Temă proiect: Simulare a unei aplicații de găsit locuri de muncă

Autor: Andra-Maria Bordincel

Grupă: 325CC

Grad de dificultate: avansat

Durată aproximativă de realizare a temei temei: 5 zile

Implementare

- **Arhitectura aplicației:**

În implementarea mea am ales să lucrez folosind principiul încapsulării pentru toate clasele realizate, deși era necesar doar în cazul clasei **Information.**

- **Application:**
  Această clasă am implementat-o folosind Design Pattern-ul de tip Singleton, declarând un obiect de tip Application inițializat cu **null,** realizând un constructor privat și o metodă de getInstance(). Pe lângă acestea, am implementat metodele de **getCompanies()** – returnând lista de companii asociate aplicației, **getCompany(String name)** – în care iterez prin lista de companii înscrise și returnez compania cu numele transmis ca și parametru, **add(Company company)** – adaug în lista de companii, compania transmisă ca și parametru, **add(User user)** – adaug în lista de utilizatori ai aplicației utilizatorul transmis ca și parametru, **remove(Company company)** – elimin din lista de companii înscrise compania transmisă ca și parametru, dacă aceasta există, **remove(User user)** – elimin din lista de utilizatori asociați aplicației utilizatorul transmis ca și parametru, dacă acesta există, **getJobs(List<String> companies)** – declar o variabila de tip **ArrayList<Job>** pe care o returnez la final după adăugarea tuturor companiilor de care este interesat utilizatorul. Pentru această adăugare iterez prin toate companiile existente în aplicație și prin numele companiilor transmise ca parametru. În cazul în care exista o companie față de care utilizatorul prezintă interes, iterez prin toate Job-urile existente în aceasta și le adaug în variabila **jobs**. Pe lângă aceste metode, dat fiind faptul că am folosit principiul încapsulării, am realizat și metodele de get() și set() pentru câmpurile de date rămase.
- **Consumer:**
  - **Resume** – clasă internă în clasa abstractă Consumer, realizată cu Design Pattern-ul de tip Builder. La rândul său, aceasta are o clasă internă statică numită **ResumeBuilder.** Builder-ul creează părți ale obiectului complex de fiecare dată când este apelat și reține toate stările intermediare pentru a obține un mai ușor control asupra obiectului. Pe lângă aceasta, am implementat și metode de get și set pentru datele din clasă.

  Clasa abstractă Consumer cuprinde constructorii clasei și metodele **add(Education education)** – care accesează resume-ul obiectului și adaugă în lista de educații educația transmisă ca și parametru, **add(Experience experience)** – care accesează resume-ul obiectului și adaugă în lista de experiențe ale utilizatorului, experiența transmisă ca și

parametru, **add(Consumer consumer)** – care adaugă în lista de prieteni consumer-ul transmis ca şi parametru, **getDegreeInFriendship(Consumer consumer)** – îmi declar două variabile de tip **ArrayList<Consumer>,** prietenaşi – salvez lista de prieteni a obiectului, vizitaţi în care adaug pe rând elementele parcurse. Daca utilizatorul actual este acelaşi cu consumer-ul transmis ca şi parametru returnez gradul 0 pentru prietenie. Iterez apoi prin lista de prietenaşi pentru a vedea dacă am vizitat vreunul dintre ei, iar dacă nu este, îl adaug în lista de vizitaţi şi verific dacă se regăseşte consumer-ul transmis ca şi parametru, caz în care cresc gradul şi îl returnez. Dacă consumer-ul nu este găsit în nicio listă, returnez -1; **remove(Consumer consumer)** – elimin din lista de prieteni consumer-ul transmis ca şi parametru, **getGraduationYear()** – iterez prin lista de educaţii a resume-ului obiectului current şi verific dacă nivelul absolvit este cel de „college" returnez anul în care acesta s-a terminat; **meanGPA()** – iterez prin lista de educaţii din resume-ului şi salvez media în **var**, cresc contorul **cont**, şi adaug valoarea la sumă, apoi la final calculez media dintre suma obţinuă pe valoarea finală a contorului, returnând media obţinută. Pe lângă aceste metode, dat fiind faptul că am folosit principiul încapsulării, am realizat şi metodele de get() şi set() pentru câmpurile de date rămase.

