NotificationPage este o pagina in care poti alege userul si ii apar toate notificarile pana in acel punct.