- **Information:**
  Fiind o clasă implementată prin metoda încapsulării în aceasta am realizat doar constructorii clasei şi metodele de get() şi set().
- **Education:**
  În această clasă care extinde **Comparable** am implementat constructorii clasei, unul dintre aceştia aruncă excepţia **InvalidDatesException** în cazul în care datele calendaristice au fost introduse greşit, metodele de get() şi set() şi am suprascris metoda de **compareTo(Object object) –** efectuând o sortare descrescătoare după anul absolvirii nivelului de învăţământ, iar în caz că aceasta este încă în curs, vor fi sortate crescător după data de început.
- **Experience:**
  În această clasă care extinde **Comparable** am implementat constructorii clasei, unul dintre aceştia aruncă excepţia **InvalidDatesException** în cazul în care datele calendaristice au fost introduse greşit, metodele de get() şi set() şi am suprascris metoda de **compareTo(Object object) –** efectuând o sortare descrescătoare după anul absolvirii nivelului de învăţământ, iar în caz că aceasta este încă în curs, vor fi sortate crescător după numele companiei la care lucrează.
- **User:**
  În această clasă care extinde **Consumer** am implementat constructorii clasei, metodele de get() şi set() şi metodele **nrYearsOfExperience() –** în care iterez prin experienţele de lucru obţinute din resume, calculez numărul total de zile şi calculez anii lucraţi, **getTotalScore()** în care iniţializez numărul de ani lucraţi şi calculez scorul utilizatorului după formula dată.
- **Employee:**
  În această clasă care extinde **Consumer** am implementat constructorii clasei, metodele de get() şi set() şi metodele **nrYearsOfExperience() –** în care iterez prin experienţele de lucru obţinute din resume, calculez numărul total de zile şi calculez anii lucraţi.
- **Recruiter:**
  În această clasă care extinde **Consumer** am implementat constructoriul clasei – în care setez rating-ul la 5 şi adaug recruiter-ul în departamentul de IT, iterând prin departamentele companiilor din aplicaţie până la compania recruiter-ului, până în momentul în care găsesc departamentul IT, metodele de get() şi set().

- **Manager:**
  În această clasă care extinde **Employee** am implementat constructorul clasei şi metodele de get() şi set().
- **Job:**
  În această clasă am implementat constructorii clasei şi metodele de get() şi set().
- **Constraint:**
  În această clasă am implementat constructorii clasei şi metodele de get() şi set().
- **Company:**
  În această clasă am implementat constructorii clasei şi metodele **add(Department department)** – adaug în lista curentă de depatamente departamentul transmis ca şi parametru, **add(Recruiter recruiter)** – adaug în lista curentă de recruiteri recruiterul transmis ca şi parametru, **add(Employee employee, Department department)** – iterez prin departamentele mele şi daca unul dintre acestea este egal cu numele departamentului transmis ca şi parametru, adaug angajatul în acest departament, **remove(Employee employee)** – iterez în lista mea de departamente şi pentru fiecare extrag lista de angajaţi din care elimin angajatul transmis ca şi parametru, **remove(Department department)** – iterez prin lista de departamente şi în cazul în care unul din ele corespunde cu numele departamentului transmis ca şi parametru îl elimin, **remove(Recruiter recruiter)** – iterez prin lista de recruiteri şi in cazul în care unul dintre aceştia corespunde cu recruiter-ul transmis ca şi parametru îl elimin din listă, **move(Department source, Department destination)** – adaug departamentul destiniatie în lista mea de departamente şi apoi extrag lista angajaţilor din departamentul sursă, pe care îi adaug pe rând în departamentul destinaţie, urmând stergerea departamentului sursă din listă, **move(Employee employee, Department department)** – iterez prin lista mea de departamente şi prin cea de angajaţi ai fiecarui departament şi în cazul în care este găsit angajatul corespondent celui transmis ca şo parametru îl adaug în no-,ul departament şi îl şterg din cel vechi, **contains(Department department)** – parcurg lista de departamente şi verific dacă există cel transmis ca şi parametru, caz în care returnez true, iar dacă nu există, întorc false, **contains(Employee employee)** – iterez prin lista de departamente şi verific dacă în lista de angajaţi a fiecăruia există angajatul transmis ca şi parametru, caz în care întorc true, **contains(Recruiter recruiter)** – iterez prin lista de recruiteri şi verific dacă există şi cel transmis ca parametru, caz în care returnez ture, **getJobs()** – adaug lista de job-uri din fiecare departament în lista mea finală de job-uri pe care o returnez. Pe lângă acestea am implementat şi metodele de get() şi set().
- **Department:**
  Clasa abstractă Department cuprinde constructorul clasei, metoda abstractă **getTotalSalaryBudget()** şi metodele de **getJobs()** – întoarce lista de joburi dintr-un departament, **add(Employee employee)** – adaugă în lista de angajaţi ai departamentului angajatul transmis ca şi parametru, **remove(Employee employee)** – elimină din lista de angajaţi ai departamentului angajatul transmis ca şi parametru, **add(Job job)** – adaugă jobul transmis ca şi parametru în lista actuală de joburi a departamentului , **getEmployees()** – întoarce lista curentă de angajaţi ai departamentului.
- **IT:**
  Această clasă extinde **Department** şi implementează metoda **getTotalSalaryBudget()** unde se calculează suma tuturor salariilor angajaţilor din acest departament aplicând 0% impozit pentru fiecare salariu.

- **Management:**
  Această clasă extinde **Department** și implementează metoda **getTotalSalaryBudget()** unde se calculează suma tuturor salariilor angajaților din acest departament aplicând 16% impozit pentru fiecare salariu.
- **Marketing:**
  Această clasă extinde **Department** și implementează metoda **getTotalSalaryBudget()** unde se calculează suma tuturor salariilor angajaților din acest departament aplicând 10% impozit pentru fiecare salariu mai mare de 5000 de lei, 0% impozit pentru cele mai mici de 3000 de lei și 16% pentru celelalte rămase.
- **Finance**
  Această clasă extinde **Department** și implementează metoda **getTotalSalaryBudget()** unde se calculează suma tuturor salariilor angajaților din acest departament aplicând 10% impozit pentru fiecare salariu al angajaților cu mai puțin de 1 an de experiență de muncă și 16% impozit celorlalți angajați. Un punct pe care l-am identificat neclar în formularea acestui enunț a fost faptul că nu s-a specificat daca este necesară calcularea experientei totale de muncă a angajatului sau doar pe cea din cadrul companiei. Eu am ales să implementez metoda ținând cont de experiența totală.

Pe lângă aceste clase implementate mi-a fost necesară implementarea a încă două clase, anume **InvalidDatesException** – care extinde clasa **Exception** și afișează un mesaj prin care utilizatorului îi este cerut să verifice datele introduse; și clasa **DepartmentFactory** pe care o folosesc în implementarea Design Pattern-ului de tip Factory – aceasta are în componența sa o metodă ce primește ca argument tipul departamentului pe care trebuie să îl creeze și în funcție de acesta întoarce un obiect de tipul specific al departamentului**.**

- **Șabloane de proiectare:**

Dintre acestea am implementat în rezolvarea temei Design Pattern-urile de tip **Singleton** pentru clasa ***Application***, **Builder** pentru clasa ***Resume*** și **Factory** pentru ***Department***, în modurile detaliate în arhitectura aplicației.